https://doi.org/10.35662/unine-thesis-2876

Université de Neuchâtel
Faculté des Sciences
Institut d'Informatique

An Empirical Comparison of Recurrent Neural Network Models on Authorship Analysis Tasks

par

Nils Schaetti

Thèse

présentée à la Faculté des Sciences
pour l'obtention du grade de Docteur ès Sciences

Acceptée sur proposition du jury:

Prof. Jacques Savoy, Directeur de thèse
Université de Neuchâtel, Suisse

Dr. Francisco Manuel Rangel Pardo, rapporteur
Head of Product, Symanto, Allemagne

Dr. Valerio Schiavoni, rapporteur
Université de Neuchâtel, Suisse

Soutenue le 9 octobre 2020

UNIVERSITÉ DE NEUCHÂTEL

Faculté des Sciences
Secrétariat-décanat de Faculté
Rue Emile-Argand 11
2000 Neuchâtel – Suisse
Tél : + 41 (0)32 718 21 00
E-mail : secretariat.sciences@unine.ch

IMPRIMATUR POUR THESE DE DOCTORAT

La Faculté des sciences de l'Université de Neuchâtel autorise l'impression de la présente thèse soutenue par

Monsieur Nils Schaetti

Titre:

"An Empirical Comparison of Recurrent Neural Network Models on Authorship Analysis Tasks"

sur le rapport des membres du jury composé comme suit:

- Prof. Jacques Savoy, directeur de thèse, Université de Neuchâtel, Suisse
- Dr Francisco Rangel, Symanto Research, Nürnberg, Allemagne
- Dr Valerio Schiavoni, Université de Neuchâtel, Suisse

Neuchâtel, le 13 octobre 2020

Le Doyen, Prof. A. Bangerter

Abstract

In the last few years, a machine learning field named *Deep-Learning* (DL) has improved the results of several challenging tasks mainly in the field of computer vision. Deep architectures such as *Convolutional Neural Networks* (CNN) have been shown as very powerful for computer vision tasks. For those related to language and timeseries the state of the art models such as *Long Short-Term Memory* (LSTM) have a recurrent component that take into account the order of inputs and are able to memorise them. Among these tasks related to Natural Language Processing (NLP), an important problem in computational linguistics is *authorship attribution* where the goal is to find the true author of a text or, in an *author profiling* perspective, to extract information such as gender, origin and socio-economic background.

However, few work have tackle the issue of authorship analysis with recurrent neural networks (RNNs). Consequently, we have decided to explore in this study the performances of several recurrent neural models, such as *Echo State Networks* (ESN), LSTM and *Gated Recurrent Units* (GRU) on three authorship analysis tasks. The first one on the classical *authorship attribution* task using the Reuters C50 dataset where models have to predict the true author of a document in a set of candidate authors. The second task is referred as *author profiling* as the model must determine the gender (male/female) of the author of a set of tweets using the PAN 2017 dataset from the CLEF conference. The third task is referred as *author verification* using an in-house dataset named SFGram and composed of dozens of science-fiction magazines from the 50s to the 70s. This task is separated into two problems. In the first, the goal is to extract passages written by a particular author inside a magazine co-written by several dozen authors. The second is to find out if a magazine contains passages written by a particular author. In order for our research to be applicable in authorship studies, we limited evaluated models to those with a so-called *many-to-many* architecture. This fulfills a fundamental constraint of the field of stylometry which is the ability to provide evidences for each prediction made.

To evaluate these three models, we defined a set of experiments, performance measures and hyperparameters that could impact the output. We carried out these experiments with each model and their corresponding hyperparameters. Then we used statistical tests to detect significant differences between these models, and with state-of-the-art baseline methods in authorship analysis.

Our results shows that shallow and simple RNNs such as ESNs can be competitive with traditional methods in authorship studies while keeping a learning time that can be used in practice and a reasonable number of parameters. These properties allow them to outperform much more complex neural models such as LSTMs and GRUs considered as state of the art in NLP. We also show that pretraining word and character features can be useful on stylometry problems if these are trained on a similar dataset. Consequently, interesting results are achievable on such tasks where the quantity of data is limited and therefore difficult to solve for deep learning methods. We also show that representations based on words and combinations of three characters (*trigrams*) are the most effective for this kind of methods. Finally, we draw a landscape of possible research paths for the future of neural networks and deep learning methods in the field authorship analysis.

Keywords— Authorship attribution, author profiling, author verification, deep learning, recurrent neural networks, neural networks, echo state network, reservoir computing, long short-term memory, natural language processing, computational linguistics, machine learning

Résumé

Au cours des dernières années, un domaine de l'apprentissage automatique nommé *Apprentissage Profond* (Deep-Learning (DL)) a permis d'importantes améliorations sur plusieurs tâches difficiles principalement dans le domaine de la vision par ordinateur. Des modèles neuronaux profonds tels que les *réseaux convolutifs profonds* (Convolutional Neural Networks (CNN)) se sont révélés très puissants pour les tâches de reconnaissance d'objets ou de segmentation d'images. Concernant le traitement des langues naturelles et les séries temporelles, certains modèles profonds tels que les *Long Short-Term Memory* (LSTM) ont une composante récurrente qui prend en compte l'ordre des entrées et sont capables de mémoriser des données pour une période de temps fini. Parmi ces tâches liées au traitement du langage naturel, un problème important de la linguistique computationnelle est l'*attribution d'auteur* où le but est de trouver le véritable auteur d'un texte ou, dans une perspective de *profilage d'auteur*, d'extraire des informations à son sujet telles que le sexe, l'origine et la situation socio-économique.

Cependant, peu de travaux ont evalué des modèles neuronaux récurrents (RNNs) sur des tâches d'analyse d'auteur. Par conséquent, nous avons décidé d'explorer dans cette thèse les performances de plusieurs RNNs, tels que les *Echo State Networks* (ESN), les LSTMs et les *Gated Recurrent Units* (GRU) sur trois tâches en analyse d'auteur. La première concerne le problème d'*attribution d'auteur* utilisant le jeu de données Reuters C50 où le but est de prédire le véritable auteur d'un document dans un ensemble fini d'auteurs possibles. La deuxième tâche est appelée *profilage d'auteur*, car le modèle doit déterminer le sexe (homme/femme) de l'auteur d'un ensemble de tweets. Pour cela nous avons utilisé le jeu de données PAN 2017 de la conférence CLEF. La troisième tâche est appelée *vérification d'auteur* et basée sur un jeu de données nommé SFGram, créé pour l'occasion, et composé de dizaines de magazines de science-fiction des années 50 aux années 70. Cette tâche est séparée en deux problèmes. Dans le premier, le but est d'extraire les passages écrits par un auteur particulier à l'intérieur d'un magazine coécrits par plusieurs dizaines d'auteurs. Dans le deuxième, il s'agit de trouver si un magazine contient du texte écrit par un auteur particulier. Afin que nos recherches soient applicables en analyse d'auteur, nous avons restreint les modèles testés à ceux ayant une architecture dite *many-to-many* capables de fournir une prédiction pour chaque ensemble du texte analysé. Ceci permet de remplir la contrainte fondamentale de l'analyse d'auteur qu'est la capacité à fournir des évidences pour chaque prédiction faites. Pour évaluer ces trois modèles, nous avons défini un ensemble d'expériences, de mesures de performance et d'hyperparamètres qui pourraient influencer les performances de ces méthodes. Nous avons ensuite effectué ces expériences avec chaque modèle et leurs hyperparamètres correspondants. Puis nous avons utilisé des tests statistiques afin de détecter des différences significatives entre ces modèles, et avec les méthodes de référence en analyse d'auteur.

Nos résultats montrent que des RNNs et particulièrement simples tels que les ESNs peuvent être compétitifs avec les méthodes traditionnelles en analyse d'auteur tout en gardant un temps d'apprentissage utilisable en pratique et un nombre de paramètres raisonnable. Ces propriétés leur permettent de surpasser des modèles neuronaux beaucoup plus complexes comme les LSTMs et les GRUs considérés comme l'état de l'art en traitement du langage naturel. Nous montrons également que pre-entraîner des représentations de mots et de caractères peut être utile sur des problèmes en analyse d'auteur si celles-ci sont entraînées sur un jeu de données similaire. Ceci permet d'obtenir des résultats intéressants sur des problèmes où la quantité de données est limitée et donc difficile à résoudre pour des méthodes d'apprentissage profond. Nous montrons également que les représentations basées sur les mots et les combinaisons de trois caractères (*trigrams*) sont les plus efficaces pour ces types de modèles. Pour finir, nous dessinons un paysage des voies de recherche et d'applications possibles des réseaux de neurones et des méthodes d'apprentissage profond en analyse d'auteur.

Mots clés— Traitement automatique du langage naturel, linguistique computationnelle, attribution d'auteur, classification automatique, apprentissage automatique, réseaux de neurones artificielles, réseaux de neurones récurrents, profilage d'auteur, apprentissage profond

Contents

List of Figures

List of Tables

List of Publications

Notebooks

1. Schaetti, N. (2017). *UniNE at CLEF 2017: TF-IDF and Deep-Learning for Author Profiling.* In Proceedings of the Eight International Conference of the CLEF Association (CLEF 2017);

2. Schaetti, N. *UniNE at CLEF 2018: Bidirectional Echo State Network-based Reservoir Computing for Cross-domain Authorship Attribution.* In Proceedings of the Ninth International Conference of the CLEF Association (CLEF 2018);

3. Schaetti, N. (2018, September). *UniNE at clef 2018: Character-based convolutional neural network and resnet18 for twitter author profiling.* In Proceedings of the Ninth International Conference of the CLEF Association (CLEF 2018);

4. Schaetti, N. (2018). *UniNE at CLEF 2018: Character-based Convolutional Neural Network for Style Change Detection.* In Proceedings of the Ninth International Conference of the CLEF Association (CLEF 2018);

Conference papers

1. Schaetti, N., Savoy, J. (2018). *Comparison of Neural Models for Gender Profiling.* In Proceedings of the 14th international conference on statistical analysis of textual data (Jun 2018);

2. Schaetti, N. (2019). *Author Verification in Stream of Text with Echo State Network-based Recurrent Neural Models.* In Proceedings of the Fourth Swiss Text Analytics Conference;

3. Schaetti, N. (2019). *Behaviors of Reservoir Computing Models for Textual Documents Classification.* International Joint Conference on Neural Networks (IJCNN 2019);

Articles

1. Schaetti, N. *Comparison of Visualisable Evidence-based Authorship Attribution Methods using Recurrent Neural Networks.* Journal of Information Processing and Management (IPM). In the process of publication;

List of Abbreviations

AA	Authorship Attribution
AD	Author Diarization
AE	Author Extraction
AI	Artificial Intelligence
AID	Author Identification
AP	Author Profiling
APRL	Atiya-Parlos Recurrent Learning
AV	Author Verification
AVMA	Author Verification in Multi-Authored Documents
BOW	Bag-Of-Words
BPDC	BackPropagation-DeCorellation
BPTT	BackPropagation Through Time
C1	Character Unigrams
C2	Character Bigrams
C3	Character Trigrams
CNN	Convolutional Neural Network
DL	Deep Learning
ESN	Echo State Network
FFNN	Feedforward Neural Network
FW	Function words
GP	Gender Profiling
GPU	Graphic Processing Units
GRNN	Generalized Regression Neural Network
GRU	Gated Recurrent Unit
IF	IF Science-Fiction Magazine
LM	Language models
LSA	Latent Semantic Analysis

LSTM	Long Short-Term Memory
LVQ	Learning Vector Quantization
ML	Machine Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NER	Named Entities Recognition
NLP	Natural Language Processing
NMT	Neural Machine Translation
OCR	Optical Character Recognition
OOV	Out-Of-Vocabulary
PCA	Principal Component Analysis
POS	Part-Of-Speech
RC	Reservoir Computing
RNN	Recurrent Neural Network
RR	Ridge Regression
SCD	Style Change Detection
SGD	Stochastic Gradient Descent
SMS	Short Message Service
SOM	Self-Organizing Maps
SVD	Singular Value Decomposition
SVM	Support Vector Machine
TDS	Text Deconvolution Saliency
TTR	Type-Token-Ratio
WV	Word Embedding
WWW	World Wide Web

Notation

Sign	Description	Size/Space	First appearance/ definition
n_c	Number of units in the LSTM memory cell	$\mathbb{N}$	pages 39, 41
n_e	Size of the embedding space	$\mathbb{N}$	page 46
n_h	Number of units in an RNN hidden layer	$\mathbb{N}$	page 33
n_q	Size of the context vector (attention mechanism) or the number of tokens on each side (Word2Vec model)	$\mathbb{N}$	page 42
n_s	How many states taken into account for state-gram	$\mathbb{N}$	page 79
n_x	Number of input units	$\mathbb{N}$	pages 34, 36
n_y	Number of output units	$\mathbb{N}$	page 34
A	The set of candidate authors in an authorship attribution task	$\lvert A\rvert$	page 20
$\lvert A\rvert$	The number of candidate authors in the set A	$\mathbb{N}$	page 20
$E(\boldsymbol{Y}, \hat{\boldsymbol{Y}})$	Error between target signal $\boldsymbol{Y}$ and model output $\hat{\boldsymbol{Y}}$	$\mathbb{R}$	page 33
$\boldsymbol{H}$	Matrix containing all hidden states of a RNN resulting from samples in the training set	$n_h \times \tau$	page 39
$\boldsymbol{H}^{(i)}$	Matrix containing all hidden states of a RNN resulting from the i-th sample. The activation vector of the hidden layer at time t corresponds to the t-th column in matrix $\boldsymbol{H}^{(i)}$ ($\boldsymbol{h}^{(i)}(t) = \boldsymbol{H}^{(i)}_{:,t}$)	$n_h \times \tau^{(i)}$	page 39
$\boldsymbol{h}(t)$	Activation vector of an RNN hidden layer at time t. Each component is referred as $h_1(t), h_2(t), \ldots, h_{n_h}(t)$	n_h	page 36
$R_i(x)$	Relevance score of the i-th feature in input x	$\mathbb{R}$	page 18
τ	Length (time) of a sample	$\mathbb{R}$	page 35
$\tau^{(i)}$	Length (time) of the i-th sample	$\mathbb{R}$	page 35
$\boldsymbol{W}^e$	The embedding matrix projecting the inputs space into the embedding space	$n_e \times n_x$	page 46
w_{ij}	Weight between the i-th and j-th neurons	$\mathbb{R}$	page 39
$\boldsymbol{X}$	The $n_x \times \tau$ matrix containing all input examples	$n_x \times \tau$	page 35
$\boldsymbol{X}^{(i)}$	The $n_x \times \tau^{(i)}$ matrix containing the i-th input example of length $\tau^{(i)}$ and input size n_x	$n_x \times \tau^{(i)}$	page 35
$x^{(i)}(t)$ or $\hat{\boldsymbol{x}}^{(i)}(t)$	Input vector at time t. If the input signal is a vector, each component is referred as $\boldsymbol{x}(t) = [x_1(t), x_2(t), \ldots, x_{n_x}(t)]$	$\mathbb{R}$ or n_x	page 35
$\boldsymbol{Y}$	The $n_y \times \tau$ matrix containing the target signal to be learned of length τ and size n_y	$n_y \times \tau$	pages 33, 39
$\hat{\boldsymbol{Y}}$	The $n_y \times \tau$ matrix containing the model's output signal of length τ and size n_y	$n_y \times \tau$	pages 33, 36
$\hat{\boldsymbol{Y}}^{(i)}$	The $n_y \times \tau^{(i)}$ matrix containing the model's output signal resulting of the i-th sample of length $\tau^{(i)}$ and size n_y. The vector $\hat{\boldsymbol{y}}^{(i)}(t)$ corresponds with the t-th column of this matrix ($\hat{\boldsymbol{y}}^{(i)}(t) = \hat{\boldsymbol{Y}}^{(i)}_{:,t}$)	$n_y \times \tau^{(i)}$	page 36

Sign	Description	Size/Space	First appearance/ definition
$\hat{y}^{(i)}(t)$ or $\hat{\boldsymbol{y}}^{(i)}(t)$	Model output signal at time t for the i-th sample. If the output signal is a vector, each component is referred as $\hat{\boldsymbol{y}}(t) = [\hat{y}_1(t), \hat{y}_2(t), \ldots, \hat{y}_{n_y}(t)]$	$\mathbb{R}$ or n_y	page 33
$\boldsymbol{Y}^{(i)}$	The $n_y \times \tau^{(i)}$ matrix containing the target signal to be learned for the i-th sample of length $\tau^{(i)}$ and size n_y. The vector $\boldsymbol{y}^{(i)}(t)$ corresponds with the t-th column of this matrix ($\boldsymbol{y}^{(i)}(t) = \boldsymbol{Y}^{(i)}_{:,t}$)	$n_y \times \tau^{(i)}$	page 39
$y^{(i)}(t)$ or $\boldsymbol{y}^{(i)}(t)$	Target output signal to be learned at time t for the i-th sample. If the target signal is a vector, each component is referred as $\boldsymbol{y}(t) = [y_1(t), y_2(t), \ldots, y_{n_y}(t)]$	$\mathbb{R}$ or n_y	page 33
α	Leak rate controlling the dynamics of an ESN	$\mathbb{R}$	page 36
$\boldsymbol{b}$	Bias vector of ESN reservoir units	n_h	page 36
$\boldsymbol{\epsilon}(t)$	Error term at time t. Each component is referred as $\epsilon_1(t), \epsilon_2(t), \ldots, \epsilon_{n_h}(t)$	n_h or n_x	page 36
λ	Regularisation parameter of the Ridge Regression (RR)	$\mathbb{R}$	page 39
$\rho(\boldsymbol{W})$	Spectral radius of matrix $\boldsymbol{W}$	$\mathbb{R}$	page 38
$\boldsymbol{U}$	RNN input weights matrix	$n_h \times n_x$	page 36
$\boldsymbol{v}(t)$	Noise vector at time t. Each component is referred as $v_1(t), v_2(t), \ldots, v_{n_h}(t)$	n_h or n_x	page 35
$\boldsymbol{W}$	ESN internal weights matrix	$n_h \times n_h$	page 33
$\boldsymbol{W}^{out}$	RNN trainable output weights matrix	$n_y \times n_h$	page 37
$\boldsymbol{a}(t)$	Attention weights computer by normalising score vector $\boldsymbol{sc}(t)$ which allow models to explicitly focus on certain parts of the input	n_q	page 42
$\boldsymbol{b}^f$	Bias vector of the forget gate (LSTM)	n_c	page 40
$\boldsymbol{b}^i$	Bias vector of the input gate (LSTM)	n_c	page 39
$\boldsymbol{b}^o$	Bias vector of the output gate (LSTM)	n_c	page 40
$\boldsymbol{c}(t)$	State of the memory cell at time t (LSTM). Each component is referred as $c_1(t), c_2(t), \ldots, c_{n_c}(t)$	n_c	page 39
$\boldsymbol{f}(t)$	State of the forget gate at time t (LSTM). Each component is referred as $f_1(t), f_2(t), \ldots, f_{n_c}(t)$	n_c	page 39
$\boldsymbol{i}(t)$	State of the input gate at time t (LSTM). Each component is referred as $i_1(t), i_2(t), \ldots, i_{n_c}(t)$	n_c	pages 39, 43
$\boldsymbol{o}(t)$	State of the output gate at time t (LSTM). Each component is referred as $o_1(t), o_2(t), \ldots, o_{n_c}(t)$	n_c	page 40
$\boldsymbol{q}(t)$	Context vector computed as a weighted sum of past hidden states $\boldsymbol{h}(t - n_q), \boldsymbol{h}(t - n_q + 1), \ldots, \boldsymbol{h}(t)$	n_h	page 42
$\boldsymbol{S}$	Score matrix learning a score for each state in the context window	$n_q \times n_h$	page 42
$\boldsymbol{sc}(t)$	Score vector learned during training encoding which past state $\boldsymbol{h}$ is valuable to solve the task	n_q	page 42
$\boldsymbol{s}(t)$	Attention vector used to compute the learned output $\hat{\boldsymbol{y}}$	n_h	page 42
$\boldsymbol{U}^c$	LSTM input-to-cell weights matrix	$n_c \times n_x$	page 40
$\boldsymbol{U}^f$	LSTM forget gate matrix controlling the flow of information remaining in the memory cell $\boldsymbol{c}(t)$ at time t based on inputs $\boldsymbol{x}(t)$	$n_c \times n_x$	page 39
$\boldsymbol{U}^i$	LSTM input gate matrix controlling the flow of information going into the memory cell $\boldsymbol{c}(t)$ from the inputs $\boldsymbol{x}(t)$ at time t based on inputs $\boldsymbol{x}(t)$	$n_c \times n_x$	page 39

Sign	Description	Size/Space	First appearance/ definition
$\boldsymbol{U}^o$	LSTM output gate matrix controlling the flow of information going out of the memory cell $\boldsymbol{c}(t)$ to the hidden layer $\boldsymbol{h}(t)$ at time t based on inputs $\boldsymbol{x}(t)$	$n_c \times n_x$	page 40
$\boldsymbol{W}^c$	LSTM hidden-to-cell weights matrix	$n_c \times n_h$	page 40
$\boldsymbol{W}^f$	LSTM forget gate matrix controlling the flow of information remaining in the memory cell $\boldsymbol{c}(t)$ at time t based on the state of the hidden layer	$n_c \times n_h$	page 39
$\boldsymbol{W}^i$	LSTM input gate matrix controlling the flow of information going into the memory cell $\boldsymbol{c}(t)$ from the inputs $\boldsymbol{x}(t)$ at time t based on the state of the hidden layer	$n_c \times n_h$	page 39
$\boldsymbol{W}^o$	LSTM output gate matrix controlling the flow of information going out of the memory cell $\boldsymbol{c}(t)$ to the hidden layer $\boldsymbol{h}(t)$ at time t based on the state of the hidden layer	$n_c \times n_h$	page 40
$\boldsymbol{b}^r$	Bias vector of the reset gate (GRU)	n_h	page 43
$\boldsymbol{b}^z$	Bias vector of the update gate (GRU)	n_h	page 43
$\boldsymbol{r}(t)$	State of the reset gate at time t (GRU)	n_h	page 43
$\boldsymbol{U}^r$	GRU reset gate matrix controlling the flow of information remaining in the hidden layer $\boldsymbol{h}(t)$ at time t based on inputs $\boldsymbol{x}(t)$	$n_h \times n_x$	page 43
$\boldsymbol{U}^z$	GRU update gate matrix controlling the flow of information going into the hidden layer $\boldsymbol{h}(t)$ from the inputs $\boldsymbol{x}(t)$ at time t based on inputs $\boldsymbol{x}(t)$	$n_h \times n_x$	page 43
$\boldsymbol{W}^z$	GRU update gate matrix controlling the flow of information going into the hidden layer $\boldsymbol{h}(t)$ from the inputs $\boldsymbol{x}(t)$ at time t based on the state of the hidden layer	$n_h \times n_h$	page 43
$\boldsymbol{z}(t)$	State of the update gate at time t (GRU)	n_h	page 43

Part I

Theory, methodology and implementation

> Art works because it appeals to certain faculties of the mind. Music depends on details of the auditory system, painting and sculpture on the visual system. Poetry and literature depend on language.
>
> Steven Pinker

The field of Authorship Analysis (stylometry) consists to find the true author of a textual document or some of its characteristics. Originally approached from the angle of statistics and lexical features in a computer-assisted way up to the 90s, the state-of-the-art methods use since the early 2000s methods from machine learning (ML) and artificial intelligence (AI). Since 2010, some researchers explored deep learning (DL) models with multiple levels of abstraction mainly applied to social network data. However, an extensive analysis of neural network models applied to authorship analysis is still missing. This thesis is a comparison a different flavours of Recurrent Neural Network (RNN) models on three Authorship Analysis tasks referred as Author Profiling (AP), Authorship Attribution (AA) and Author Verification (AV). The purpose of this study is to compare three different architectures, the first based on the Reservoir Computing (RC) paradigm, and the other two on gated networks (LSTM and GRU), all with various kind of shallow and deep features, and to evaluate their relative performances. As required by the field of author analysis, we have limited our choice to specific architectures referred as "many-to-many" which allow us to extract and visualise evidences based on their predictions.

The first part of this document will introduce the theoretical and practical aspect of the field of Authorship Analysis by defining the mathematical and formal basis of this field of research so that this document can be autonomous. This part will define the state of the art of current research by introducing the necessary concepts, features and measures used in this field. Chapter 1 will define the context and the purposes of this thesis. Chapter 2 will introduce the methodology and data sets used to evaluate different models. Chapter 3 will formally introduce the different flavours of RNNs used in this study, Echo State Networks-based (ESN) Reservoir Computing (RC), Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU), along with mathematical bases of these models and their hyperparameters. This chapter will also introduce different features based on syntactic and lexical representations used alongside models. The goal of this first part is to give to the reader all the necessary information about the methodology used in this study, and the concepts required to understand how models work.

The second part will focus on the experimentation and application of RNN models to the three tasks in order to evaluate different strengths and weaknesses of these models and which feature works best for each model. The performances will be evaluated on the same basis and measures and will be compared to state-of-the-art methods in this field. Chapter 4 will present results obtained on the Authorship Attribution (AA) tasks with the Reuters C50 dataset at the document and sentence level. In chapter 5 we will present results obtained on the Author Profiling (AP) task with the PAN17 dataset. Finally, chapter 6 will present results on the Author Verification (AV) tasks using the home-made dataset SFGram. This task is split in two, we will refer to the first as Author Extraction (AE) where the goal is to extract from a document passages written by a target author. We will refer to the second as Author Verification in Multi-Authored Documents (AVMA) where the goal is to determine if text written by a target author is present in a multi-authored document.

The last part is composed of Chapter 7. In this last chapter we will present, first, a broad comparison of the different models and features on the three tasks, and secondly, the main conclusions of this study by identifying the theoretical and practical issues of each model. Finally, we will conclude with the possible future axis for research aimed at applying deep learning and neural network methods to the field of author analysis and the current and future challenges that these models have to face.

Chapter 1

Introduction

1.1 Context

With its remarkable ability to use language, *homo sapiens* can exchange ideas and shape events in each other's mind. A language connect people into communities and allow us to share information, solve problems, build groups and share a common culture. Oral communication is a powerful tool to share ideas and organise human behaviours but is highly dependent on the memories and interpretations of people, transforming, deforming and losing information through generations. Language became a much more powerful tool around 3300 BC in Mesopotamia (current Irak) when writing was invented in the form of cuneiform script. Around 2600 BC, writing started to be used not only for administrative purposes but to keep track of history and to share mythical stories. The written forms of language rises new issues and possibilities. As language is put in writing form, the information it contains is transmitted much more faithfully, but the name of the authors can also be lost, kept secret or hidden willingly. The Epic of Gilgamesh, the earliest surviving great work of literature, is a good example as the name of the authors are still unknown. But this is a recursive fact through history as the authors of many work of literature since then are unknown or done under a pen name.

But the power of the writing form was dependent on the evolution of its support which undergoes two other revolutions. The first attempt to improve the manufacturing process happened in China between 300 and 600 A.D with the introduction of woodcut characters as stamps, and in Europe with the appearance of the amanuensis, a class of professional charged to reproduce written documents. However, at the end of the 14th century, 37 days of work was still needed for a 250 pages document. The second breakthrough that facilitated printing techniques was invented in 1452 by Johannes Gutenberg. His invention made of mobile lead characters precociously increased the productivity in printing. As a major event of the Renaissance, it was determining in the diffusion of texts and knowledge. As a result, the productivity in book manufacturing and paper making grew by more than twenty-fold in the following years.

However, the transition from oral tradition during the Middle-Age to the written tradition and printing allowed authors to live from their pen by signing their texts, but also to spread information and ideas while hiding their identity, to write under a pen name, or to grab someone's work. This last example is known as *plagiarism* and created lots of controversies between great authors during the 17th century as schools were teaching methods of plagiarism. In the next centuries, plagiarism remained an issue even with the introduction of law considering it as an offence. Hidden identities and pen names are also a problem for historians and humanists as many authors or contributors of famous piece of arts, books or pamphlets are not known. Such as *Beowulf*, an epic Anglo-Saxon poems written at the end of the first millennium by one or more anonymous authors. Furthermore, pen names were a form of choice during centuries for authors facing repression or social reprobation for their ideas. Anti-religious treaties such as *The Life of Lazarillo de Tormes* (1554) or the *Theophrastus redivivus* (1600 to 1700) were written anonymously or under a pen name. In the 18th century, famous work such as *An Essay on the Principle of Population* (1798) by T. R. Malthus, *Frankenstein* (1818) by Marry Shelley, or *Common Sense* (1776) by the philosopher Thomas Paine were first published anonymously.

This profusion of anonymous work and pen names put a shadow on the authorship of many work in literature or political discourse. For many years, scholars attempted to determine the authorship of William Shakespeare's canon, the possible collaboration and the identity of these collaborators. Many plays were added to Shakespeare's canon through years and some are still under scrutiny. The *Federalist Papers* (1787-

1788) is a collection of 85 articles written by three Founding Fathers of the United States, Alexander Hamilton, James Madison and John Jay, under a pseudonym (Publius) to encourage the adoption of the US Constitution. Writing and printing allowed everyone to communicate with people living in different and distant part of the world but it also allowed poorly cared people to use these technologies and anonymity for criminal acts such as *threatening letters*, *hate mail*, *terrorist claims*, *ransom notes* and the spreading of *hate speeches*.

The middle of the 20th century saw the appearance of a new revolution of its support. In the 50s, scientists and engineers developed concepts of communication networks through electronic devices and created practical application between the 60s and the 70s. These decades saw the development of the first modem, packet switching and the first global computer network between universities. At the end of the 80s, more than 100,000 computers were connected to this global network renamed *Internet*. In late 80s, a physician at the CERN in Geneva created the *World Wide Web*, a system of document linked through hypertext and organised as a web. In 2014 more than 1 billions of web sites were created on the Web and the Internet gave us many new ways to communicate. E-mails quickly replaced traditional mails and allowed us to exchange message in seconds, web sites allowed the creation of forums where thousands or even millions of users exchange ideas, concepts and information. Blogs allowed everyone to publish on a regular base personal articles on a large variety of subjects. In 2011, there were more than 150 millions blogs and 1 million articles published each day. Instant messaging offered real-time text transmission over the Internet between two parties (IM) or multiple parties (Chatrooms). Since the end of the 2000s, the Web saw the appearance of social networks and social medias linking their users and their social relationships to offer personalised contents and information sharing. These technologies increased the amount of information shared between humans on a scale never seen before. This new era surcharged the use of information to guide every other technology and induced an exponential improvement of the field of communication technology itself. With these new possibilities came new ways for people or groups to use communication channels to influence, deceive, threaten while hiding their true identity. E-mails and instant messaging could be used to send hate mails and harassing messages anonymously. Blogs and forums could also be used to spread hate speeches and conspiracy theories much faster. Recently, social medias introduced the possibilities to post fake information but also to perform user profiling in order to influence the way we make important decisions such as choosing who we want to vote for or what we want to buy.

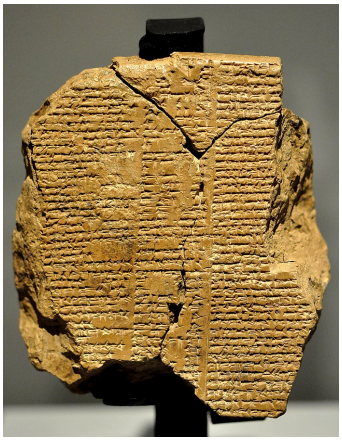


Figure 1.1 – Epic of Gilgamesh

A phenomenon referred as *fake news* took a singular magnitude in 2016 with the Brexit and the US election. Fake news are articles, news or post on social medias with catchy headlines and completely manufactured fake elements designed to mimic real news sites, in order to misinform and take financial advantages or discredit public persons. They are the creation of a single person or a group, or sometimes governments, using non-institutional medias, blogs and social networks. Social medias allow everyone to become a source of information and give anonymity which prevent prosecution for calumny and do not subject authors to the deontology of classical journalism. The phenomenon hit mainly the political life with a peak in 2018 when it was revealed that Cambridge Analytica, a British company, participated to the 2016 US election by collecting and exploiting, through targeted ads, the data of 82 millions US citizens to influence their voting choice. But fake news also affect the medical community as health is a huge market where information can easily becoming viral. As a result, an industry of medical fake news quickly appeared promoting miracle cures against health issues such as obesity, cancer or complex decease, or spreading anti-scientific ideas and false medical advices. A lot of technical solution are explored which include fact-checking software, penalisation of medias spreading fake news by social network companies, laws against misinformation or software packages to identify compromised and fake accounts.

The use of new communication technologies to influence audiences has reached a new heights in recent years with the phenomenon of *Astroturfing*. A technique of manual or algorithmic propaganda used for advertising or political purposes. Its name is derived from AstroTurf, a brand of artificial grass. Astroturfing consists to simulate spontaneous popular movements and can take the form of software packages multiplying fake accounts on social medias, by creating fake blog or posting fake reviews while trying to dissimulate their true identity or the fact that multiple accounts are use by a same author or organisation. It is used by a single person or by organised groups with the intention of misinforming or to make the opinion favourable

to them, and to give the impression of a majority to justify a political position. Software packages can be used to attack other users, post comment about competitors and fake identities in one personal management system. Detection systems use profile information such as images, behavioural patterns or textual data to identify the true identity of the author [1]. As with new communication technologies come the need for new methods to process the large amount of textual data and extract meaningful information.

The first theoretical and practical attempts to automate natural language processing started in the 1950s with Alan Turing's article "Computing Machinery and Intelligence" [2]. He introduced the most famous intelligence test named the *Turing test* where a computer has to fool a human in a text-based conversational situation. In 1957, Noam Chomsky introduced the *universal grammar* [3], a rule-based system of syntactic structures. However, in 1966, the ALPAC report [4] showed that 10 years of research failed to fulfil the expectations for automated language processing devices, which drastically reduced funding. This showed the difficulties faced when we try to use computers to solve a broad range of tasks involving natural languages. These tasks and the study of methods to solve them is known as *Natural Language Processing* (NLP). Computers are incredibly powerful to compute *pi* with a lot of numbers after the comma or to prove simple theorem, and as a result early artificial intelligence practitioner thought that hard problem such as language understanding would be solved quickly. But 50 years of in Artificial Intelligence (AI) research teached us a lesson : language-related tasks are part of these so-called hard problems which are very easy for humans but incredibly difficult for computers.

Before the late 1980s, natural language processing system were based on complex sets of handwritten rules, but in the 1990s researchers introduced machine learning with algorithms not based on human expertise but able to learn a task directly from data. This development was made possible by two trends. On one hand, the increasing computational power available to us since the beginning of computer science. On the second hand, the availability of large "real-text" multilingual corpora directly available from the Web that are necessary for machine learning algorithms to be efficient. Early methods were based on rule-systems and decision trees, but since the early 90s researcher have increasingly focused on statistical models. From 2010s, a breakthrough technology in the field of machine learning known as *deep learning* allowed huge advances in the field of computer vision [5] and speech recognition [6] and the research community quickly applied these methods to the field of natural language processing with some successes. In recent years, researchers have focused on unsupervised or semi-supervised learning algorithms [7] which can learn from large masses of unlabelled data. The field of *Natural Language Processing* (NLP) has a large range of application such as *Automatic Image Annotation* [8], *Dialog System* [9], *Machine Translation* [10], *Information Retrieval* [11], *Spam Filtering* [11], *Question Answering* [12] and *Natural Language Generation*. It is at the core of almost every technology we use today such as *search engine*, *virtual assistant* [13], *word processing*, *e-mail* and *social networks*. Today, it is at the centre of attention as it could have large impact on humanities, communication, medias and politics. Difficult problems such as machine translation have seen in recent years important advances and while part of the early expectations are being fulfilled, a lot of work remains to be done to face challenges of the social network era.

The problem of author anonymity of textual data is at the centre of multiple field. In humanities, many authors of important historical documents are unidentified and *plagiarism* remains a problem in literature and education. In forensics, the police often faces anonymity of letters, email or social medias posts such as threatening letters, hate mail, terrorist claims or hate speeches. In the social network era, anonymity allows people to post fake information, without the possibility to face prosecution for calumny and the imperative of deontology of classical journalism, and to create multiple identities in order to create influence by generating false impression of majority. The field of research that aims to identify or to find useful characteristics of the true authors behind textual documents based on their writing style is known as *authorship analysis*.

1.2 Authorship Attribution and Stylometry

Authorship analysis (or *stylometry*) is defined as the application of the study of linguistic style to distinguish between texts written by different authors based on measures of some textual features. Since the beginning of this field, it had legal, literary and academic applications and is strongly associated with the field of *forensic linguistics*. In 1901, Thomas C. Mendenhall published the first book [14] dedicated to *stylometry*. His goal was to use word frequencies to characterise the style of writers and applied his method to the work of Shakespeare and Francis Bacon.

This pioneer work was followed during the 20th century by statistical studies, first in 1932, by George

Kinsley Zipf and his book *The Psychobiology of Language* [15] where he introduced his law, the Zipf's law, which empirically demonstrates that the frequency of any word is inversely proportional to its rank in the frequency table.

Early work based on statistical models and machine learning are referred as *non-traditional authorship attribution studies* compared to traditional ones based on human expert evaluation. Then, in 1964, authorship attribution was first applied to the field of political science on a detailed study of the "Federalist paper", a series of 85 political essays written by James Madison, Alexander Hamilton and John Jay, which became one of the most famous and influential work on authorship attribution. Until the 90s, authorship attribution methods have mainly been applied to literary research to identify author of literary work. Applications were based on features defined manually to quantify the writing style and measures such as sentence length, word length, word frequencies and vocabulary richness were used.

In 1990s, the CUSUM/QSUM technique (Cumulative Sum) [16] method was used as evidence in court in the United States despite being strongly criticised by researchers [17]. Practitioners faced strong methodological limitations and a lack of evaluation methods. Furthermore, the testing ground based on literature made the estimation of performance difficult. Textual data were often too long and not stylistically homogeneous with small sets of candidate authors. Furthermore, the field lacked widely spread benchmarks controlled for topics.

Between 2000 and 2010, the field of authorship attribution is marked by the appearance of large datasets, thank to the rising use of the Web and of fully-automatic methods. But also to the use of machine learning methods able to handle sparse and multidimensional data. Stylometric methods were applied on "real-world" corpora such as e-mail, blogs and forums. Studies were influenced by fields such as *Information Retrieval* (IR) and *Natural Language Processing* (NLP) and evaluated with standard evaluation methods. Researchers worked on common benchmarks corpora and evaluation campaign. The influence of several different factors were analysed, such as the size of the corpora used to train models, the number of candidate authors and their distribution of their training texts. The social network era and the increasing number of cyber-crime incidents raised new problems and issues such as plagiarism, fake news detection and the communication between terrorist groups. Furthermore, the possibility for extremist groups to communicate in forums and short message services, and the enormous amount of data of the Big Data era, have led to the need of powerful authorship attribution methods to find authors of social network messages, which are characteristically short, in a very large set of candidate authors. In this context, authorship attribution has been applied to e-mail forensics [18, 19], instant messaging [20, 21] and tweets [22].

Stylometry has been applied to identify the author of literary work. For example, inter-textual distances and statistical authorship attribution methods have been applied to compare and analyse the work of Molière and Corneille [23]. Bayesian methods have been applied to the work of Shakespeare and Francis Bacon [14]. Moreover, stylometry has recently been applied to identify the true author behind Elena Ferrante [24, 25, 26]. In intelligence and criminal law, authorship analysis has been applied to attribute messages to terrorist groups and link messages by authorship [27]. The most publicised application of forensic linguistics is the arrest in 1996 by the FBI of Theodore Kaczynski, also known as the Unabomber, an American mathematician, terrorist and anarchist who killed 3 people and injured 23 others between 1978 and 1995 in a bombing campaign targeting people involved in airlines, academy and technology. After the publishing of his manifesto in the New-York Times, his style and idea was recognised by his brother and further identified by forensic linguistic specialists. Since this pioneer work, authorship analysis and forensic linguistics have been used in several criminal cases and murder trial. In 2005, two disappearances have been solved and the perpetrator convicted using DNA and forensic linguistic evidence of SMS messages. In civil law and computer forensic, these methods are used in copyright disputes and author identification of virus source code [28, 29, 30, 31].

1.3 Definition

The problem of authorship attribution consists to assign one candidate author to a text of unknown authorship based on a set of candidate authors for whom we have textual documents with true and undisputed authorship. Using machine learning terms, this problem is a single-labelled and multi-class categorisation task of textual document. In addition to this classic task, several other tasks are studied which we group together under the term *authorship analysis*. This group contains tasks such as:

1. The apparition of the Internet made possible for anyone to plagiarise the work of others. To this end, the **plagiarism detection** task seeks to find stylistic similarities between two texts or part of

these texts. There are two kinds of tasks in plagiarism analysis, *external plagiarism detection* and *intrinsic plagiarism detection*. The first refers to the use of a given reference corpus to identify pairs of very similar passages in a suspicious document. In the second, also known as *style change detection*, no reference corpus is given and we must rely on the detection of irregularities, inconsistencies or anomalies within a document ;

2. **Author verification** seeks to decide if a set of textual documents have been written by a given author. In machine learning terms, we have a two classes (true/false) and single-labelled categorisation task. A more difficult version of this task consists to find parts in documents written by the author in a document or in a stream of text ;

3. In the **author identification** task, we want to identify the author of a new document based on texts with undisputed authorship. In the closed-set context, authors belong to a set of known candidates, while in the open-set context the true author could be unknown ;

4. The field of **author profiling** seeks to extract information, such as education level, age or gender, about the author of some textual documents (single-labelled, multi-class categorisation task) ;

The overall set of attribution methods can be classify into two categories: *profile-based approaches* and *instance-based approaches*. The profile-based approach concatenates the training files of each author into a single one and extract a single profile per author. When a new unseen text needs to be categorised, the system take the most likely author based on some similarity measures. Differences between the training documents of each authors are disregarded and the training process is quite simple as it imply only to extract a profile. In the other hand, *instance-based* approach aims to develop probabilistic models that attempt to maximise the probability $P(x|a)$ for a text x to belong to a candidate author a. When an unseen file need to be classified, the model seeks the author with the highest probabilities. This thesis presents a study of RNNs applied to authorship attribution of single and multi-authored documents based on a varieties of different features.

1.4 Stylometric features

The field of *stylometry* is based on style marker-based models and researchers seek to compare the computational requirements needed for measuring them. Such features are extracted from textual documents that can be of several types. The simplest features are extracted directly as a sequences of *letters*. Specific software packages referred as tokenizer can extract *lexical* features based on words and vocabularies. Documents are composed of a series of elements named *tokens*. A deeper linguistic analysis can be used to extract deeper linguistics markers such as *semantic* and *syntactic* features. When documents are specific to an application such as e-mail or Web, *application-specific features* can be used.

Lexical features are based on vocabularies such as sequence of *tokens*, numbers or punctuation marks. Most lexical features need a tokenizer to be extracted which can introduce noise. Furthermore, tokenization is a non-trivial task for language such as Chinese. The first lexical features used were *sentence-length* and *word-length* [14]. *Vocabulary richness* are another example of lexical features commonly used which quantify the diversity of the vocabulary. *Type-token ratio*, a typical measure of vocabulary richness, is defined as,

$$TTR = \frac{V}{N}$$

where V is the number of different word types in the document and N the total number of tokens. However, vocabulary richness is heavily dependent on text length and several solutions to this issues have been proposed such as K [32] and R [33] with mixed results. Another feature is the *hapax legomena* which designates a word-type used only once.

One of the most commonly used feature is the *Bag-Of-Words* (BOW) where word frequencies are put into a single vectors without regard for word orders. The vast majority of studies are based on that. Among the best features to discriminate between authors are the *Most Frequent Words* (MFW). Early studies defined the set of the most 100 frequent word-types as the most adequate to represent the writing style of an author. This can lower the number of dimensions used as inputs and powerful machine learning methods such as *Support Vector Machines* (SVM) can handle high-dimensional input features and can then be based on the

250 to 1,000 most frequent words. It is to be noted that as the dimensionality increases with more frequent words, more content words are added.

One of the possible issues that can arise using features with a lot of lexical meaning is that models can base their prediction on content if the corpus is not topic controlled. To make the prediction *topic-independent*, it is possible to use *function words* [34], which are words with little lexical meaning and represents grammatical relationships among words. They are like the glue that holds sentence together and are then important elements in the structure of language. Words which are not function words are named *content words* which include adjectives, verbs, most adverbs and nouns. Their meanings are defined in dictionaries and function words usage are defined by grammars. Other lexical features may require other tools such as *stemmers* and lemmatizers.

Features based on word frequencies and bag-of-words discards all information about word order. To take contextual information into account, *word n-grams* have been used in many studies. However, word n-grams increase dimensionality considerably with n as it takes account of all possible combinations of tokens. As most combinations have a probability of zero, these methods result in an highly dimensional sparse representations. The second issues is the possibility that word n-gram will capture topic-related information not related to style markers.

Another solution to avoid completely the possibility of models making prediction using mainly content related information is to use *character features*. Based only on letters, a large set of features can be defined such as alphabetic characters, digits, upper cases or lower cases, letter frequencies or punctuation marks. They have been proven quite useful to measure style markers and are easily extractable with any language. Another solution to take into account word order is to base features on frequencies of *character n-grams* which can capture more nuances in writing style and more lexical information [35, 36, 37]. They are also more tolerant to noise such as grammatical error which are often present. Such features have multiple advantages, they are a solution to oriental languages based on ideogram or graphical symbols as they are language independent but have also pretty low computational requirements. Very good results are obtained with character n-grams which are considered as one of the most effective measures.

We can represent more complex linguistic knowledge by employing *syntactic* features as authors tend to use similar syntactic patterns. This kind of features could be more reliable fingerprints than lexical features. However, theses features are based on robust and reliable NLP tools which makes them language-dependent and vulnerable to noisy datasets. A common method to extract syntactic features is to use *Part-of-Speech* (POS) tagger which will assign morpho-syntactic information to each token based on the context. Commonly, tags assigned to tokens are noun, verb, article, adjective, preposition, pronoun, adverb, conjunction and interjection. Some researchers used POS n-gram frequencies to represent author's styles. Other syntactic features are similar to the ones used by human experts and are based on syntactic errors such as *sentence fragment*, *run-on sentence* and *mismatched tenses*.

One recent trend in text representation is to use continuous vectors to represent tokens. In NLP, continuous representation is known as *embedding* and a set of methods coming from language modelling and unsupervised feature learning is used to map tokens in the vocabulary to vectors of real numbers. It transforms a very sparse highly-dimensional input space with one dimension per word to a continuous embedding space with a much smaller number of dimensions. Different methods exist to generate such mappings such as neural networks (Continuous Bag-Of-Words, Word2Vec and Skip-Gram)[38], probabilistic models [39] or dimensionality reduction on word co-occurence matrix [40, 41, 42]. With neural network models, shallow or deep classifiers are trained to predict a word from its direct context. It has been shown that when embedding are used as input representation, it boosts the performance of many NLP tasks. It is also much simpler for neural networks to generalise when continuous vector representation are used. The underlying idea behind word embedding is the Firth's concept that a word is characterised by the company it keeps. It comes from the field of *distributional semantics* which try to categorise and quantify the semantic relations between linguistic elements. As a result, words with similar semantic contents are grouped near each others in the embedding space [43]. The emphasis is then more on the semantics and less on the stylistic aspects.

The early studies developing these techniques started with the development of vector space model in the 1960s. The idea was to reduce the dimensionality using *Singular Value Decomposition* (SVD) and in 1980s *Latent Semantic Analysis* (LSA) was introduced. In 2000s, researchers started to use neural probabilistic models [44] to learn distributed representations of words in order to reduce the dimensionality of classical representations. The field really took off after 2010 with the possibility to train very deep models on huge amount of data. Today, three models are mostly used. The first known as *Word2vec* which was developed by a team at Google. It can use two different architectures. The *Continuous Bag-Of-Words* (CBOW) [43]

architecture where the model has to predict the current word from a window of surrounding context words without taking into account their order. And the *continuous skip-gram* [38] where the current word is used to predict the surrounding words. CBOW is faster than the skip-gram model but the latter has better performances with infrequent words. The second is known as *GloVe* [45] and use dimensionality reduction methods such as singular value decomposition on word co-occurence matrices. A third is *fastText* and has been developed by the Facebook's AI Research Lab (FAIR) which use CBOW and Skip-gram and has been trained on more than 294 languages.

These kinds of features which take the meaning of tokens into account are known as *semantic features* and often need pretraining on a task related to the original problem at hand. Along continuous token representations such as word vectors, *semantic features* include *semantic dependency graphs*, *synonyms* or *hypernyms* such as *WordNet* [46], or *Systemic Functional Grammar* which consists to associate certain word with semantic information. *Semantic features* in addition with lexical and syntactic information have been shown to improve accuracy. Finally, *application-specific features* use structural-measures such as font colour or size present in HTML documents and content-specific keywords.

To avoid overfitting, a good solution is to reduce the dimensionality of the input. This process is referred as *feature-selection*. One kind of method is the feature subset selection which select the most effective feature from a set of initial features. It is possible to select the most frequent features which capture more stylistic variation such as character n-gram selected with χ^2, *information gain* or *odds ratio*. These kind of frequency-based feature selection have been shown as very effective. In [47], authors proposed a new important criterion for selecting features for authorship attribution referred as *instability*. Stable features are those which remain practically unchanged in all texts given of number of variations of the same text all with the same meaning.

Another solution is *feature extraction* where we start from an initial set of measured data and build a set of derived features intended to be more informative and non-redundant to make the learning easier. A lot of methods can be used to extract new features, mainly from the field of *dimensionality reduction*, such as *independent component analysis* [48] (ICA), *latent semantic analysis* (LSA) [49], *principal component analysis* (PCA) [50] or *auto-encoders* [51]. The most used method for feature extraction in *authorship attribution* is the *principal component analysis* (PCA) which combine linearly the input features. With PCA, one can represent input texts in a two-dimensional space by selecting the two most important principal components.

In neural network-related models, methods such as *auto-encoders* have been extensively used for *dimensionality reduction* and *unsupervised feature extraction*. The goal of an *auto-encoder* is to learn a smaller representation (a code) of the input data which can effectively be use to reconstruct initial inputs in order to reduce their dimensionality. Some of the most powerful machine learning techniques involved sparse auto-encoders stacked inside deep neural networks. However, *auto-encoders* have not been evaluated for reducing textual data in the field of *authorship attribution*.

1.5 Authorship analysis in the Age of Post-Intelligent Design

In recent years, a breakthrough technology in the field a machine learning known as *deep learning* allowed huge advances in the fields of computer vision and speech recognition. Theses methods based on the connectionist model, often referred as neural networks, achieved very low error rate on some classical benchmarks in computer vision and speech recognition. However, deep learning methods are parts of an ongoing trend which sees a radical change in research and development (R&D) of algorithms and software. Indeed, at the beginning of the computing industry in the 50s and until the large adoption of machine learning technologies, algorithms were designed by highly trained computer engineers who thought every part of their systems. With the broad adoption of machine learning methods since the 90s, the computing community focused their research on methods able to learn a competence out of a large amount of training examples. From the 90s to the early 2010s, the learning algorithms were mainly used to learn specific tasks using high-level representations designed by engineers specialised in the concerned field.

However, since 2012 and the breakthrough achievement of deep learning, learning algorithms are used not only to learn tasks based on high-level representations, but to learn the complete bottom-up transformation from low-level representations (such as pixel) to high-level ones (objects, structures, etc). Human-designed, top-down methods, are more and more replaced by mindless bottom-up processes with a high level of competence but with absolutely no comprehension. Thanks to the availability of large amount of data and processing power, our capacity to predict has reached unparalleled heights while our ability to understand predicted phenomena has been greatly reduced by bottom-up methods. These opaque models are also referred

Feature type	**Feature**	**Properties**
Lexical features	Word length	Language dependent, noise
	Vocabulary richness (TTR, K, R, HL)	Language dependent, stability issues, unreliable results
	Bag-Of-Words (word frequencies)	Language dependent, vast majority of studies based on that, disregard word order
	Word n-gram	Dimensionality increase with n, very sparse vectors, can capture content-specific information, need machine learning models able to handle high dimensionality, not always better than individual words
	Most frequent words (MFW)	Lower dimensionality
	Function words	Lower dimensionality, topic-independent
Character features	Bag-of-character (character frequencies), alphabetic, digit, uppercase, lowercase	Language independent
	Character n-gram (frequencies of n-gram)	Can capture nuance of style and lexical information, tolerant to noise, solution for oriental language, low computational requirements, most effective measure, how to choose n?
Syntactic features	Sentence length	Language independent
	Part-Of-Speech	Need a POS tagger and robbust NLP tools, language-dependent
	Sequence of POS	
	Rewrite Rule Frequencies	
	Sentence fragment	Similar to human experts
	Run-on sentence	Similar to human exports
	Mismatched tense	Similar to human experts
Semantic features	Word embedding (GloVe, Word2Vec, ...)	Need pre-trained embeddings
	Character embedding	Need pre-trained embeddings
	Character n-gram embedding	Need pre-trained embeddings
	WordNet	Synonyms + hypernyms
	Systemic Functional Grammar	
	Semantic Dependency Graphs	
Application-specific features	Structural-measures (HTML, font color, font-size)	
	Content-specific keywords	

Table 1.1 – Overview of the different features used in the field of authorship analysis with their descriptions and properties.

Figure 1.2 – An Australian termite castle, an example of bottom-up process.

Figure 1.3 – The Sagrada Familia, an example of top-down process.

in the literature as *black-box models*. In computing and engineering, black boxes are systems or devices where only inputs and outputs are visible, without any knowledge of its internal working. Deep learning models are not real black box models as we have complete access to their internal working, but are closed to grey-box models. Indeed, their implementation is "opaque" as the designed process is not done by any human but by numerical approximation algorithms and the internal data processing is distributed over a gigantic quantity of local mindless units. As a consequence, we have little information about how and why these models came to a given conclusion.

The French physicist Hubert Krivin refers to this trends in [52] as the capacity to predict without understanding. According to Krivin, understanding is the ability to identify causal relationships that underlies correlations, to link phenomena, and to connect a priori unrelated knowledge. But deep learning models are composed of many mindless units having high competence as an emerging property while being unable to give a justification for their prediction such as a set of rules or if-then clauses. They have the ability to predict without related theories, and without specifying why. He then asks a central question : *in which circumstances do big data and deep learning help to better understand our environment* ? The capacity to predict without comprehension could lead to act without comprehension as big data and deep learning cannot replace scientific theories. According to him, scientific theories are not a simple and systematic accumulation of empirical data, but are also able to explain and anticipate unknown events. However, the current trend in machine learning is not directed towards good theories but towards good predictions as a basis for profit and marketing. The trend is well explained in the famous article of Chris Anderson, published in Wired in 2008, *The End of Theory: The data deluge makes the scientific method obsolete* [53]. He argues that since the appearance of search engines and digital giants such as Google, a completely different approach, based on statistics and applied mathematics, has become dominant. If statistics about data is predicting a useful property, that's enough, no semantic or causal analysis is required.

The American philosopher Daniel C. Dennett refers to this trend under a broader sense as *Competence without Comprehension* [54]. According to him, this trend is similar to the Darwinian evolution as it embodies a completely different approach of the R&D process. According to Dennett, Darwinian evolution is a design process exploiting information in the environment to create, maintain and improve the design of things. Design can then be separated in two main varities, the first being evolution by natural selection, which takes a lot of time (or computing power) and needs an extensive amount of data. The second is the human intelligent design (HID) which is faster, purposeful and less dependent on data. However, evolution by natural selection is able to produces designs that are more cunning, more devious, and much more efficient. These two kinds of process can also be viewed as *bottom-up* and *top-down* processes according to Dennett, who gives two examples presented in Figures 1.2 and 1.3. The Australian termite castle is the perfect example of a bottom-up process where millions of little local mindless but very competent units (the termites) are working together without any central and purposeful controlling entity. The castle and all its advances architectural features is an emerging property of this decentralised process. The human brain and its 100 billions neurons is another example of bottom-up process. On the other side, the Sagrada Familia is the perfect example of

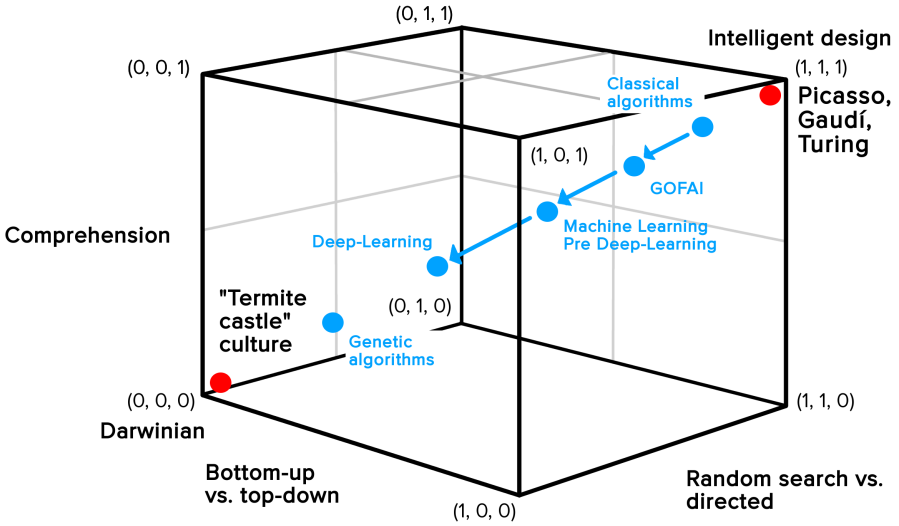


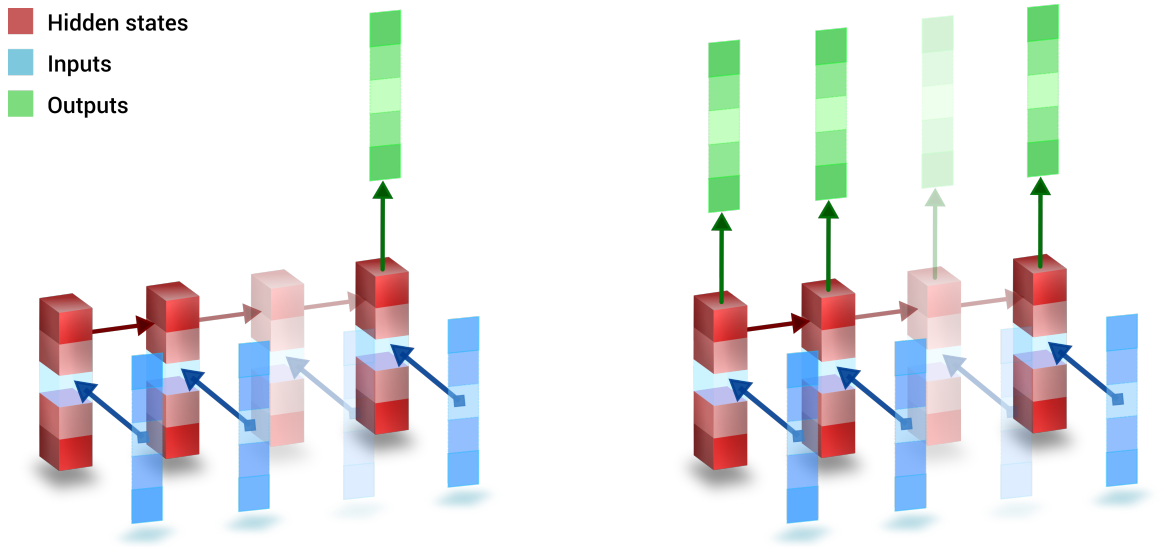
Figure 1.4 – The 3D darwinian space defined by Daniel Dennett.

top-down design performed by a central all-controlling and very competent architect, using all the human intelligent design tools such as blue prints and proofs of concept. Surprisingly, the two constructions are very similar but also the result of a completely different design process.

This separation into two types of processes by Dennett is the result of two reversals of reasoning that had a significant impact on science, computing and artificial intelligence. The first was made by Charles Darwin when he introduced the evolution by mean of natural selection. As stated by Dennett : *in order to make a perfect and beautiful machine, it is not requisite to know how to make it.* The second was brought by Alan Turing in its work on the universal Turing Machine and is summarised by Dennett as : *in order to be a perfect and beautiful computing machine, it is not requisite to know what arithmetic is.*

The Darwin's inversion of reasoning shows that we don't need to know how to make a very efficient algorithm to make it, while Turing's one shows that these algorithms do not need comprehension to be very efficient at a task. For example, in NLP, these inversions mean that we don't need to know how to make a very efficient document classification algorithm to build one, and that this algorithm does not need to know or comprehend the human language. The only things necessary are a sufficient amount of training examples, and an algorithm able to extract relations between input data and the output to be learned.

To show formally the differences between the two design processes, Dennett introduces the Darwinian space as a cube where the 3 dimensions represent respectively the bottom-up versus top-down process, the comprehension versus no comprehension, and the random search versus directed search approach. Figure 1.4 shows the two examples with the evolution of methods in artificial intelligence. The termite castle example is at the lower left, being completely Darwinian, with a bottom-up process, no comprehension and based on random search. On the top right corner, the human intelligent design is completely directed with full comprehension and based on top-down processes. Classical algorithms are based on this approach along with the first AI systems which were based on deduction systems created by human engineer. In the 90s, pre-deep learning machine learning algorithms became widely available and based on a learning approach with less comprehension and using less directed search. The deep-learning and neural networks methods used since the early 2010s removed the human intelligent design in the feature selection loop to attain an even less directed search, less comprehension using a large amount of computing units in a bottom-up process where competence is an emergent property of the training process. This trend from the top-right corner of the Darwinian space to the bottom-left part was made possible by the large amount of data made available by the Web and social networks, and the increasing computing power made available by GPUs and specialised chips. Today, neural network models use optimisation algorithms such as the *Stochastic Gradient Descent* (SGD) based on semi-random search through a space of possible functions with no understanding of the task to handle being present in the design process. Furthermore, these optimised functions are made of millions of simple and mindless units (artificial neurons).



Many-to-one Many-to-many

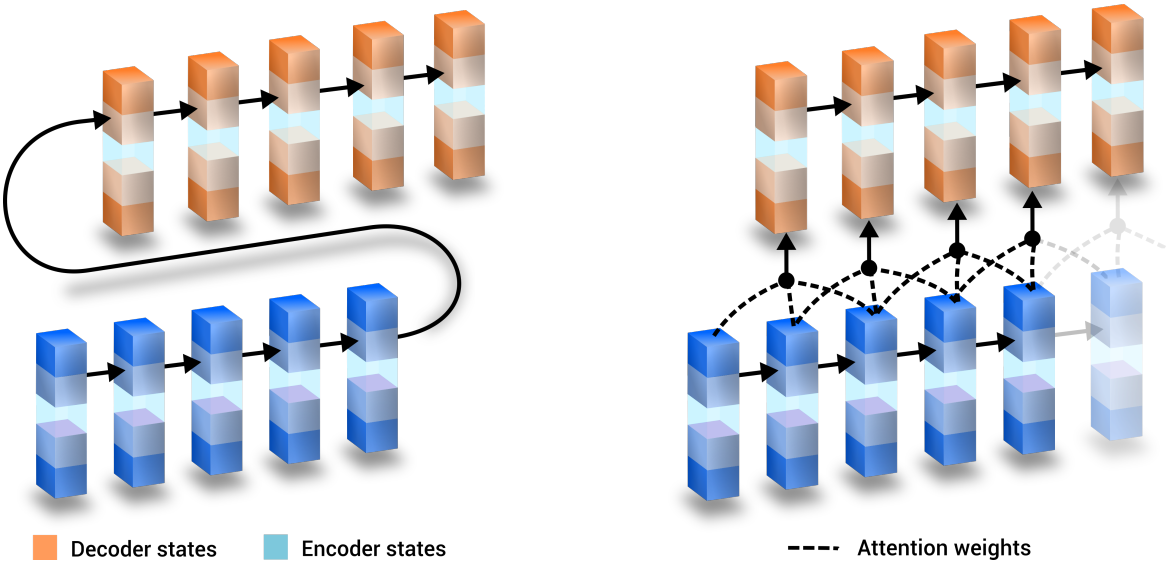
Figure 1.5 – Many-to-one and many-to-many architectures.

The work of Daniel Dennett asks fundamental questions. *How to extract comprehension out of ultra-efficient termite-type models such as neural networks ? How comprehending top-down processes can emerge from bottom-up systems made of units without any comprehension ?* This question is the focus of a field of research currently very active in the *deep learning* community and usually referred as *explainable AI* [55]. In NLP and document classification, we see four different approaches to extract evidences from neural models : many-to-many architectures, attention mechanisms, deconvolution layers and sensitivity analysis. We will not give in this section a complete technical introduction to these methods (see Section 3.2.8) but rather how they can be used to extract comprehension and evidence out of termite castle-like models such as recurrent neural networks. Even if this question is fundamental for science, it might seem trivial in certain domains such as computer vision or speech recognition where we do not require any comprehension from the system. However, in some case, the comprehension is fundamental and could have a huge impact on society and science. The field of authorship analysis is one of these fields which require serious and robust methods to extract comprehension from highly efficient models. This verdict led us to focus on more transparent models such as many-to-many, or those able to use attention mechanisms. The smallness of datasets available in the field of authorship analysis also led us to focus on the performances of the different models evaluated on datasets of different sizes.

1.5.1 Many-to-many vs. many-to-one

Classical approaches to authorship attribution use the *Bag-Of-Words* (BOW) paradigm which see text as one block with a beginning and an end. Each text belonging to one or more classes that have to be predicted. An alternative approach is to see text as a stream with a beginning but which can continue indefinitely. With the social network era and the availability of large amount of these kind of data, coming mainly from Twitter, a lot of studies addressed the problem of attributing an author to short texts. However, the kind of data are also heavily time related as users post messages, comments and reaction on a regular basis through very long time intervals. For example, a Twitter time line is a stream of tweets sharing some properties, while blog posts are time-related textual documents sharing the same topic or the same author. Furthermore, each elements share properties with previous ones while being continuously evolving. For examples, textual data can be affected by temporal changes such as the evolution of language and writing styles with age.

Furthermore, most textual data are not coming out of a monolithic block but can belong to multiple classes through time and the source that generates them can change properties. For example, a Twitter account can be fed by multiple authors or be hijacked by someone else. In July 2020, multiple Twitter accounts of prominent figures, including Joe Biden, Elon Musk, Barack Obama and Bill Gates have been compromised as part of a large scam. Piece of literary work are often a collaboration of more than one author. Blogs are



Encoder-decoder | Encoder-decoder and attention mechanisms

Figure 1.6 – Encoder-decoder architecture with and without attention mechanism.

not always managed by only one author, and more generally, parts of textual data can be a copy of someone else's work (a case referred as *plagiarism*). In the field of *authorship analysis*, identifying portion(s) of text in a document written by more than on author is known as *author diarization* or *style change detection* (SCD). Given a document and an external collection of text written by candidate authors, the system should output which portion of the text has been written by each author. The text document is written by n authors, no single main author is given, and each author can contribute in arbitrary content. *Extrinsic plagiarism detection* is opposed to *intrinsic plagiarism detection* [36, 56, 57, 58, 59] where the system should determine whether a suspicious document is written by a single author without any potential sources.

Two main approaches to plagiarism detection exists. The first is to split a text document into a set of text segments (e.g. sentences) and feed a set of features to an author style function that measure an authorship similarity and find a critical threshold to detect plagiarised segments. For example, in [36], authors use a set of overlapping segments, known as a *sliding window*, with frequencies of character trigrams. Other studies used inverted frequencies [58], punctuation and part-of-speech tags count [56]. The PAN challenges in 2016 [60] introduced a general framework for *intrinsic plagiarism detection* referred as *author diarization* (AD). The second possible approach used in neural networks and in this study is referred as *many-to-many architectures*. Here we consider documents as timeseries where each input (token) is linked to an output modelling the class probability at time t. This architecture is opposed to the traditional many-to-one used in traditional authorship analysis methods and with some neural models used for classification. Figure 1.5 shows the two architectures. On the left, the many-to-one architecture transforms all inputs (red) to an hidden representation (green) but only the last hidden representation is used to compute the outputs (blue) and then class probability. On the right, the many-to-many architecture transforms also each input to an hidden representation, but the output class probability is learned and computed at each time steps. The final class probability for a document can then be computed using aggregation methods such as the average probability over time.

The many-to-many architecture has two main advantages over the many-to-one. The first is the possibility to handle stream of text and multi-authored documents. Furthermore these models are able to give a class probability at every time step and at different levels (token, sentence, paragraph) without computing the model again. The second is their ability to give evidences on how the document is classified. The output stream of class probability can be used to determine which tokens, sentences or parts of the text has a high probability for one specific author and then a list of lexical or syntactical representation which are typical of this author. Furthermore, in traditional black-box methods [61], the observer records all the observation over time in order to investigate its internal processes and to test hypotheses about causes and effects. All knowledge obtainable from the black-box depends on the variety of recorded behaviours. The many-to-many architecture have an evident high output variety which can be put to work to investigate how the neural

Figure 1.7 – Example of an input image to an attention network

Figure 1.8 – The attention map created from the attention mechanism

model processes textual information. In other words, the many-to-many box is much more transparent than its many-to-one counterpart. This study includes multiple tasks with multi-authored documents, some coming from social medias. To match the need of the field of authorship analysis for developing a robust comprehension of how models perform their conclusions, we therefore decided to evaluate only neural models based on the many-to-many architectures.

1.5.2 Attention mechanisms

Attention is a very important idea in the deep learning community and was initially introduced in the context of neural machine translation using sequence-to-sequence models [62, 63, 64]. This subsection gives a non-formal definition of attention mechanisms, for a mathematical and formal definition see 3.2.9. It is now used in various problem such as object recognition [65], document classification [66, 67, 68] and image captioning [69, 70]. A sequence-to-sequence model uses a RNN (the encoder) to encode a sequence into a single hidden representation (many-to-one), and another RNN (the decoder) to decode this hidden representation. In machine translation, an RNN is used to encode a sentence in the source language, and another RNN is used to produce a new sentence in the target language. The attention mechanism was introduced as a response to a common issue with these models which have problems remembering longer sequences. Normal sequence-to-sequence models forget the intermediate states of the encoder and use only the last hidden representation (states) as the first state of the decoder (the context vector). However, it is very difficult to embed all the necessary information into a single vector when the sequences is very long. The main idea behind attention mechanism was to use all the encoder state to construct the context vector. This context vector can be constructed by taking into account the last past n_q encoder states and by assigning to each of them an attention weights. These attention weights can be learned by the neural network which is then able to learn to identify the relevant states for each output. To determine these attention weights, a softmax layer is used so that all weights are between 0 and 1, and that all weights sum to 1. We are then able to have a formal probabilistic description of the attention weights.

By multiplying each state by their attention weight, a state with an attention weight near 1, will have more of its information embed in the context vector. Figure 1.6 shows the encoder-decoder architecture with and without attention mechanism. The decoder state is computed using a weighed sum of the encoder state based on the learned attention weights. However, the attention mechanism allows more than a better efficiency in neural machine translation, it also let use this attention mechanism to understand what the neural network is focusing on when doing a prediction. Although units that control attention are not trained for the purpose of creating human readable explanations, they do directly reveal a map of which information passes through the network, which can serve as a form of explanation. Attention mechanism can also be used for image classification as shown in figures 1.7 and 1.8. Here the attention weights are used to produced an attention map which shows where the neural model is focusing its attention to get its classification. The same principle can be used in neural machine translation to determine which word in the source language corresponds to which word in the target language. In document classification, it can be used to determine on which word neural networks are focusing on when doing their predictions.

Deconvolutional layers (deconvnet)

The deconvolutional layers were introduced in [71]. The original idea was to use deconvolutional layers as a visualisation technique to understand why and how CNNs perform so well in a variety classification task. Using this technique, named multi-layered Deconvolutional Network (deconvnet), as a diagnostic tool, the authors were able to find new models outperforming the current state-of-the-art models, and to discover the performance contribution from different layers. The main idea is to project the feature activation of a specific layer back to the input space, showing what input pattern originally caused a given activation in the feature maps of the CNN. The deconvnet can be seen as a model using the same transformation as a CNN (filtering, pooling) but in reverse, mapping features to pixels (in case of images) instead of pixels to feature maps. A deconvnet is then attached to each layer of a convnet providing a continuous path back to the input space. At first, an input is presented to the CNN and the hidden representations are computed at each layers. To examine a specific CNN's feature map, they put all the other maps to zero, and set the map to examine to the attached deconvnet layer to reconstruct the inputs. The result obtained from a single map gives thus only a small piece of the original inputs but with structures weighted according to their contribution to the map's activation. The deconvnet layer can then show which parts of the input are discriminative.

In NLP, deconvnets were mainly used in [72] to propose the strategy called Text Deconvolution Saliency (TDS). TDS can be used to visualise linguistic information used by a CNN for a text classification task. The authors introduced TDS as a response to the issue of the utilisation of CNN in computational linguistics as these models can learn implicit linguistic features but are not able to explain or show the linguistic structures underlying their decisions. TDS extends deconvnets to text by inverting feature maps to the original embedding space by up-sampling the feature maps to obtain a sample space of size (words, embedding dimension, filters). Three datasets are then used to show that these models are able to outperform classical methods such as z-test on classification tasks while being able to explain which words and structures underlies these performances.

1.5.3 Sensitivity analysis

The idea behind sensitivity analysis is to identify important input features based on the model's locally evaluated gradient or on some other local measure of variation. The relevance score $R_i(x)$ of the i-th feature is defined as the squared partial derivative of the output with respect to this feature.

$$R_i(x) = (\frac{\partial f}{\partial x_i})^2 \tag{1.1}$$

The gradient is measured for the data point x using back-propagation, which makes this technique easy to implement. The important features are those to which the output is the most sensitive. Sensitivity analysis has been used in medical diagnosis and image classification [73].

1.6 Neural networks in Authorship Analysis

Neural networks and deep learning have been extensively applied to *Natural Language Processing* (NLP) problems but the first attempt to process natural languages with computers were based on a set of rules as human languages are symbolic and discrete in nature. But neural networks and statistical approaches can also be very effective on these tasks as natural language is also highly variable and ambiguous. As a result, first applications of machine learning to NLP were based on *statistical learning* and were centred on models such as *Support Vector Machines* (SVM) and *Logistic Regression* on a supervised learning basis. Regarding features, these models were applied on highly dimensional and very sparse input vectors. But *machine learning* models have to face a challenge with natural languages as their properties lead to *data sparseness*. The size of the set of all possible sentences is infinite and each discrete symbols, or words, can be combined in almost infinite ways to form new meanings. This is a huge challenge for artificial intelligence practitioners as we have to build systems able to learn from examples. However, these systems also have to process examples we are very unlikely to observe and very different from those found in the training set as the number of possible sentences is extremely large.

Deep learning is a re-branded version of neural network with deeper architecture inspired by how the brain works. Deep learning consists not only to make prediction based on past observations but also to find new and effective representations of the data for the task at hand. A major component of neural network

models applied to natural language is the *embedding layer* which maps discrete symbols to continuous vectors in a much smaller dimensional space. As a result, words are not isolated symbols but mathematical objects with relationships between them. This principle makes the generalisation easier from one word to another. This embedding layer is part of the learning process and helps to overcome the challenges of discreteness and data-sparsity.

There is basically two different neural network architectures, the feed-forward neural networks (FFNNs) or multi-layer perceptron (MLPs) and recurrent neural networks (RNNs). MLPs work with fixed sized inputs or with variable-length inputs where elements order are disregarded. On the other hand, RNNs are specialised in the process of sequential data where the inputs are sequences of elements. As they are very efficient on sequential data, they are a target of choice for language processing.

Furthermore, RNNs do not follow the *Markov assumption* which states that future states of the process depends only upon the present state, and not on past states. This allows RNNs to condition entire sentences taking partial word order into account. This leads to impressive results in *Language Modeling* (LM) where models have to predict the next word in a sequence. This task is at the cornerstone of many applications in NLP as a well-designed language model makes a critical difference in many applications such as speech recognition [74, 75], semantic extraction [76, 77] and machine translation [78]. As a result, NLP researchers focused on language modeling (LM) and a lot of groundbreaking studies have been published during past decades. Before the 2000s, language modeling used non-parametric approaches based on n-grams [79]. Neural networks applied to LM began to attract attention in 2003 ([44]) but only showed significant advantages over other LM approaches until RNNs were investigated in 2010s [80, 81].

Neural networks have also been extensively applied to *machine translation*. Traditional approaches on translation are phrase-based system consisting of many elements tuned separately [82]. In the other hand, in neural machine translation we try to train single and large neural models which read sentences and outputs a correct translation. Most neural machine translation systems are based on the *encoder-decoder* architecture [83, 78] with an encoder and a decoder for each language. The encoder encodes the input sentence into a single fixed-length vector while the decoder outputs a translation from the encoded vector.

Other studies have demonstrated that FFNNs can be competitive on sentiment classification, question answering [12], syntactic parsing and sequence tagging [84]. On the other hand, Convolutional Neural Networks (CNNs) allow to find topic indicators [85] independently of their position and show promising results on document classification [85], sentiment classification [86], short text categorisation, named entities recognition (NER)[87], event detection [88], paraphrase identification [89], question answering [12] and part-of-speech tagging [90]. Finally, RNNs can work with sequences and trees and preserve a lot of structural information. Recursive networks can handle tree-like structure while RNNs can model sequences. RNNs obtained impressive results on LM tasks [91, 92, 93], sequence tagging [94], machine translation [62, 95, 83], parsing [96, 97], text normalisation [98], response generation [99] and POS tagging [100].

The first studies on neural networks applied to authorship attribution started in mid-1990s with the application of FFNNs to literary work. For example, authors in [101, 102, 103] used frequencies of a small set of words and word n-grams on work of Shakespeare, Marlow and Fletcher. In [104], the same model was used with function words and applied to the Federalist Papers. In 1999, authors in [105] used character-based features with a neural network model to identify Dutch poets. The field of authorship attribution had to wait until 2007 to see new studies using neural models to identify authors. In [106], authors used the first FFNN with two hidden layers with function words and punctuation marks applied to novels. It is the first work applying deep neural networks to authorship attribution. In [107], a not-specified neural model was applied again to the work of Shakespeare, Marlowe and to the Federalist Papers. Learning Vector Quantization (LVQ) [108] was first used on authorship attribution with function words on work of Tamil scholars and in 2010, FFNN was again used with many different features on documents of the Greek Parliament [109]. Studies from early 1990s to 2010 were monopolised by FFNNs with Bag-of-Words (BOW) and syntactic features applied on corpus with small sets of candidate authors (from 2 to 3).

The first work on *deep learning* applied to authorship attribution started in early 2010s with the application in [110] of self-organising maps (SOMs). SOMs are neural network models trained on an unsupervised way to produce a low-dimensional and discrete representation of the inputs named a map. In [111], authors used Generalized Regression Neural Network (GRNN) with function words and morphological features on work of Tamil scholars [112]. The first modern *deep learning* model (CNN) was used with GloVe words embedding on a subset of the Gutenberg Project [113]. The same year saw the first application of RNNs and recursive neural networks to authorship attribution. First in the PAN 2015 competition [114] and then on data coming from Wikipedia with recursive NN and LSTM [115]. Following studies applied deep feed-

Model	**Year**	**Feature**	**Dataset**	$\|A\|$
FFNN [101, 102]	1993, 1994	FW	Shakespeare, Marlow, Fletcher	2
FFNN [103]	1994	Word n-gram (n=1,..,5)	Unknown corpus	2
FFNN [104]	1996	FW	The Federalist Paper	3
Unknown [105]	1999	Characters	Dutch poets	3
2 hidden layers FFNN [106]	2007	FW, punc. marks	Writers novels	2
Unknown [107]	2008		Shakespeare, Marlowe, Federalist Papers	2, 3
Learning Vector Quantization (LVQ) [108]	2009	FW	Tamil scholars	3
FFNN [109]	2010	Verb-related, POS, structural, negation words, lemma	Greek Parliament	
Self-organizing maps [110]	2011		Unknown benchmarks	
Restricted Boltzmann Machine (RBM) [111]	2013			
Generalized Regression Neural Network [112]	2013	FW, morphological	Tamil scholars	
CNN [113]	2015	GloVe	Gutenberg	
Multi-headed RNN [114]	2015	Characters	PAN 2015	
Recursive NN, LSTM [115]	2015	Grammatical, vocabulary	Wikipedia	
Self-Organizing Maps [116]	2015	BoW-related	Professional writers	14
FFNN-LM [117]	2016	Word embedding	Coursera transcript	
CNN [118]	2016	Word, character embeddings	Eron, IMDb62, Blog, Twitter influencer, reddit gaming	
FFNN [119]	2017	Lexical, layout, structure and syntax	Java source code samples	40
Deep Belief Networks [120]	2017	Lexical, syntactic and application-specific	Enron, Twitter, Impostors tweets	150, 100, 10
CNN [121]	2017	Character n-gram embedding	Twitter	50-1000
RNN, LSTM, GRU [122]	2017	GloVe	Reuters RCV1	50
LSTM [123]	2017			10
LSTM, GRU, Siamese [124]	2017	GloVe size 50	Reuters C50, Gutenberg	50

Table 1.2 – Overview of neural models and features used in authorship analysis in the scientific literature with the year of publication and the number of authors in the dataset.

forward architectures [117] with convolutional and embedding layers [118] on many different benchmarks such as Coursera transcript, Eron emails, Blog, Twitter and for the first time computer software source code [119]. In 2017, researchers focused on RNNs such as LSTMs, GRUs [122, 123] or Siamese networks [124] and such models achieved state-of-the-art performances on classical benchmark such as the Gutenberg Project. Years 2010 showed the usage of large datasets coming from social networks with much bigger sets of candidate authors (from 10 to 1,000). Table 1.2 shows the evaluation of neural network models applied to authorship analysis. The application of RNNs to authorship attribution stay sparse as no peer-reviewed papers were published on the subject despite the impressive results obtained with these models in other NLP tasks such as language modelling and machine translation. Consequently, a lot of space remains to discover. First to compare models on several criteria such as performance and complexity. Secondly to evaluate which representations are the most appropriate to these tasks. Indeed, some RNNs have never been applied to this task and some representations and dimensionality reduction methods such as auto-encoder have also never been applied to this field, to the best of the author's knowledge. The present study is designed to feel this gap.

Chapter 2

Evaluation methodology and dataset

2.1 Introduction

In this chapter, we introduce the objectives of this study, our methodology, the corresponding formalism and the datasets used to evaluate models. We will explain how these datasets were created and built, how we evaluated our models, which tasks we choose, and how to reproduce our results. This section introduce briefly our objectives and how to achieve them. Sections 2.2, 2.3 and 2.4 will introduce respectively each tasks. Section 2.5 will compare the three tasks. Finally, Section 2.6 will show how to reproduce our results and the different tools used to implement models.

2.1.1 Objectives

As shown in Chapter 1, an empirical study evaluating performances of RNNs in authorship analysis is still missing despite their impressive capacities on many NLP tasks. This study is intended to fill this gap. To fulfill this task, we want to compare these models to state-of-the-art methods in authorship analysis. But we also want to determine which kind of RNN performs best and in which context. We therefore decided to evaluate three types of recurrent models. The first based on the Reservoir Computing (RC) paradigm referred as Echo State Networks (ESN). The second and the third based on the deep learning paradigm and referred as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU). For each models, we evaluated their performance with different hyperparameter values to define the main factors that determine these results. In authorship analysis, the kind of feature used is decisive for many methods, we therefore decided to test RNN models with different features based on word embedding (WV), Part-Of-Speech (POS), function words (FW), character unigrams (C1), character bigrams (C2) and character trigrams (C3). These three models and their evaluated hyperparameters are,

1. **Echo State Networks (ESN)**, based on the RC paradigm, which use only linear regression as training method and a recurrent neural layer expanding the input space into a higher dimensional space. This idea is similar to Support Vector Machines (SVM) but with a temporal dimension. ESNs are sometimes referred as *Temporal SVM*. We evaluated this model with different reservoir sizes, leak rates and input types. As these models use a simple linear regression algorithm for training, input features can only be pretrained, we then restricted our evaluations to this option for these models. This model is introduced in 3.2.2 and evaluated on the three tasks respectively in 4.3, 5.3 and 6.2 ;

2. **Long Short-Term Memory (LSTM)**, based on the deep learning paradigm. The training phase uses the *Stochastic Gradient Descent* (SGD) algorithm and the hyperparameter evaluated is the cell size $\boldsymbol{c}(t)$ (memory capacity). As LSTMs use SGD for training, input features can be pretrained or trained, we evaluated then these models with these two possibilities. This model is introduced formally in 3.2.8 and evaluated on the authorship attribution task in 4.5 ;

3. **Gated Recurrent Units (GRU)**, also based on the deep learning paradigm and the SGD algorithm. They are basically a simplified version of LSTMs. The hyperparameter evaluated is the number of units in the hidden layer $\boldsymbol{h}(t)$. As GRUs use SGD for training, input features can be pretrained or trained, we evaluated then these models with these two possibilities. This model is formally introduced in 3.2.10 and evaluated on the three tasks respectively in 4.5, 5.4 and 6.3 ;

Our approach is to evaluate these models with the different hyperparameters and features on the different tasks with the corresponding datasets and measures. Using these evaluations, we used statistical tests (two-samples bilateral t-test) to find statistically significant difference between these evaluations. Finally, authorship analysis is constrained by the amount of data available which is intrinsically limited in this field. We have therefore chosen tasks with different dataset sizes to determine the capacity of each model to handle a small training dataset.

2.1.2 Method

In this study, we performed experiments related to three different kinds of task in authorship analysis. For each kind we introduce corresponding datasets and performance measures.

1. **Authorship Attribution**, where the goal is to answer the following question : *who is the true author behind a given document ?* Given a set of sample documents written by a closed set of candidate authors. This field will be split in two tasks, the first where the goal is to determine who is the author of a single-authored document. And the second where we want to known the authors of two-authored documents at the sentence level. For this field, we will use the well-known C50 Reuters dataset restricted to 15 authors, with 100 documents per author. We evaluated our model using accuracy and 10-fold cross validation. This task and its dataset are introduced in 2.2 and formally described in 2.2.1. The methodology (measure) and the corpus used are described in 2.2.2. Finally, the results are presented in Chapter 4 ;

2. **Author Profiling**, where the goal is to determine if the author of tweets is a men or a women. This is a two classes problem and we used a dataset of 360,000 tweets (up to 140 characters) written by 3,600 authors created for the PAN17 evaluation lab of the CLEF 2017 conference. We evaluated our model using accuracy and 10-fold cross validation. This task and its dataset are introduced in 2.3 and formally described in 2.3.1. The methodology (measure) and the corpus used are described in 2.3.2. Finally, the results are presented in Chapter 5.

3. In **Author Verification**, one wants to determine if a document has been written by a given author or not. This is a binary classification problem, or one-class problem in machine learning as we want a probability that a document has been written by the given author. In this study we evaluated models on two tasks, the first where we want to find which part of a multi-authored document has been written by a target author. We will refer to this task by the name *Author Extraction* (AE). In the second task, we want to know if the author has collaborated in the writing of a document. We will refer to this task by the name *Author Verification in Multi-Authored Documents* (AVMA). For these tasks we created a dataset, named SFGram, of Science-Fiction magazines written between the 50s and the 70s. We evaluated our model using the F_1 score and 5-fold cross validation. This task and its dataset are introduced in 2.4 and formally described in 2.4.1. The methodology (measure) and the corpus used are described in 2.4.2. Finally, the results are presented in Chapter 6.

In this study, a special attention will be paid to the performance obtained by the different model on small datasets as this is a current issue with neural models. In order to define the complexity of each model and the time they need to give an answer, we will pay attention to the number of parameters that each model has to learn. The number of learned parameters is the number of parameters in a model that are the target of the learning processes. In neural networks, it can be the total number of weights between neurons and biases that can be tuned by methods such as SGD or linear regression. As the final training time depends not only on how many parameters there is to be estimated, but also on the method used to train the network.

For the first two tasks, we used a 10-fold cross validation to evaluate models, and a 5-fold cross validation for the last task. This choice is based on the variation of the performances of the different evaluated models on different test sets. For example, on the authorship attribution task, the estimated accuracy can vary from 88% to 96% for the word-based ESN (ESN-WV). It was therefore necessary to evaluate the accuracy rate for the first two tasks with a cross-validation in order to reduce this variation and to achieve a more accurate evaluation. As we obtained an accurate estimation of models performances, our results can still be compared to results obtained on the same datasets in the literature. For the last task (Author Extraction and Verification), we used a 5-fold cross validation, this is due to the number of documents in the dataset. SFGram is composed of 91 documents and we need to split this dataset in three sets, the training set, the validation set and the test set. The validation set is used to compute an optimal threshold to separate positive

	Positive	**Negative**
Predicted positive	True positive (TP)	False positive (FP)
Predicted negative	False negative (FN)	True negative (TN)

Table 2.1 – Confusion matrix

and negative samples. Using a bigger number of folds, and then a smaller validation set, can seriously impact the performance of a model as the evaluated threshold is of a very poor quality. We then decided to use the 5-fold cross validation instead of the common 10-fold cross validation on this task.

2.2 Authorship Attribution (Reuters C50)

Today, RNNs research focuses mainly on deep architectures which are able to discover long and short range dependencies in data. Whether this property can help in the field of authorship analysis is still an open question. Furthermore, the capacity to model long and short-term dependencies depends on the size of the dataset used for training. Unfortunately, the quantity of data is intrinsically limited in this field. As a result, we wanted to evaluate different RNN models on a classical authorship attribution task using various syntactic and lexical features. This thesis seek to assess the capacity of these methods to perform authorship attribution tasks and the document and sentence-level using the Reuters C50 dataset. To this end, we compared these models to a baseline composed of the state-of-the-art methods in that field. As we restricted our choice to many-to-many architectures, we show how RNNs can be used to project and visualise the attribution process on investigated documents. This characteristics is essential to be useful in the field of authorship attribution.

2.2.1 Task

Authorship attribution is a well-known task in the field of NLP where the objective is to answer the following question: *who is the author of a text or document?* To answer this question, models can based their predictions on linguistic patterns and stylistic markers. This task consists then to find who wrote a new unseen textual document given a corpus of sample documents. It is of particular interest for the fields of humanities, law and intelligence, marketing and computer security. The classical approaches to authorship attribution are based on statistical learning methods and researchers have tried to apply deep learning methods with promising results [121] but also with high complexity, and then with a long training time and a need for larger datasets. However, deep learning methods have faced the lack of sufficiently large datasets available in authorship attribution. Furthermore, the amount of data available per class is intrinsically limited by the amount of textual data that a single person can produce. Consequently, shallower RNN architectures such as vanilla RNNs achieved interesting results [114].

However, there is still no peer-reviewed work on RNNs applied to authorship attribution in the literature and whether deeper models are useful on this task and whether neural networks can be competitive with state-of-the-art methods remain open questions. Authorship attribution is constrained by the size of the dataset [125] but it also requires models which allow a full exploration and visualisation of the evidences they provide. Consequently, exploring the performances on authorship attribution tasks of many-to-many RNNs with shallower architectures is of great relevance.

We selected two tasks to evaluate our models where classifiers have to determine who is the author of a sample document. In the first task, a model has to predict the author of an unknown text in a set of 15 candidate authors. In the second one, it has to predict the author of a sentence contained in an unknown document written also by one of the 15 candidate authors. To build the set of two-authored documents, we concatenated two files written by two different authors to build a new sample document. We then used the resulting 1,500 files and 10-fold cross validation to evaluate our models. As RNNs have a memory component, they can use context to find the author of one sentence. We then compared our models to state-of-the-art models in authorship attribution such as SVMs, Naive Bayes classifier and Random Forest based on most frequent words and Bag-Of-Words (BOW).

2.2.2 Corpus and methodology

The Reuters C50 dataset (Reuter_50_50)[1] is a well-known dataset used in the field of authorship attribution. It is a subset of the RCV1 dataset which is a collection of several thousands documents distributed over size categories and written in six languages (French, English, Italian, Greek, Spanish). All the documents contained in the Reuters C50 dataset are only written in English and they are all extracted from Reuters news. 50 authors were extracted from the RCV1 dataset, according to the total size of articles, with the particularity of having all their texts labelled with at least one subtopic of the class CCAT (corporate/industrial). This selection is an attempt to minimise the topic factor between sample documents. The training and test sets contain both 2,500 documents for a total of 50 documents per author in each set. For our experiments, we merged the training and the test sets and we selected 15 random authors. We then used 10-fold cross validation on this merged dataset to have an accurate evaluation of model performances. We reduced the number of authors to decrease the training time and, consequently, to increase the number of models we can evaluate.

The left-side of figure 2.1 shows the distribution of word counts over the different documents. There is an average of 608 words in the documents with a standard deviation of 163. The longest document has 1,789 words and the smallest 61 words. The right-side of figure 2.1 shows the distribution of total sample sizes (in word count) for each of the 15 authors. Regarding authors, they have in average a sample size of 61,229 words (in word count) for each of the 15 authors. The author with the biggest collection has 71,776 words and the one with the smallest collection 53,141 words. To make our comparisons, we used two measures : *accuracy* and *the number of learned parameters*. We choose these two measures because we think that they cover important aspects of evaluated models. The data used with this task is well balanced, and we therefore think that accuracy is a good measure to evaluate the performances of the different methods. To compare complexity, the number of parameters to be learned is a good choice as it allows to compare complexity and to determine whether a model can be put into application.

For classification tasks, the terms *true positives* (TP), *true negatives* (TN), *false positives* (FP) and *false negatives* (FN) refer to a classifier's predictions compared to a ground truth. Here, terms such as *true* and *false* refer respectively to predictions corresponding, or not, to the ground truth. Formally, if positive and negatives instances are defined as P and N, the four preceding terms can be defined in a confusion matrix as in Table 2.1. In classification, *accuracy* refers to the ratio of documents correctly classified. Using the terms introduced above, *accuracy* is define as,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

2.3 Author Profiling (PAN17)

In the age of big data, a lot of applications are based on a huge amount of various data such as videos, pictures, links, articles and blogs. These data are published directly from a variety of devices such as computers (websites), smartphones or sensors. New platforms of communication such social networks have appeared in the late 2000s. They are based on fast interactions which generate a large variety of content with different characteristics (mention, reply, repost, hashtags, etc). It is difficult to compare these new contents to traditional texts such as books, articles and novels. This raises new questions such as : *Can we establish whether the author of textual data is a woman or a man? Is is possible to identify the place of origin, the age group or the psychological profile of this author?* These questions are important in order to solve current issues of the social network era such as plagiarism, identify theft and fake news.

2.3.1 Task

Author profiling is an important applications related to marketing, forensics and security. For example, it would be interesting for police forces to determine the gender, the age group or the socio-cultural background of the author of harassing messages. Applied to marketing, sellers could use consumer profiles based on the analysis of social network posts to do targeted advertising. To extract these profiles, classical methods based on statistics are used as they have been shown as very efficient for text classification.

[1] https://archive.ics.uci.edu/ml/datasets/Reuter_50_50

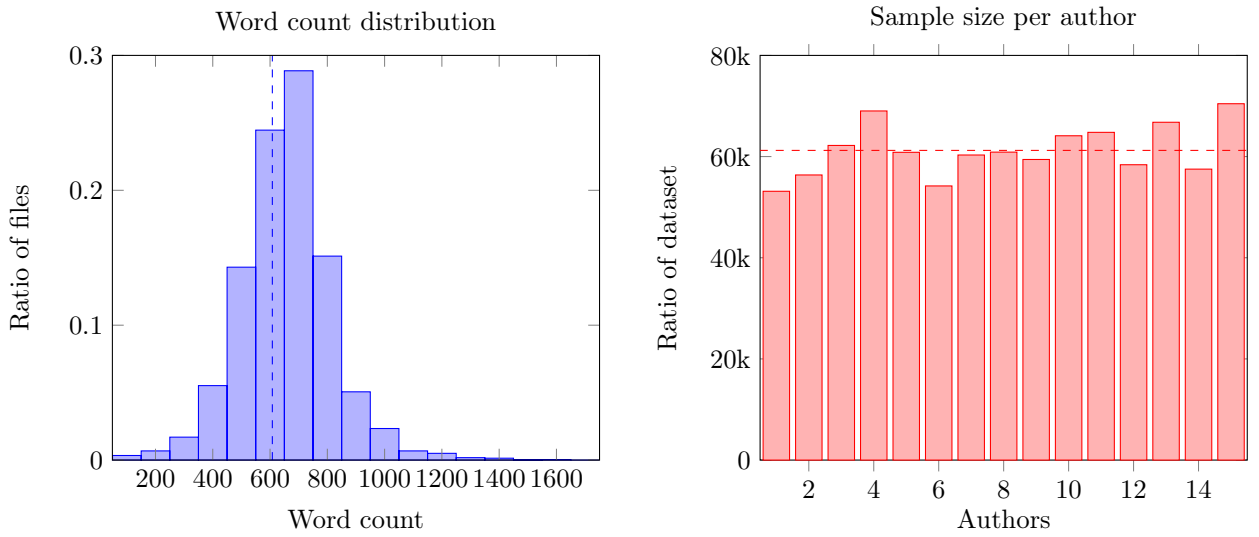


Figure 2.1 – Reuters C50 Corpus statistics. Left: The distribution of the number of words in the 5'000 text samples. Documents have an average word count of 608 and a standard deviation of 163. Right. The distribution of the number of words per authors.

In the last few years, a set of methods from artificial neural networks and grouped under the term deep learning became a breakthrough technology in NLP and computer vision. However, it is still facing many difficulties in the field of author profiling and in-depth work is still missing on the evaluation of neural models on this kind of tasks. We therefore decided to evaluate various RNNs on a related task named gender profiling and to compare them with the state-of-the art methods in addition to another classical neural model. This neural network baseline is based on Convolutional Neural Networks (CNN) and bigrams of characters. The set of traditional methods is composed of SVMs, Random Forest, Naive Baye classifier, all based on combination of n-grams of words and characters, and stylistic features.

2.3.2 Corpus and methodology

In order to compare different models on the gender profiling task, a common ground composed of the same dataset and evaluation measures is necessary. The PAN laboratory of the CLEF conference was launched [126] to fulfil this need and allowed multiple research groups to propose profiling methods and compare them with the same methodology. For the 2017 edition of the PAN CLEF evaluation campaign, four test collections of tweets were created containing text in several languages including English[2]. Based on these corpora, the challenge was to classify Twitter profiles per language variety (e.g., UK vs. US English) and gender. We were then able to use this common ground in this thesis to compare the performances on the gender profiling task of different models. In this study, we restricted our tests to the English collection.

Data were collected by the authors of the dataset on Twitter and are composed of 100 tweets per author for a total of 3,600 authors and 360'000 tweets. For each author, a label indicates the gender (male, female). All authors reside in United-States, Great Britain, Ireland, New Zealand, Australia or Canada. We have exactly 600 authors per country and 1,800 in each class. The overall performance of a model is based on the accuracy on the gender profiling task. The accuracy rate is the number of correctly classified documents divided by the total number of documents in the test set. Based on the properties of the dataset, a random baseline will produce an accuracy rate of 0.5 (or 50%).

2.4 Author Verification in Stream of Text (SFGram)

Author Verification is a well-known task in the field of authorship analysis. Given a set of documents written by a single author, one has to find whether a new unseen text has been written or not by this target author. This is a binary classification problem where the number of candidates is limited to one. This is a harder

[2]https://pan.webis.de/clef17/pan17-web/author-profiling.html

(a) Galaxy Magazine, April 1966

(b) Galaxy Magazine March 1972

(c) Lazarus come forth, Robert Silverberg, April 1966

(d) The Gods Themselves, Isaac Asimov, March 1972

Figure 2.2 – Examples of covers and novels extracted from archives.org and included in the SFGram data after digitalisation.

question than the traditional attribution task because only a single author is provided without a set of possible impostors. Furthermore, the kind of textual data available today on the internet and social networks does not have a structure which allow them to be handled as simple documents. Indeed, communication systems such as Twitter, Facebook and instant messaging are more similar to continuous text streams where the segmentation into paragraphs is sometimes not possible or problematic. Moreover, it is also often difficult to identify boundaries between two text streams. As a consequence, we need systems able to detect such boundaries or events in addition to document classification algorithms.

The problem of author verification is presented in [127], the author suggests using multiple regression models to predict if a document has been written by a given target author. A similar model was used in [128] in conjunction with a statistical learning approach. The most known method for this task is the unmasking method based on SVMs. It was introduced in [129] and use precision, recall and the F_1 score as evaluation measures and a corpus of student essays written in Dutch. These metrics were used in [130] to evaluate the application of particle swarm model to extract sections written by a given author. An effective method was introduced in [131] to transform this one-class classification task to a multi-class problem. This method is based on additional texts reflecting the style of other possible authors (impostors).

The author verification task was present in different PAN@CLEF evaluation campaigns. In the 2011 edition [132], the author identification task included a verification problem with three authors and a corpus of emails. The 2013 version of PAN@CLEF was entirely focused on author verification [133]. Precision, recall and F_1 score were used as evaluation measures in these applications.

2.4.1 Tasks

In this task, we seek to evaluate the performances of the three proposed RNN models on two new tasks where the objective is to identify text passages written by a target author. A second objective is to detect *points of interest* defined as position in a textual stream where the authorship probability is very high. These points can then be used afterwards to detect whether a given author collaborated or not in a multi-authored document. We want to evaluate the capacity of these RNN models to identify a target author in a noisy stream of text. To complete this task, models are based on one-class learning that seeks to draw an optimal threshold circumscribing all positives examples of the true author.

As a dataset, we use science-fiction magazines publicly available through `archive.org` to evaluate our models. *Optical Character Recognition* (OCR) was used to digitise these documents and therefore they contain a high level of errors and misidentification of words. We selected thee authors among all known authors in this corpus, namely Isaac Asimov, Philip K. Dick and Robert Silverberg. These issues were published between 1952 and 1974. The selection of these three authors was based on three criteria: popularity, the number of contributions, and the existence or not of pseudonyms used by one of these authors.

Author	Documents	Words	Ratio
Asimov	22	341,480	4.25%
Dick	25	264,504	3.26%
Silverberg	45	911,219	11.25%
SFGram	91	8,102,853	

Table 2.2 – Number of documents and words per authors included in the dataset. The SFGram line shows the total number of documents and words. The total number of documents does not correspond to the sum of the column because an issue can contain text from more than one author. The ratio is the percentage of words written by the author in the corpus.

Regarding the different tasks, two have been considered. The first seek to answer the question: *Which text passages have been written by author x?* Each magazine issue is handled as a stream of text and the model must determine an authorship score at each time step (measured by word-token). Performances are evaluated using the F_1 score as a metric. The second task consists to respond to: *In the collection, which issue contains text written by author x?* We used interest points for this second task to detect whether or not a target author wrote a passage (paragraph, or sequence of paragraphs) in that issue. Performances are also evaluated using the F_1 score as a metric.

2.4.2 Corpus and methodology

To carry out our evaluation campaign, we created a dataset named SFGram composed of 1,393 science-fiction books, magazines and novels[3]. This dataset contains a total of 7,067 authors. Magazines where extracted from `archive.org` and novels from the Gutenberg project. Magazine issues come from two well-known science-fiction magazines, namely *IF Science Fiction* (IF) and *Galaxy Science Fiction*. Both are American science-fiction magazines but Galaxy was published from 1950 to 1980 and the leading science fiction magazine of that time. Regarding IF magazine, it was published from 1950 to 1974. These two magazines contain sections written by different authors. Although most parts are science-fiction novels, they also contains ads (sometimes in the middle of a novel), readers's mail and editorials. Consequently, the dataset contains an unknown number of authors.

For our evaluation campaign, we selected three famous science-fiction authors who published during the same period: Isaac Asimov, Philip K. Dick, and Robert Silverberg. They are all three American writers who published novels under their own name with the exception of Silverberg who used different pseudonyms. These pseudonyms are not present in the dataset. Figure 2.2 shows examples of two covers of Galaxy Magazine and two corresponding pages containing novels written respectively by Silverberg and Asimov. The first (a and c) comes from the April 1966 issue. The second (b and d) comes from the issue of March 1972. From the SFGram dataset, we extracted magazine issues containing a novel (or part of it) written by at least one of these three authors. In addition to this set of documents, we included two other issues that do not contain any text written by these authors. We end up with a dataset composed of 91 issues for a total of 8 millions words. To build our groundtruth, we segmented manually each issue to tag each part written by the three selected authors. Table 2.2 and Figure 2.2 present the number of documents, words, total ratio and the distribution of issues published during three decades (50s, 60s, 70s) for each author (left). The right side of 2.2 shows the total number of words available for each author. Isaac Asimov is present in 22 issues (or documents) out of the total 91 with 341,480 words for a ratio of 4.25% of the whole dataset. Philip K. Dick wrote in 25 issues with 264,504 words for a ratio of 3.26% of the dataset. Robert Silverberg is present in more issues with a total of 45 issues and 911,219 words for a total ratio of 11.25% of the dataset.

Issues from the two magazines were transformed to raw text through *Optical Character Recognition* (OCR). As a consequence, resulting textual data are highly noisy. Furthermore, it includes a significant amount of ads and promotions. Novels can then be interrupted by unrelated ads or illustration with caption. For example, sample documents are filled with wrongly recognised character or ads such as :

[3] https://github.com/nschaetti/SFGram-dataset

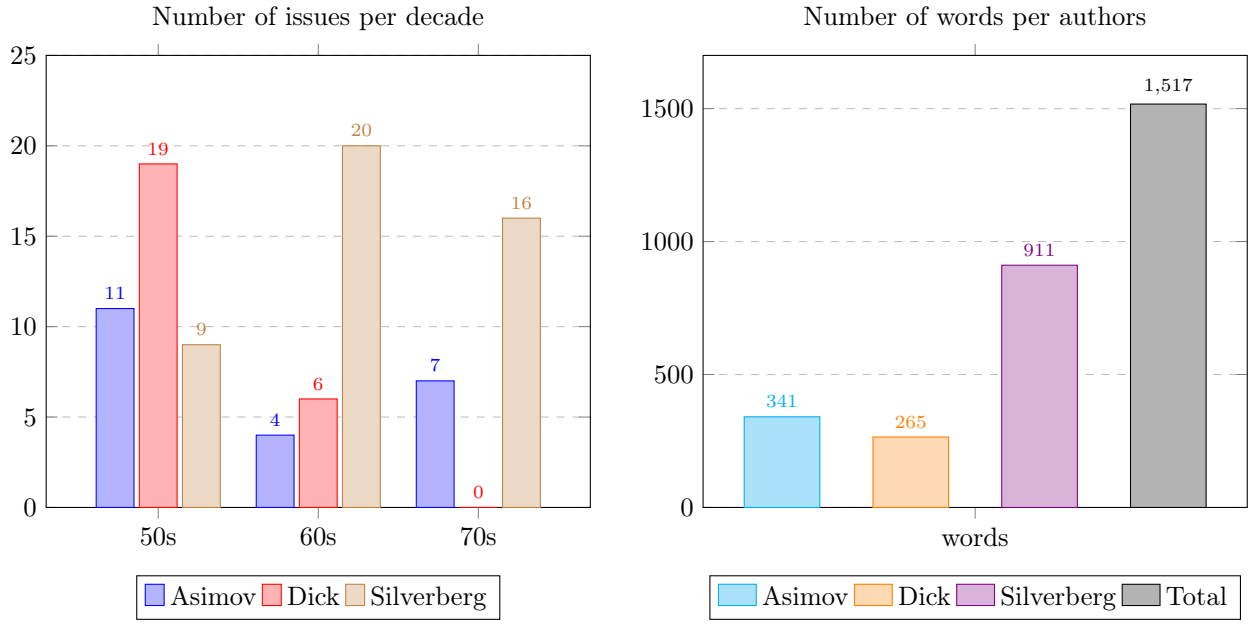


Figure 2.3 – Datasets statistics. *Left*: Distribution of the number of issues per decade for each author. *Right*: Average number of words (in thousands of words) per class for each dataset.

> y AMAZING LOW-PRICE OFFER! on this Mechanics All-Purpose
> /V SOCKET WRENCHnfr
> p/tvC j|,j cgniplete Workshop That You've Always Wanted!

Documents contain various sections such as a table of contents or indexes and multiple authors have collaborated on the edition of an issue. We know from the SFGram dataset that a minimum of 1,308 authors contributed to these 91 issues for an average of 14 authors per document. However, the participation of unknown authors, such as publisher, editors, ads writers and readers intervention in the reader's mail, are not taken into account. As a metric, we used the F_1 score to evaluate our model which is a well-known measure used with binary classifiers. The F_1 score is the harmonic average of the precision and recall and is equal to one when both precision and recall are perfect, zero when they are both null. Formally, the F_1 score is defined as,

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.2}$$

where *precision* and *recall* are defined respectively by,

$$precision = \frac{TP}{TP + FP} \tag{2.3}$$

$$recall = \frac{TP}{TP + FN} \tag{2.4}$$

TP, *FP* and *FN* refer respectively to *true positive*, *false positive* and *false negative*. We used the 5-fold cross validation to get a fair evaluation of these measures. For each fold, we trained a model and computed the output stream of authorship score on validation and test sets. We normalised this output to achieve an average of zero and a standard deviation of one. In the next step, we looked for the best threshold to separate points written or not by the target author using the validation set. We then computed the F_1 measure on the test set. With this threshold, we separated the stream into sections considerer by the model as written or not by the target author.

We used the same method for the second task but in this case we looked for the best threshold to separate documents (and not tokens) in which the author participated. The classification is not done at the token

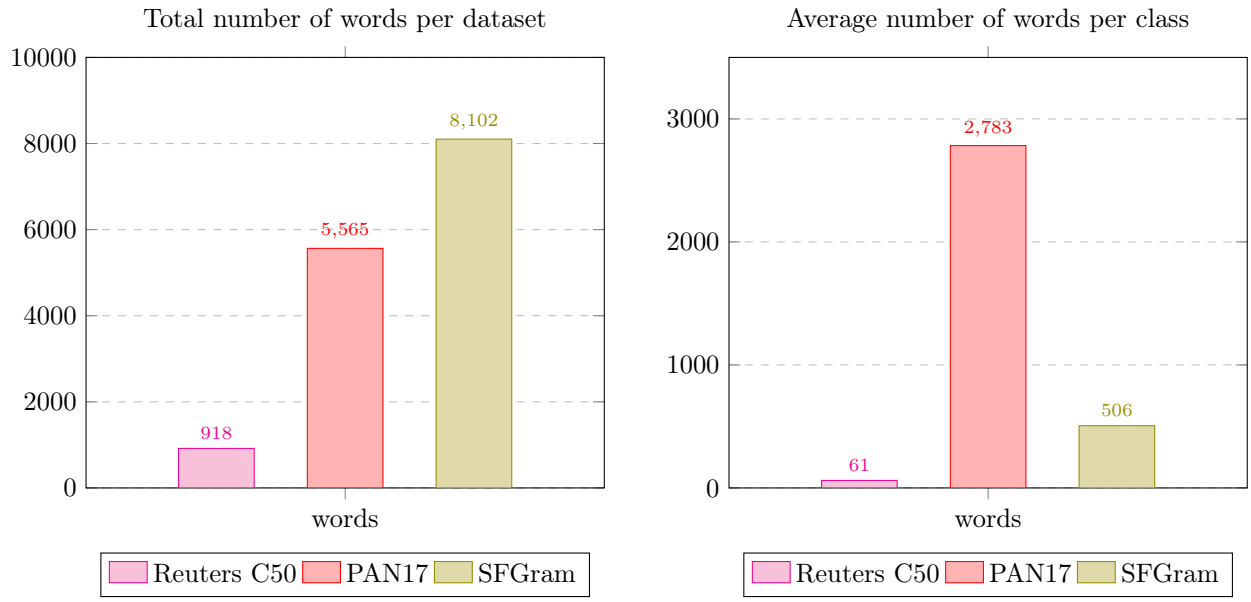


Figure 2.4 – Dataset comparison. Left: the total size (in thousands of words) of each dataset used in this study. Right: total number of words (in thousands of words) available per class.

level (first task) but at the document level and the threshold allows us to predict if an author collaborated in an issue or not.

2.5 Comparison

Figure 2.4 shows the total number of words, and the number of words per class, of each dataset used in this study. The smallest dataset is the Reuters C50, used for the authorship attribution task, with 918,435 words in total. The second dataset by size is the PAN17 dataset used in the author profiling task with 5.57 millions words in total. Finally, the biggest dataset is SFGram, used in the author verification problem, with 8.1 millions words in total.

In terms of words count per class to predict, the Reuters C50 dataset is again the smallest dataset with an average 0f 61,229 words per class (author). For the second and the third biggest dataset, we can observe an inversion as the biggest dataset (SFGram) has only an average of 506,000 words per author against 2.78 millions for each class in the PAN17 dataset. This must be taken into consideration when evaluating the performances of each models on each dataset as we know that models with more parameters need more positive examples to achieve a good estimation of learned parameters. This also show why the F_1 score is more appropriate to the author verification task as the SFGram is not well balanced compared to other datasets and has much more negative than positive samples.

2.6 Reproducibility

In order to allow the complete reproduction of our experiences, the code and tools developed in this study are available online in Github repositories. For this study we used a common deep learning framework named PyTorch and we developed two extensions :

- EchoTorch : a package to exploit and implement Echo State Network with PyTorch[4] ;
- TorchLanguage : a package which gives basic models for text classification and simple access to dataset[5] ;

[4]https://github.com/nschaetti/EchoTorch
[5]https://github.com/nschaetti/TorchLanguage

The two packages are available online and the code used in our experiments is stored in four Github directories, a first one for the authorship attribution experiment related to ESNs[6], a second one for those related to RNNs trained with SGD[7], a third one for the author verification experiment with ESNs[8] and a last one for the author profiling experiments with ESNs and baselines[9]. The SFGram dataset is also available online in a GitHub repository with raw texts, covers and illustrations[10].

[6] https://github.com/nschaetti/RCNLP-authorship-attribution
[7] https://github.com/nschaetti/LSTM-GRU-authorship-attribution
[8] https://github.com/nschaetti/ESN-SF-JCDL
[9] https://github.com/nschaetti/PAN17-author-profiling
[10] https://github.com/nschaetti/SFGram-dataset

Chapter 3

Models and Features

3.1 Introduction

Recurrent Neural Networks (RNN) are neural models having one or several connection cycles (see Figures 3.1 and 3.2). They form a discrete time system with dynamical behaviours and have an internal memory allowing them to compute sequences of data of arbitrary length. Compared to FFNNs which are similar to functions, RNNs are dynamical systems.

The first kind of RNNs were developed in the early 80s. In these models, every neuron is connected to every other network's unit. Some neurons are designed as input neurons, other as outputs and other as hidden. All other RNNs are specific cases of this architecture.

Hopfield networks were developed by John Hopfield [134] in 1982, they have symmetrical connections and can store static patterns by converging to stable states. The network is trained with Hebbian training and the convergence is guarantee. Elman networks [135] are another kind of RNN, structured in 3 layers similar to classical FFNNs, but with additional outer units connected to the 3 layers maintaining a copy of layers values. These networks can solve some tasks related to timeseries prediction which are out of reach of classical FFNNs. The classical method to train RNNs is derived from the classical *Stochastic Gradient Descent* (SGD) algorithm . Unit weights are adapted in an iterative process according to the gradient $\frac{dE}{d\boldsymbol{W}}$, with $\boldsymbol{W}$ the matrix of connection weights, to minimise an error function $E(\boldsymbol{y}, \hat{\boldsymbol{y}})$ between a target signal $\boldsymbol{y}$ to be learned and the model's output $\hat{\boldsymbol{y}}$. A variant of this method often used to train RNNs is the *BackPropagation Through Time* algorithm (BPTT) [136] which can be used to train Elman networks. It consists to unfold a network through temporal steps in order to use classical back-propagation. This method has a temporal complexity of $O(n_h^2)$, where n_h is the number of units in the hidden layer, for each weight update for one output. The *Real-time Recurrent Learning* algorithm (RTRL) [137] is another example of this kind of method where the gradient estimation is done in an iterative way going further in time with complexity $O(n_h^4)$.

A new approach was proposed in [138], the *Atiya-Parlos Recurrent Learning* algorithm (APRL). It allows a faster convergence than previous methods by estimating the gradient $\frac{dE}{d\boldsymbol{x}}$ of the error function E with respect to neuron activations $\boldsymbol{x}$ to push the dynamics in the right direction. A review of gradient descent methods to train RNNs can be found in [138].

RNNs are promising and fascinating because they are able to approximate any dynamical systems to any degree of precision and could be viewed as similar to networks found in the brain. However, RNNs suffer from the problems of vanishing and exploding gradient. In neural networks, each weight is updated using the gradient estimation through partial derivatives of the error function which respect to weights and biases. Unfortunately, the gradient will vanish as the neurons activation function, such as the sigmoid and the hyperbolic tangent, have gradients between zero and one. Backpropagation algorithm using the chain rule will drive the gradient towards zero, the gradient decreasing exponentially across the layers. Formally, if the *spectral radius* $\rho(\boldsymbol{W})$ of the internal matrix $\boldsymbol{W}$ is lower than 1, then

$$\lim_{n \to \infty} W^n = 0 \tag{3.1}$$

In the worst case, this could lead to a complete stop of further training. To avoid this problem, a first solution was found in [139] where the Long Short-Term Memory (LSTMs) were introduced. LSTMs use BPTT and overcome the vanishing gradient issue by using gates allowing or blocking the flow of information.

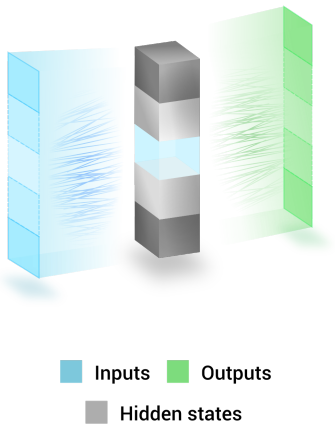


Figure 3.1 – Feed Forward Neural Network

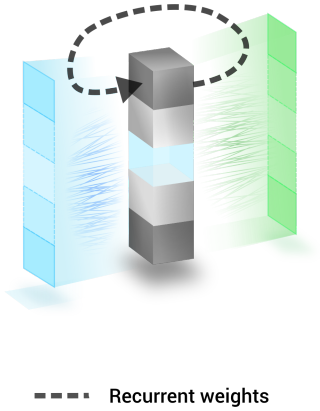


Figure 3.2 – Recurrent Neural Network

LSTMs are the state-of-the-art baseline in several time-related tasks such as speech recognition, handwritten recognition and language modelling. A second solution introduced in [140] consists to create a recurrent layer of randomly connected neurons (called the reservoir) and to train only the output layer with linear methods such as *Ridge Regression* (RR). This kind of network is known under the name of Echo State Network (ESN) and are often enough to get good results in practice.

In authorship analysis, classical approaches are based on statistical methods and researchers have tried to apply neural models with good results but high complexity and thus long training time. Deep learning are known to be very efficient on computer vision tasks such as image or video classification [141]. However when applied to authorship analysis tasks, deep learning have faced many troubles while classical RNNs have been applied successfully to tasks such as language modelling [80], dialog systems [142] and authorship attribution [122]. RNNs are more suited to tasks such as speech recognition or time series forecasting (for example financial markets) where there is a functional dependencies between continuous samples. Furthermore, natural languages are composed of series of tokens where order is important and textual data can be seen a temporal signals where RNNs are able to remember past words and take into account token order. In this section, we will introduce formally each kind of recurrent network used in this study. Section 3.2.2 will be a complete formal introduction to the main model of this study, the Echo State Network (ESN), with all the necessary concepts and properties. Sections 3.2.8 and 3.2.10 will respectively introduce LSTMs and GRUs which will serve as state-of-the-art baselines in deep learning models for document classification.

3.2 Models

3.2.1 Notation

In this document we will use the standard notation used in physics with vectors written in bold lowercase letters while bold uppercase letters will refer to matrices. As we are dealing with temporal signals, we will often use vectorial functions with time t as the only parameter. For example, for all models, the vectorial function $\boldsymbol{x}(t)$ will represent inputs at time t with $\boldsymbol{x}(t) = [x_1(t), \dots, x_{n_x}(t)]$. Similarly, $\boldsymbol{y}(t)$ and $\hat{\boldsymbol{y}}(t)$ will represent the target output to be learned and model outputs at time t respectively. n_x and n_y representing the dimension of respectively the input and the output signal. $\boldsymbol{x}(t)$, $\boldsymbol{y}(t)$ and $\hat{\boldsymbol{y}}(t)$ can be scalar of vector-valued functions depending on the task at hand but we will use the vectorial notation for simplicity. The principle stays the same for hidden states which are refered with the vector-valued function $\boldsymbol{h}(t) = [h_1(t), \dots, x_{n_h}(t)]$. Again n_h indicates the number of units in the hidden layer. Likewise, τ will refer to the length of the signal.

For various uses, these temporal signals can be concatenated into single matrices. In this regard, $\boldsymbol{X}$, $\boldsymbol{Y}$, $\hat{\boldsymbol{Y}}$ and $\boldsymbol{H}$ refer to the matrix containing respectively all inputs, target signals, model outputs and hidden states collected during the training phase (or during the test phase if specified). To refer to a sample in the training/test set, we will use the exponent form, for example $\boldsymbol{X}^{(i)}$ represents the matrix of size $n_x \times \tau^{(i)}$ containing the timeseries from the i-th sample of length $\tau^{(i)}$. If no exponent is present, these matrices contain whole signals collected during training.

For all models, matrices $\boldsymbol{U}$ will represent connection weights between hidden or memory layers and inputs $\boldsymbol{x}(t)$ while matrices $\boldsymbol{W}$ will refer to internal connection weights. The exponent form will be used to differentiate connection weights between various models and flavours. For example, matrix $\boldsymbol{U}$ with no exponent will refer to ESN's input connection weights while $\boldsymbol{U}^f$ will represent input connection weights in the LSTM's *forget gate*. Similarly, vector $\boldsymbol{b}$ will be used to refer to bias and the exponent form will differentiate between different models.

For each model, a table will be given with a list of parameters and their training status (training, pre-trained, fixed, fine-tuned, etc).

3.2.2 Echo State Networks

Classical RNNs are difficult to train by methods based on gradient descent for several reasons:

1. A high temporal complexity slowing down the training process;
2. Instability and bifurcations;
3. Gradient vanishing through time and topology;
4. Local minimas;

It's in a context of slow and painful progress that a new approach called afterwards *Reservoir Computing* has been discovered by researchers in the field of machine learning under the name *Echo State Network* [143] and by Neuroscientists as *Liquid State Machine* [144]. The concept of ESN was introduced in 2001 where a first definition was given. This idea comes from the fact that training only the output layer of an RNN randomly constructed is often enough to get excellent performances in practice. The training of an ESN is thus not only easier, since it is focused only on the output layer, but also because it results in solving a system of linear equations. Furthermore, the key concept is to separate the part where the computing is done and the output layer where the learning is done.

The *Reservoir Computing* paradigm has been discovered thanks to the observation that in the *Atiya-Parlos Recurrent Learning*, the output weights $\boldsymbol{W}^{out}$ change much more rapidly that the ones in the hidden layer. It is from this observation that the *BackPropagation-DeCorellation* (BPDC) algorithm, which consists to apply APRL only to output weights, has been introduced. The advantage is a shorter training time with, however, a bias toward the recent input signal history and a fast forget of input data. Since the rise of neural networks, some work have studied the behaviour and performances of RNNs on authorship attribution [122, 124]. However, no work so far have studied the performances of ESN on such tasks, and therefore, we decided to investigate their ability to perform authorship attribution on a set of 15 authors of the Reuters C50 data set.

ESN has been applied to a large field of scientific domains, from astrophysics to robotic motor control and interaction [145, 146, 147, 148], temporal series forecasting and classification in finance and weather forecasting [149, 150, 151, 152]. Finally, it has been applied to image classification to the MNIST dataset [153, 154] with stacked architectures inspired from deep-learning [155]. RNNs, LSTMs and GRUs have been applied to authorship attribution in [122] and [124] respectively. To our knowledge, this is the first time ESN is applied to NLP, document classification and authorship analysis.

3.2.3 Reservoir Computing and Kernel Functions

A *temporal task* consists of learning a function f with input signal $\boldsymbol{x}(t) = [x_1(t), x_2(t), \ldots, x_{n_x}(t)]$ and output signal $\boldsymbol{y}(t) = [y_1(t), y_2(t), \ldots, y_{n_y}(t)]$, with $t = 1, \ldots, \tau$. The unknown function $f : (\mathbb{R}^{n_x})^\tau \to \mathbb{R}^{n_y}$ to be learned is defined by :

$$\boldsymbol{y}(t) = f(\ldots, \boldsymbol{x}(t-1), \boldsymbol{x}(t)) \tag{3.2}$$

The goal is always to minimise $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ where $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are respectively the signal to be learned and model's output. The temporality lies in the functional dependency between continuous observations which allows us to predict $\boldsymbol{y}(t)$ based on $\boldsymbol{x}(t), \boldsymbol{x}(t-1), \ldots$. We can add a noise vector $\boldsymbol{v}$ to function f,

$$\boldsymbol{y}(t) = f(\ldots, \boldsymbol{x}(t-1), \boldsymbol{x}(t)) + \boldsymbol{v}(t) \tag{3.3}$$

The noise term is also known as the *error term* $\boldsymbol{\epsilon}$ and represents all other factors influencing the dependent variable $\boldsymbol{y}$ other than factors contained in $\boldsymbol{x}$. This error term limits therefore the precision of learning we can achieve with any model. The simplest solution is based on a linear model defined by

$$\boldsymbol{y}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{b}_0 \qquad \boldsymbol{A} \in \mathbb{R}^{n_y \times n_x} \tag{3.4}$$

where $\boldsymbol{A}$ is the parameter matrix containing the *regression coefficients* and $\boldsymbol{b}_0$ is the *intercept term* or *bias*. Most tasks in machine learning (ML) cannot be effectively solved by a simple linear model. The errors $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ staying large for all parameters as the input space cannot be effectively separated by a linear boundary.

The simple idea to overcome this issue is to project non-linearly the input signal $\boldsymbol{x}$ in a latent space of higher dimension $\boldsymbol{h}(t) \in \mathbb{R}^{n_h}$ and to use it with a linear method. This is the idea behind several machine learning methods such as *Support Vector Machine* (SVM) or *Feed-Forward Neural Networks* (FFNN), where the hidden layer is an expansion of higher dimensionality of the input space. This method is part of the set of *kernel* methods and is traditionally defined by a function K which expand the input signal $\boldsymbol{x}$ to the latent space of $\boldsymbol{h}$,

$$\boldsymbol{h}(t) = K(\boldsymbol{x}(t)) \tag{3.5}$$

The choice of the kernel function is done manually by a trial-and-error method. With temporal task, the target signal $\boldsymbol{y}$ to be learned is dependent on the history of the input signal $\boldsymbol{x}$ defined as $\ldots, \boldsymbol{x}(t-2), \boldsymbol{x}(t-1), \boldsymbol{x}(t)$. Therefore, K projects the input history in the new latent space of $\boldsymbol{h}$. To this end, we modify the function K introduced in Equation 3.5 to take into account not only the signal $\boldsymbol{x}$ at current time step t but also its potentially infinite history.

$$\boldsymbol{h}(t) = K(\ldots, \boldsymbol{x}(t-2), \boldsymbol{x}(t-1), \boldsymbol{x}(t)) \tag{3.6}$$

To avoid having a function K with an infinite number of parameters, we define it recursively by replacing in 3.6 the history of input $\boldsymbol{x}$ by the output of K at the previous step ($\boldsymbol{h}(t-1)$).

$$\boldsymbol{h}(t) = K(\boldsymbol{h}(t-1), \boldsymbol{x}(t)) \tag{3.7}$$

Vector $\boldsymbol{h}$ can be seen as a spatial transformation of the history of input signal $\boldsymbol{x}$. We can then use $\boldsymbol{h}$ and a linear method to get an approximation $\hat{\boldsymbol{y}}$ of $\boldsymbol{y}$:

$$\hat{\boldsymbol{y}}(t) = \boldsymbol{W}^{out}\boldsymbol{h}(t) = \boldsymbol{W}^{out}K(\boldsymbol{h}(t-1), \boldsymbol{x}(t)) \tag{3.8}$$

A commonly used method, especially in signal generation task, is to expand the input signal and the last output $\hat{\boldsymbol{y}}$. The model then become,

$$\boldsymbol{h}(t) = K(\boldsymbol{h}(t-1), \boldsymbol{x}(t), \hat{\boldsymbol{y}}(t-1)) \qquad \hat{\boldsymbol{y}}(t) = \boldsymbol{W}^{out}\boldsymbol{h}(t) = \boldsymbol{W}^{out}K(\boldsymbol{h}(t-1), \boldsymbol{x}(t), \hat{\boldsymbol{y}}(t-1)) \tag{3.9}$$

The goal of the training phase is to find weights of the output matrix $\boldsymbol{W}^{out}$ which minimise the error $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$. It is interesting to observe that the function K in Equations 3.7 and 3.8 defines a non-autonomous dynamical system guided by the input signal $\boldsymbol{x}$ which can be implemented by any systems having such characteristics.

3.2.4 A Formal Definition of ESNs

The important idea behind Reservoir Computing (RC) is to use a set of randomly connected artificial neurons as the kernel function K. The main kind of RC model used in this section is known as Echo State Network (ESN) and comes directly from Equation 3.10. Here, the non-linear expansion vector $\boldsymbol{h}(t)$ is defined by

$$\boldsymbol{h}(t) = (1-\alpha)\boldsymbol{h}(t-1) + \alpha g(\boldsymbol{U}\boldsymbol{x}(t) + \boldsymbol{W}\boldsymbol{h}(t-1) + \boldsymbol{b}), \qquad t = 1, ..., \tau \tag{3.10}$$

where $\boldsymbol{h}(t) \in \mathbb{R}^{n_h}$, with n_h the number of neurons in the reservoir, is its activation vector at time t. The matrix $\boldsymbol{U} \in \mathbb{R}^{n_h \times n_x}$, with n_x the dimension of the input signal $\boldsymbol{x}$, represents the weights applied to inputs $\boldsymbol{x}$, and $\boldsymbol{W} \in \mathbb{R}^{n_h \times n_h}$ is the matrix of internal weights. $\boldsymbol{b}$ is the bias vector of size n_h of reservoir's units. $\alpha \in [0, 1]$ is the *leak rate*, the idea behind this parameter is to adapt the network's dynamics to temporal characteristics of the tasks to be learned. We start usually by a null state $\boldsymbol{h}(0) = \boldsymbol{0}$ for the initial vector. The concept of *Echo State Network* was introduced in 2001 by Herbert Jaeger in [140] where a first definition is given.

Parameter	Description	Size / Space	Training
$\boldsymbol{U}$	Input-to-reservoir connection weights matrix.	$n_h \times n_x$	Randomly initialised
$\boldsymbol{W}$	Reservoir-to-reservoir connection weights matrix.	$n_h \times n_h$	Randomly initialised
$\boldsymbol{b}$	Reservoir units bias.	n_h	Randomly initialised
$\boldsymbol{W}^{out}$	Reservoir-to-outputs connection weights matrix.	$n_y \times n_h$	Ridge regression
α	Leak rate allowing to modify the ESN's dynamics.	$\mathbb{R}$	Fine-tuning
λ	Ridge regression regularisation parameter. Higher values push $\boldsymbol{W}^{out}$ weights towards zero.	$\mathbb{R}$	Fine-tuning
$\rho(\boldsymbol{W})$	Spectral radius of the reservoir-to-reservoir matrix $\boldsymbol{W}$.	$\mathbb{R}$	Fine-tuning

Table 3.1 – Overview of all ESN's parameters and notations with description, sizes and related training method.

Definition 3.2.1. *An Echo State Network (ESN) is a recurrent and randomly generator network of artificial sigmoid neurons guided by a uni- or multidimensional temporal signal. Neurons activation are treated with a linear method. The reservoir acts like a complex non-linear dynamical filter which transforms input signals using a high dimensional temporal map [140].*

This idea comes from the observation that training only the output layer of a randomly generated RNN with specific characteristics is sometimes enough to get very good performance in practice. The main idea consists to use a reservoir of neurons as a temporal non-linear expansion function with a memory-less output to approximate $\boldsymbol{y}$, the expansion function acts like a transformation of an *input space* into a *feature space*. The RC method consists then of two components :

1. $\boldsymbol{h}(t)$: expansion vector of the input signal and of its history $\ldots, \boldsymbol{x}(t-1), \boldsymbol{x}(t)$;
2. $\hat{\boldsymbol{y}}(t)$: a linear combination of neurons activation $\boldsymbol{h}(t)$ to approximate the target signal $\boldsymbol{y}(t)$;

The main idea of RC is to separate parts doing the computation and the training. In ESNs, the randomly generated reservoir is the part doing the computation by projecting the input space of $\boldsymbol{x}$ in to the feature space of $\boldsymbol{h}$ in a similar way to the previous kernel function K. However, training is only done on the output layer represented by $\boldsymbol{W}^{out}$. The goal of the expansion vector $\boldsymbol{h}$ is to contain a set of rich features to make training as efficient as possible. However, good features for a task are not necessarily the same for another task. The well-known principle in machine learning known as the *no free-lunch* (NFL), which express the fact that an algorithm with good performance for a set of tasks will have degraded performances on another set of tasks, apply here. Network's outputs $\hat{\boldsymbol{y}}$ are then defined from Equation 3.8 by replacing K with state vector $\boldsymbol{h}$,

$$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{W}^{out}\boldsymbol{h}(t)) \tag{3.11}$$

where $\boldsymbol{W}^{out} \in \mathbb{R}^{n_y \times n_h}$, with n_y the number of outputs, and $\boldsymbol{W}^{out}$ the matrix of output weights between the reservoir and the outputs. Function g is usually the identity function. We can then make two observations. First, the separation between the reservoir and the output where the training is done allows to add outputs while avoiding *catastrophic inference* where a new training invalidate preceding ones. The separation of computing device and outputs allows also a true parallelism where the transformation of the input and its history is done only one time while extracting from these states more information by multiplying outputs. Secondly, the reservoir has the interesting characteristics of being at the same time a processing unit and a working memory. This allows ESNs to have a biological plausibility, confirmed by the independent discovery of *Liquid State Machines* in the field of neurosciences.

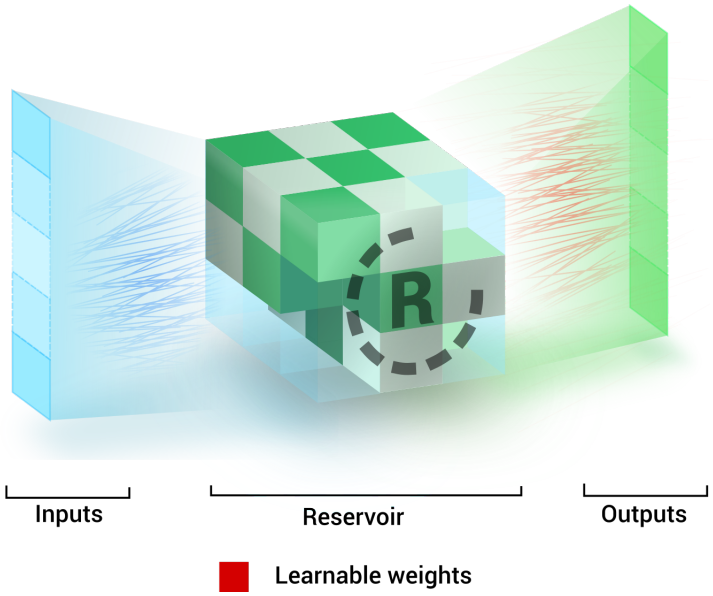


Figure 3.3 – The full reservoir architecture of the Echo State Network. The blue and red lines represent respectively input and learnable connections. In this study, the input dimension (blue squares) is 300 for WV and respectively 10, 30 and 60 for C1, C2 and C3. The five green squares being only used to illustrate the ESN architecture.

3.2.5 Reservoir construction

The original RC method introduced with ESNs in [140] is composed of the following steps :

1. Generate a random reservoir and input connections $(\boldsymbol{W}, \boldsymbol{U})$;
2. Execute Equation 3.10 with inputs $\boldsymbol{x}$ and collect the resulting states $\boldsymbol{h}$;
3. Compute the output weights matrix $\boldsymbol{W}^{out}$ using a linear method with hidden states $\boldsymbol{h}$ collected at step 2 and minimising the Mean Squared Error (MSE) $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ between the output signal $\hat{\boldsymbol{y}}$ and the target signal $\boldsymbol{y}$;
4. Use the trained network on a new signal $\boldsymbol{x}$ by computing the output $\hat{\boldsymbol{y}}$ from output weights found in 3;

Many parameters must be taken into consideration while creating the reservoir. More particularly, it is necessary to ensure the presence of the *Echo State Property* (ESP), which guarantees that the inputs are slowly vanishing through time and are not amplified.

3.2.6 ESN dynamics and reservoir's properties

A reservoir is a dynamical system and can then display chaotic behaviours, in this case the input signal loose all influence in front of initial condition, leading to a drop in performances.

Definition 3.2.2. *The Echo State Property [143] defines that the effect of preceding states* $\boldsymbol{h}(t-1), \boldsymbol{h}(t-2), \ldots$ *and preceding inputs* $\boldsymbol{x}(t-1), \boldsymbol{x}(t-2), \ldots$ *on future states* $\boldsymbol{h}(t+k)$ *must vanish gradually through time* $(k \to \infty)$ *and should not be amplified.*

It is then necessary to establish the conditions for this property to guarantee its presence during the reservoir construction phase (step 1 in 3.2.5). This point must be the object of a particular attention because it has as consequence sub-optimal performances. The first measure introduced to establish a set of conditions is the *spectral radius*.

Definition 3.2.3. *The spectral radius of a matrix* $\boldsymbol{W}$*, defined as* $\rho(\boldsymbol{W})$*, is its highest absolute eigen value.*

In most publications, a *spectral radius* below the unit ($\rho(\boldsymbol{W}) < 1$) is considered as necessary and sufficient for the ESP. However, this is not the case as this condition is necessary and sufficient only for null inputs ($\boldsymbol{x}(t) = \boldsymbol{0}$ for all t). In [156], authors introduce a new sufficient condition for the ESP to hold for all input signal $\boldsymbol{x}$ and give the following method to apply during reservoir generation.

1. Generate a random matrix $\boldsymbol{W}$ with non negative values, $w_{ij} \geq 0$;
2. Multiply $\boldsymbol{W}$ by a factor such as $\rho(\boldsymbol{W}) < 1$;
3. Change the signs of a desired number of $\boldsymbol{W}$ weights to get also negative weights;

In practice, the condition $\rho(\boldsymbol{W}) < 1$ is used most of the time, even if it does not guarantee the *Echo State Property*. Because it is less strict but also because it creates reservoirs with this condition most of the time.

3.2.7 How to train ESNs

The training consists to solve a system of linear equations by minimising the error function $E(\boldsymbol{Y}, \boldsymbol{W}^{out}\boldsymbol{H})$. The matrix $\boldsymbol{W}^{out}$ can be found with a linear method. Training can be defined by the following equation,

$$\boldsymbol{W}^{out}\boldsymbol{H} = \boldsymbol{Y} \tag{3.12}$$

where $\boldsymbol{W}^{out} \in \mathbb{R}^{n_y \times n_h}$ is the output weights matrix, $\boldsymbol{H} \in \mathbb{R}^{n_h \times \tau}$ is the matrix containing the reservoir states $\boldsymbol{h}$ obtained during the training phase (step 2 in 3.2.5), and $\boldsymbol{Y} \in \mathbb{R}^{n_y \times \tau}$ is the matrix containing the target outputs through time, with $n_h < \tau$. Under normal form, Equation 3.12 is expressed by

$$\boldsymbol{W}^{out}\boldsymbol{H}\boldsymbol{H}^T = \boldsymbol{Y}\boldsymbol{H}^T \tag{3.13}$$

A naive solution to find $\boldsymbol{W}^{out}$ is to use matrix inversion.

$$\boldsymbol{W}^{out} = (\boldsymbol{Y}\boldsymbol{H}^T)(\boldsymbol{H}\boldsymbol{H}^T)^{-1} \tag{3.14}$$

The complexity of Equation 3.14 does not depend on the training set size. Another solution is the *Ridge Regression* (RR). This method consists to add an additional cost to the optimisation function to minimise the amplitude of output weights matrix $\boldsymbol{W}^{out}$, reducing its sensibility to noise and to overfitting. These points made the RR a good method of choice for training ESN. Equation 3.14, can be rewritten with the regularisation parameter λ as

$$\boldsymbol{W}^{out} = \boldsymbol{Y}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{H}^T + \lambda\boldsymbol{I})^{-1} \tag{3.15}$$

where $\boldsymbol{I}$ is the identify matrix $\boldsymbol{I} \in \mathbb{R}^{n_x \times n_x}$ and λ is the regularisation factor. The parameter λ does not have any absolute meaning and should be optimised for each reservoir.

3.2.8 Long Short-Term Memory (LSTM)

Long Short-Term Memory were introduced in 1997 in [139] as a solution to the vanishing gradient problem. LSTMs are composed of a cell $\boldsymbol{c}(t)$, of size n_c, which remember values over arbitrary time intervals, of gates $(\boldsymbol{o}(t), \boldsymbol{f}(t), \boldsymbol{i}(t))$ which regulate the flow of information going in and out, and remaining in the cell. And an hidden layer $\boldsymbol{h}$ which is used with output layer(s) to learn the task. The main variety of LSTM is composed of one cell and three gates. The *input gate* $\boldsymbol{i}(t)$ controls the extent to which new information flows into the cell. Formally it is defined as,

$$\boldsymbol{i}(t) = \sigma_g(\boldsymbol{U}^i\boldsymbol{x}(t) + \boldsymbol{W}^i\boldsymbol{h}(t-1) + \boldsymbol{b}^i) \tag{3.16}$$

where $\boldsymbol{i}(t)$ is a vector representing the gate's activation at time t. $\boldsymbol{U}^i$ is the trained connection matrix between the inputs $\boldsymbol{x}$ and the gate. $\boldsymbol{W}^i$ is the trained matrix of connection weights between the gate and the network's hidden state $\boldsymbol{h}$ at time $t-1$. Finally, $\boldsymbol{b}^i$ represents gate's bias. The *forget gate* $\boldsymbol{f}(t)$ controls what information remains in the cell,

$$\boldsymbol{f}(t) = \sigma_g(\boldsymbol{U}^f\boldsymbol{x}(t) + \boldsymbol{W}^f\boldsymbol{h}(t-1) + \boldsymbol{b}^f) \tag{3.17}$$

where $\boldsymbol{f}(t)$ is a vector representing input gate's activation at time t. $\boldsymbol{U}^f$ is the trained connection matrix between the inputs $\boldsymbol{x}$ and the gate. $\boldsymbol{W}^f$ is the trained matrix of connection weights between the gate and

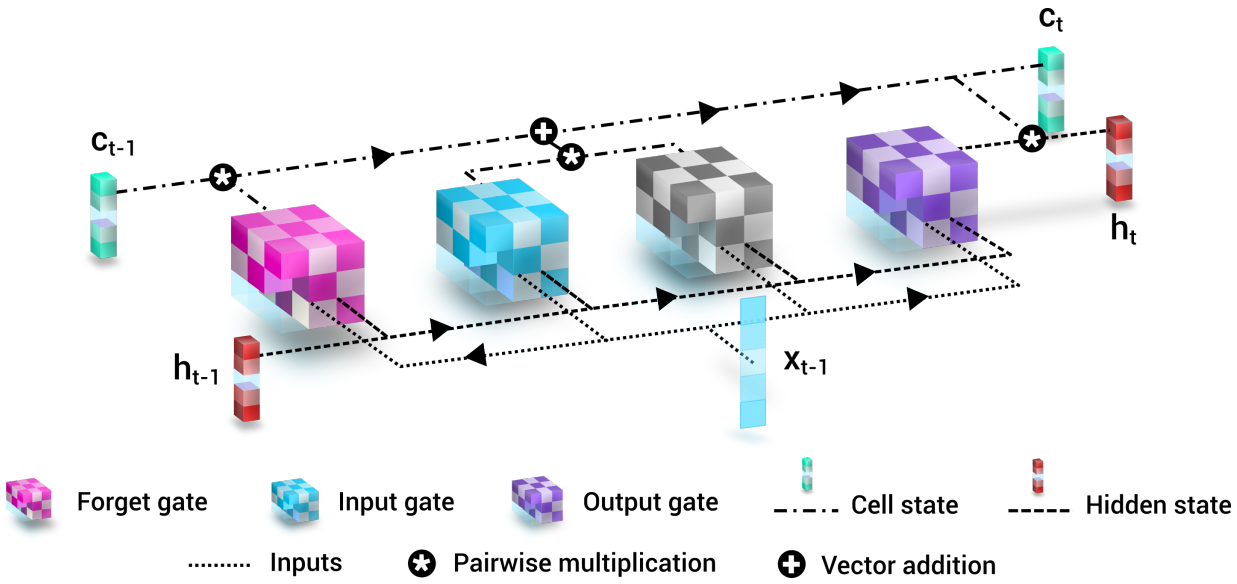


Figure 3.4 – The full LSTM architecture. The pink, blue and purple cubes represent respectively the forget, input and output gates. The red and green bars represent respectively the hidden state and cell vectors.

the network's hidden state $\boldsymbol{h}$ at time $t-1$. Finally, $\boldsymbol{b}^f$ represents gate's bias. The last gate, the *output gate* $\boldsymbol{o}(t)$, selects which value is used to compute the hidden state,

$$\boldsymbol{o}(t) = \sigma_g(\boldsymbol{U}^o\boldsymbol{x}(t) + \boldsymbol{W}^o\boldsymbol{h}(t-1) + \boldsymbol{b}^o) \tag{3.18}$$

where $\boldsymbol{o}(t)$ is a vector representing the output gate's activation at time t. $\boldsymbol{U}^o$ is the trained connection matrix between the inputs $\boldsymbol{x}$ and the gate. $\boldsymbol{W}^o$ is the trained matrix of connection between the gate and the network's hidden state $\boldsymbol{h}$ at time $t-1$. Finally, $\boldsymbol{b}^o$ represents gate's bias.

The gates activation function σ_g is usually the logistic function. This allows gates activation to be between zero and one. Gate activation can then serve as a mask in the definition of cell $\boldsymbol{c}$ and network hidden state $\boldsymbol{h}$. The single cell keeps track of dependencies in the input sequence. It is computed using its last value $\boldsymbol{c}(t-1)$ filtered by the forget gate $\boldsymbol{f}(t)$, to which input values are added after being filtered be the input gate. Formally, the cell is defined as,

$$\boldsymbol{c}(t) = \boldsymbol{f}(t) \circ \boldsymbol{c}(t-1) + \boldsymbol{i}(t) \circ \sigma_c(\boldsymbol{U}^c\boldsymbol{x}(t) + \boldsymbol{W}^c\boldsymbol{h}(t-1) + \boldsymbol{b}^c) \tag{3.19}$$

where $\boldsymbol{c}(t)$ is a vector representing the cell's activation at time t. The cell activation is computed as follow. First, the last cell activation (at time $t-1$) is filtered by the forget gate using the Hadamard product ($\circ$). This filtered activation is added to the Hadamard product of the input gate and a transformation of the inputs $\boldsymbol{x}(t)$ and the last network hidden state $\boldsymbol{h}(t-1)$. $\boldsymbol{U}^c$ is the trained connection matrix between inputs and the cell, and $\boldsymbol{W}^c$ is the trained matrix connection between last network outputs $\boldsymbol{h}(t-1)$ and the cell. σ_c is usually the hyperbolic tangent. Finally, the network outputs are defined by,

$$\boldsymbol{h}(t) = \boldsymbol{o}(t) \circ \sigma_h(\boldsymbol{c}(t)) \tag{3.20}$$

where $\boldsymbol{h}(t)$ is the hidden state at time t defined as the Hadamard product of the output gate $\boldsymbol{o}(t)$ and the nonlinear transformation of the network cell $\boldsymbol{c}(t)$. Figure 3.4 shows the complete LSTM architectures with gates.

Finally, we can use the network's hidden state $\boldsymbol{h}$ to learn a classification task with a linear layer represented by the $\boldsymbol{W}^{out}$ matrix similarly to ESNs.

$$\hat{\boldsymbol{y}}(t) = g(\boldsymbol{W}^{out}\boldsymbol{h}(t)) \tag{3.21}$$

where g is usually a softmax function. The network is trained with a loss function $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ which quantify the difference between network's outputs $\hat{\boldsymbol{y}}$ and the target signal $\boldsymbol{y}$. As the network is trained with the SGD algorithm, a variety of loss function can be used such as Cross Entropy and MSE .

Parameter	Description	Size / Space	Training
$\boldsymbol{U}^f$, $\boldsymbol{W}^f$, $\boldsymbol{b}^f$	*Forget gate* with connection weights from inputs ($\boldsymbol{U}^f$), hidden state ($\boldsymbol{W}^f$) and bias ($\boldsymbol{b}^f$).	$n_c \times n_x$, $n_c \times n_h$, n_c	SGD
$\boldsymbol{U}^o$, $\boldsymbol{W}^o$, $\boldsymbol{b}^o$	*Output gate* with connection weights from inputs ($\boldsymbol{U}^o$), hidden state ($\boldsymbol{W}^o$) and bias ($\boldsymbol{b}^o$).	$n_c \times n_x$, $n_c \times n_h$, n_c	SGD
$\boldsymbol{U}^i$, $\boldsymbol{W}^i$, $\boldsymbol{b}^i$	*Input gate* with connection weights from inputs ($\boldsymbol{U}^i$), hidden state ($\boldsymbol{W}^i$) and bias ($\boldsymbol{b}^i$).	$n_c \times n_x$, $n_c \times n_h$, n_c	SGD
$\boldsymbol{U}^c$, $\boldsymbol{W}^c$, $\boldsymbol{b}^c$	Non-linear transformation of past hidden state and inputs with connection weights from inputs ($\boldsymbol{U}^c$), hidden state ($\boldsymbol{W}^c$) and bias ($\boldsymbol{b}^c$).	$n_c \times n_x$, $n_c \times n_h$, n_c	SGD
$\boldsymbol{W}^{out}$	Hidden layer-to-outputs connection weights matrix.	$n_y \times n_x$	SGD
n_h	Size of the hidden layer. In our study, we have $n_h = n_c$ for all models.	$\mathbb{N}$	Fine-tuning
n_c	Size of memory cell.	$\mathbb{N}$	Fine-tuning
λ	Learning rate.	$\mathbb{R}$	Fine-tuning
Epoch	Number of iteration of the Stochastic Gradient Descent algorithm (SGD).	$\mathbb{N}$	Fine-tuning

Table 3.2 – Overview of all LSTM's parameters and notations with description, sizes and related training method.

LSTMs are very efficient on time-related tasks such as classifying, processing and predicting timeseries. They can deal effectively with the vanishing/exploding gradient and are pretty insensitive to gaps between dependent samples. However, some work showed that LSTM still suffer of the exploding gradient with long sequences. Recently introduced methods such as attention mechanisms are able to deal with this sensibility. LSTMs achieved impressive results and records on different tasks and datasets. In speech recognition, they achieved 95.1% accuracy on the Switchboard corpus [157] and 17.7% on the TIMIT dataset (phonemes) [75]. They are extensively used by companies such as Google, Apple and Amazon, and introduced in several well known products such as Google Translate [158], Siri or Alexa. LSTM are extensively used in speech recognition systems [159], grammar learning [160], handwriting recognition [161], timeseries prediction [162], anomaly detection [163] and semantic parsing [164].

3.2.9 Attention Layer

Attention mechanism was initially introduced in the context of Neural Machine Translation (NMT) with Seq2Seq models. It uses a simple vector computed from the outputs of a dense layer using a softmax function to allow the model to zoom in (or out) and to focus on local (or global) features. In NLP, probabilistic language models are used to assign a probability to each possible sentence.

$$P(w_1 w_2 \cdots w_n) = \prod_i^n P(w_i | w_{i-k} \cdots w_{i-1}) \tag{3.22}$$

where w_i is the word at the i-th position. With recurrent neural networks, we assign a probability conditioned by the hidden state $\boldsymbol{h}$. We then rewrite Equation 3.22 by replacing past words $w_{i-k} \cdots w_{i-1}$ by their encoding by the hidden vector $\boldsymbol{h}(i-1)$.

$$P(w_1 w_2 \cdots w_n) = \prod_i^n P(w_i | \boldsymbol{h}(i-1)) \tag{3.23}$$

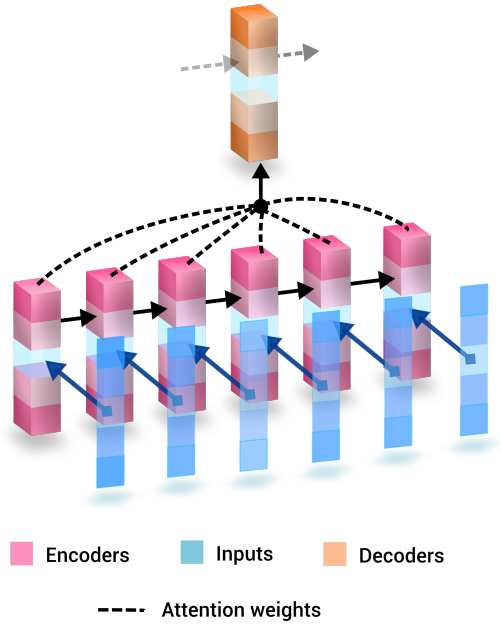


Figure 3.5 – A recurrent model with an attention layer used for sequence classification. At each step, the attention layer can be used to determine which inputs are used by the RNN for its classification.

The hidden state $\boldsymbol{h}$ encodes past words into a single vector which can result in a loss of information as the vanishing/exploding gradient problem limits the modelling of long dependencies. The key idea behind attention mechanism is to plug a context vector between encoder states and the output. This allow RNN to capture global information rather than solely infer based on a single hidden state. The context vector $\boldsymbol{q}(t)$ ($\mathbb{R}^{n_h}$) is computed from attention weights $\boldsymbol{a}(t)$ ($\mathbb{R}^{n_q}$), with n_q being context size, which allow the model to explicitly focus on certain parts of the input. As shown by the figure 3.5, the context vector $\boldsymbol{q}$ is the weighted summation of the hidden state $\boldsymbol{h}$ and the attention weights $\boldsymbol{a}$.

$$\boldsymbol{q}(t) = \boldsymbol{H}(t)\boldsymbol{a}(t) \tag{3.24}$$

Where $\boldsymbol{H}(t) = [\boldsymbol{h}(t - n_q), \boldsymbol{h}(t - n_q + 1), \ldots, \boldsymbol{h}(t)]$ is the concatenation of the hidden states in context of size n_q with $\boldsymbol{H} \in \mathbb{R}^{n_h \times n_q}$. Attention weights $\boldsymbol{a}(t) = [a_1(t), \ldots, a_{n_q}(t)]$ are the result of scores $\boldsymbol{sc}(t)$, of size n_q, normalised with a softmax function.

$$a_q(t) = \frac{exp(sc_q(t))}{\sum_{i=1}^{n_q} exp(sc_i(t))} \tag{3.25}$$

The attention weights sum to one and are in $[0, 1]$. The score vector $\boldsymbol{sc}(t)$ is obtained from,

$$\boldsymbol{sc}(t) = \boldsymbol{S}\boldsymbol{h}(t) \tag{3.26}$$

where $\boldsymbol{S} \in \mathbb{R}^{n_q \times n_h}$ is the score matrix learned by the model. The attention weights then depend on the context itself and select which contextual information to use to model the probability distribution. The final attention vector $\boldsymbol{s}(t) \in \mathbb{R}^{n_h}$ is the context vector computed by the hyperbolic tangent.

$$\boldsymbol{s}(t) = tanh(\boldsymbol{q}(t)) \tag{3.27}$$

This attention vector is used to compute the learned output $\hat{\boldsymbol{y}}(t)$ which, in the context of document classification, is not the distribution over next word but over the candidate classes at time t.

$$P(c_i(t)|w_1 w_2 \cdots) = P(c_i(t)|\boldsymbol{s}(t)) \tag{3.28}$$

Where $P(c_i(t))$ is the probability that the token at position t belongs to the i-th class. The attention weights can then be used to create an attention map (see Figure3.6) showing the weights learned by the attention layer which can be investigated to understand what the model is focusing on.

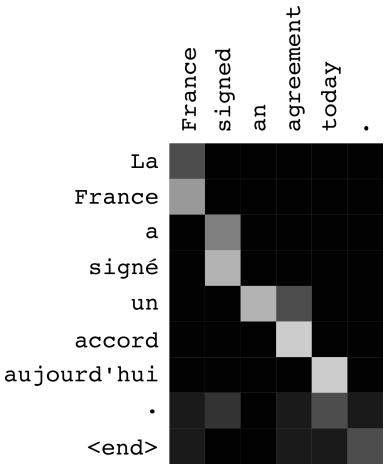


Figure 3.6 – Attention map on language translation.

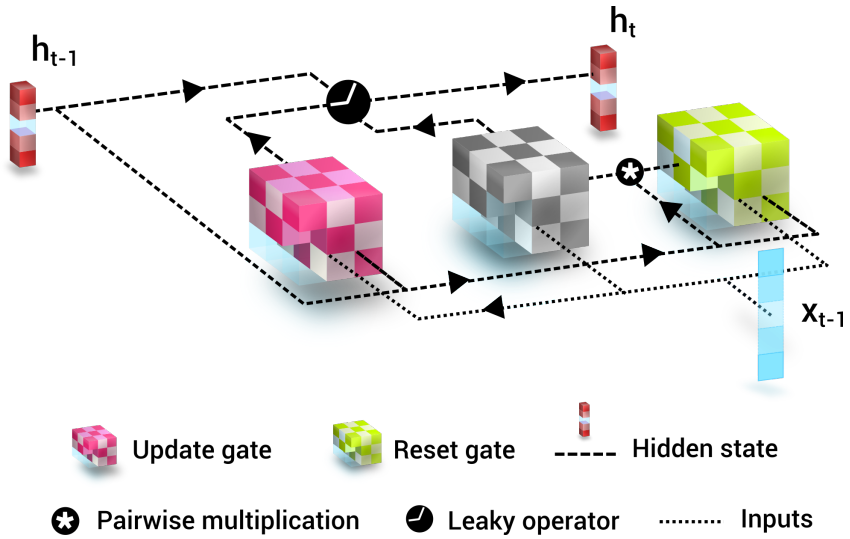


Figure 3.7 – The full GRU architecture. The pink and yellow cubes represent respectively the update and reset gates. The red bar represents the hidden state vector.

3.2.10 Gated Recurrent Units (GRU)

Gated Recurrent Units (GRU) are simplification of LSTMs. They were introduced in 2014 in [78]. Compared to LSTMs, GRUs have *update* and *reset* gates, in place of forget and input gates in LSTM, but no output gate. They have also no memory cell and only an hidden layer $\boldsymbol{h}$. Consequently, GRUs have two gates, the *update gate* $\boldsymbol{z}$ is similar to LSTM's input gate $\boldsymbol{i}$ and controls the extent to which new information flow into the memory $\boldsymbol{h}$. Formally, it is defined as,

$$\boldsymbol{z}(t) = \sigma_g(\boldsymbol{U}^z \boldsymbol{x}(t) + \boldsymbol{W}^z \boldsymbol{h}(t-1) + \boldsymbol{b}^z) \tag{3.29}$$

where $\boldsymbol{z}(t)$ is a vector representing the update gate's activation at time t. $\boldsymbol{U}^z$ is the trained connection matrix between inputs $\boldsymbol{x}$ and the gate. $\boldsymbol{W}^z$ is the trained matrix of connection weights between the gate and the network's hidden state $\boldsymbol{h}$ at time $t-1$. Finally, $\boldsymbol{b}^z$ represents gate's bias.

The *reset gate* $\boldsymbol{r}$ decides how much of the past information to forget. It is formally defined as,

$$\boldsymbol{r}(t) = \sigma_g(\boldsymbol{U}^r \boldsymbol{x}(t) + \boldsymbol{W}^r \boldsymbol{h}(t-1) + \boldsymbol{b}^r) \tag{3.30}$$

where $\boldsymbol{r}(t)$ is a vector representing the gate's activation at time t. $\boldsymbol{U}^r$ is the trained connection matrix between inputs $\boldsymbol{x}$ and the gate. $\boldsymbol{W}^r$ is the trained matrix of connection weights between the gate and the network's outputs $\boldsymbol{h}$ at time $t-1$. Finally, $\boldsymbol{b}^r$ represents gate's bias. The hidden state is defined by,

$$\boldsymbol{h}(t) = (1 - \boldsymbol{z}(t)) \circ \boldsymbol{h}(t-1) + \boldsymbol{z}(t) \circ \sigma_h(\boldsymbol{U}^h \boldsymbol{x}(t) + \boldsymbol{W}^h(\boldsymbol{r}(t) \circ \boldsymbol{h}(t-1)) + \boldsymbol{b}^h) \tag{3.31}$$

The *update gate* decide what information from past state $\boldsymbol{h}(t-1)$ and current state $\boldsymbol{h}(t)$ updated with inputs is incorporated into the next state $\boldsymbol{h}(t+1)$. The reset gate $\boldsymbol{r}$ can block information from past state to keep only new values. GRUs have similar performances to LSTM on tasks such as speech recognition and better performances on smaller datasets. However, [165] shows that GRUs cannot perform some tasks, such as unbounded counting, while LSTMs can. GRU are also unable to learn simple language, a task easily done with LSTMs [165], and are outperformed by LSTMs on machine translation tasks [166]. However, GRUs have fewer parameters and outperforms LSTMs on smaller dataset, a particularly interesting point in authorship analysis.

3.3 Features

Every model are based on a specific way to represent textual data. Over the years, different methods have been proposed to represent texts and documents, some based on words, characters or structured meta-data. Section 1.4 gives an overview of different features used in authorship analysis. In this study, we are not only interested in evaluating different RNN models, but also to determine which kind of text representations is the most effective for these tasks and these models. As a result, we will introduce in this section the different features and representations that we will use and how they are extracted or computed. Features used in this study are categorised in two classes.

1. Syntactic features composed of Part-Of-Speech (POS) and function words (FW) ;
2. Lexical features composed of word-based features and character-based features ;

Two representations are also introduced :

1. One-hot representation ;
2. Embedding layers commonly used with neural network ;
3. Pretrained deep feature extractor taking sequence of tokens and reducing it to a single vector ;

Parameter	**Description**	**Size / Space**	**Training**
$\boldsymbol{U}^z$, $\boldsymbol{W}^z$, $\boldsymbol{b}^z$	*Update gate* with connection weights from inputs ($\boldsymbol{U}^z$), hidden state ($\boldsymbol{W}^z$) and bias ($\boldsymbol{b}^z$).	$n_h \times n_x$, $n_h \times n_h$, n_h	SGD
$\boldsymbol{U}^r$, $\boldsymbol{W}^r$, $\boldsymbol{b}^r$	*Reset gate* with connection weights from inputs ($\boldsymbol{U}^r$), hidden state ($\boldsymbol{W}^r$) and bias ($\boldsymbol{b}^r$).	$n_h \times n_x$, $n_h \times n_h$, n_h	SGD
$\boldsymbol{U}^h$, $\boldsymbol{W}^h$, $\boldsymbol{b}^h$	Non-linear transformation of the past hidden state filtered by the *reset gate* and inputs with connection weights from inputs ($\boldsymbol{U}^h$), hidden state ($\boldsymbol{W}^h$) and bias ($\boldsymbol{b}^h$).	$n_h \times n_x$, $n_h \times n_h$, n_h	SGD
$\boldsymbol{W}^{out}$	Hidden layer-to-outputs connection weights matrix.	$n_y \times n_x$	SGD
n_h	Size of the hidden layer.	$\mathbb{N}$	Fine-tuning
λ	Learning rate.	$\mathbb{R}$	Fine-tuning
Epoch	Number of iteration of the Stochastic Gradient Descent algorithm (SGD).	$\mathbb{N}$	Fine-tuning

Table 3.3 – Overview of all GRU's parameters and notations with description, sizes and related training method.

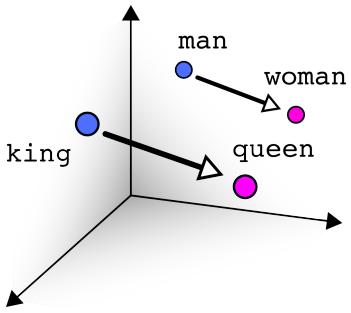


Figure 3.8 – Male-Female

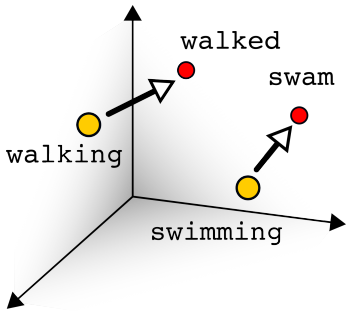


Figure 3.9 – Verb tense

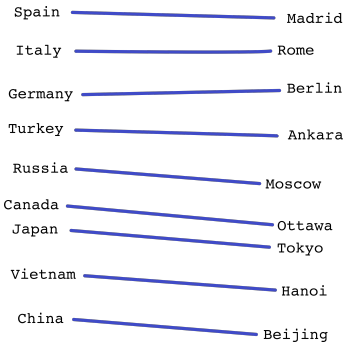


Figure 3.10 – Country-Capitals

3.3.1 How to represent textual data ?

The first and straight method to represent textual data is to assign a distinct index to each item w of our vocabulary V. To illustrate our point we will use a toy example with a vocabulary composed of four words ($|V| = 4$), such as

I = 1
like = 2
dog = 3
cat = 4

The input sentence "I like dog" of length $\tau = 3$ would be represented as $\boldsymbol{x} = [1, 2, 3]$. However, this kind of discrete representation is difficult to use with models such as neural networks and introduce an ordering between words. The key idea behind *One-Hot* vectors is to project each item to a $|V|$-dimensional space in which each component represents the i-th item of the vocabulary V. Our vocabulary would then become,

I = $[1, 0, 0, 0]$
like = $[0, 1, 0, 0]$
dog = $[0, 0, 1, 0]$
cat = $[0, 0, 0, 1]$

We are then able to present the input sentence "I like dog" as a matrix of dimension $|V| \times \tau$ where $|V|$ is the size of the vocabulary and τ the length of the sentence.

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \boldsymbol{x}(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{x}(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \boldsymbol{x}(3) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.32}$$

With recurrent neural networks, each vector $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots$ composing the sentence would be presented to the model one after the other.

However, *One-Hot vector* representation have some serious limitation. First, resulting vectors are highly dimensional and sparse. On most application, the vocabulary size can be bigger than 100,000 and the resulting vectors would be 100,000-dimensional with 99,999 zeros. As the curse of dimensionality states, it will be very difficult for neural networks to generalise and manage this kind of representation on very large vocabulary. Secondly, this simple embedding does not store any contextual information about words. For example, the distance between two words would be exactly the same ($d(dog, cat) = d(I, cat)$). Consequently, we need a more advanced embedding, a representation, which encodes similarity by mapping words *dog* in a smaller space near words such as *cat* as they share context and similarity. A solution is to compute an embedding matrix $\boldsymbol{W}^e$ mapping each word to a small embedding space while encoding similarities. This similarity constraint is formally defined as,

$$d(\boldsymbol{W}^e \times dog, \boldsymbol{W}^e \times cat) < d(\boldsymbol{W}^e \times dog, \boldsymbol{W}^e \times like) \tag{3.33}$$

Where $\boldsymbol{W}^e$ is a matrix of size $n_e \times |V|$ with n_e the size of the embedding space. The embedding matrix

Tag	Description	Example
ADJ	Adjective	Apple
ADP	Preposition	in, to, during
ADV	Adverb	Firmly
CCONJ	Conjunction	for, and, nor, but
DET	Determinant	The
INTJ	Interjection (part of an exclamation)	Ouch, hello
NOUN	Noun	cat, tree
NUM	Number	1, billion
PART	Particle	's, not
PRON	Pronoun	I, me, we
PROPN	Proper noun (specific individual, space or object)	London
PUNCT	Punctuation	;
SYM	Symbols	$
VERB	Verb	is, be
SPACE	Space	
X	Unknown	xfgh

Table 3.4 – Overview of all parts of speech used as features.

$\boldsymbol{W}^e$ can be learned by the neural network and updated while training, the embedding matrix being a part of model's parameters. Or it can be pretrained by a neural network on an self-supervised task where the model has to predict the next word and used afterwards as a non-trainable layer. An example is presented in Section 3.3.3. From our vocabulary, an embedding matrix $\boldsymbol{W}^e$, with $n_e = 2$, and the input matrix $\boldsymbol{X}$ defined in 3.32 would be,

$$\boldsymbol{W}^e = \begin{bmatrix} 0.5 & 0.1 & 0.3 & 0.3 \\ 0.1 & 0.2 & 0.3 & 0.25 \end{bmatrix}, \boldsymbol{E} = \boldsymbol{W}^e \times \boldsymbol{X} = \begin{bmatrix} 0.5 & 0.1 & 0.3 \\ 0.1 & 0.2 & 0.25 \end{bmatrix} \tag{3.34}$$

We can then use our trained representation to map a vocabulary of several hundreds of thousands words to a 2-dimensional embedding space. Furthermore, the embedding matrix encodes similarities and relationships between words as shown in figures 3.8, 3.9 and 3.10. For example, it allows us to ask question such as *"King" is to "Queen" as "Man" is to X?* formally as,

$$King - Queen \sim Man - X \tag{3.35}$$

In this study, we will use *one-hot* vectors for low-dimensional features such as Part-of-Speech (POS) and embedding representations for features with large vocabulary such as words and characters. The embedding matrix will be learned during training or pretrained, these two options being an additional hyperparameter to evaluate.

- If **pretrained**, the model will use an embedding matrix pre-computed with a common method named *Glove* using co-occurrence matrices from a large corpus [45] ;
- If **trained**, the embedding matrix $\boldsymbol{W}^e$ is part of model's parameters and learned during the training phase with SGD. The embedding space size n_e being part of model's hyper-parameters. This is only available to model trained with SGD and will then only be used with LSTMs and GRUs, but not with ESNs ;

3.3.2 Syntactic features

In linguistics, the syntax refers to the set of rules that govern the structure of sentences in a language, more specifically how words are ordered. In this study, we evaluated two syntactic features, the part-of-speech (POS) and function words (FW).

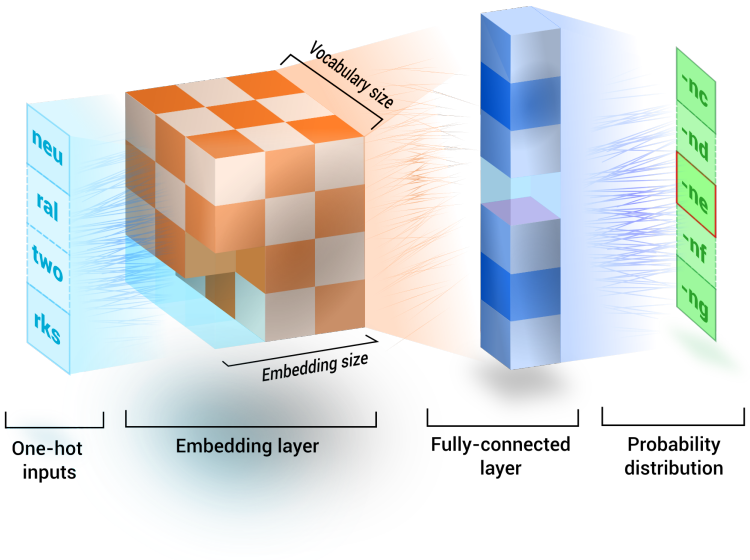


Figure 3.11 – The full architecture of the four layers FFNN used to train character embedding. Light blue squares are the four input trigrams. In orange and blue are respectively the embedding layer (size 50) and the fully-connected layer. The final green squares represent the output probability distribution for each trigram in $|V|$ by a softmax function.

Part-Of-Speech A Part-Of-Speech (POS) refers to a category of words having similar grammatical characteristics and taking place in the same syntactic structures. Common parts of speech are noun, adjective, adverb, verb, proposition, pronoun, conjunction and interjection. We added numbers, punctuation, and other categories described in table 3.4 to end up with 16 categories. In our study, we transformed textual documents in series of POS tags which were then transferred to neural models as one-hot inputs.

$$\boldsymbol{x}(t) = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ \vdots \end{pmatrix} \begin{matrix} noun \\ verb \\ adjective \\ adverb \\ pronoun \end{matrix} , \boldsymbol{W}^e = \begin{pmatrix} \mathrm{e}_{1,1} & \cdots & \mathrm{e}_{1,16} \\ \mathrm{e}_{2,1} & \cdots & \mathrm{e}_{1,16} \\ \mathrm{e}_{3,1} & \cdots & \mathrm{e}_{1,16} \\ \vdots & \ddots & \vdots \\ \mathrm{e}_{n_e,1} & \cdots & \mathrm{e}_{n_e,16} \end{pmatrix} \begin{matrix} noun \\ verb \\ adjective \\ noun \end{matrix} \tag{3.36}$$

Where $\boldsymbol{x}(t)$ is the input vector at time t. The one-hot input representation can then transformed to continuous representations by an embedding matrix $\boldsymbol{W}^e$ (with $|V| = 16$) for models using SGD for training (LSTM and GRU). Table 3.4 shows the different POS categories used and corresponding examples.

Function words In linguistics, function words are defined as words with little lexical meaning and expressing some grammatical relationships between words. Consequently, they are an important parts in the structure of languages. They are opposed to content words which name objects and their properties. In the English language, about 99.9% of words are content words but function words are used at much higher rate as any English text is made of 50% of function words. This is due to the fact that function words bind content words together which introduce an interdependence between function and content words [167]. We transformed textual data to vector representations by keeping only function words in distributed or one-hot representations.

3.3.3 Lexical features

Words As presented in 3.3.1, words are represented using trained or pretrained embedding layers which map vocabulary items to dense vectors of real numbers. Different methods exist to pretrain embedding matrices. The most commonly used are based on dimensionality reduction of word co-occurrence matrix,

	n_q	n_e	$\|V\|$
C1	2	10	4,635
C2	2	30	32,908
C3	2	60	167,026

Table 3.5 – Summary of each character embedding pre-trained on Wikipedia. The context is the number of n-grams on each side used for prediction.

such as GloVe [45], or on neural networks, such as Word2Vec [43] and Skip-gram [38]. In this study, we used mainly GloVe as continuous vector representations to initialise pretrained embedding layers. The vocabulary contains 1.1 million word and there is only 4.3% of Out-Of-Vocabulary items (OOV) which are removed from inputs. Each word is mapped to a continuous vector of dimension 300.

Characters To test our model with character-based features, we pretrained character embedding using a model similar to *Word2Vec* where a FFNN has to predict a token from its context. Here, we want to predict the character (C1), character bigram (C2) or trigrams (C3), from surrounding tokens on both sides of the target token. As shown by figure 3.11, in the case of C3, with the input text "*neural-networks*" and a context of size 2 ($n_q = 2$), the neural network has to predict the trigram "*-net*" from the context "*neural*" and "*works*".

To this end, we trained a FFNN, with a fully connected layer and the ReLU activation function, on top of an embedding layer, and a softmax function as outputs which represent the probability distribution over possible tokens. This pretrained network has the following architecture :

1. An input layer of size $4 \times |V|$, one input for every two tokens on each side. Taking the example above, the four input tokens would be "*neu*", "*ral*", "*two*" and "*rks*" ;
2. An embedding layer with $n_e = 10$ for C1, $n_e = 30$ for C2 and $n_e = 60$ for C3 ;
3. A fully connected layer with a ReLU non-linear activation function ;
4. A softmax function as output of size $|V|$. One output for each possible possible token. Taking the example above again, this output would be trained to have an output probability of 1.0 for "*-ne*" and zero for other trigrams ;

This model outputs a probability distribution over possible tokens. We trained our model on 230 million examples extracted from Wikipedia for 20 iterations with SGD, a learning rate of 0.001, and negative log-likelihood (NLL) as loss function. The performance at each iteration is evaluated with perplexity. We then used the embedding layer as fixed inputs of our ESN model. Figure 3.11 shows the architecture of the feed-forward model used to pretrain character embedding. Table 4.2 describes the three character-based embedding (C1, C2, C3) we trained along their context size (n_q), dimension (n_e) and vocabulary size $|V|$.

3.3.4 Feature extractors

To evaluate the impact of input dimensionality reduction on ESN performances, we trained a feature extractor on data with no temporal dimension and then used it as an input layer. Here we used a character-based deep FFNN trained first with the classical SGD algorithm. The ESN is then trained with standard linear methods and with the pretrained feature extractor as input. We will refer to this deep architecture as CE.

It takes a series of 20 raw characters as input and proceeds as follow :

1. An embedding layer of dimension $n_e = 50$;
2. A fully connected layer composed of 300 features with a ReLU non-linearity ;
3. A softmax function as output with the number of dimension depending on the number of classes ;

The model is trained with a learning rate of 0.001 for 300 iterations and Cross-Entropy as loss function. Once trained, the output layer is removed, and the linear layer is used as a feature extractor for the ESN inputs. Figure 3.12 shows the complete architecture of the ESN model with CE as inputs.

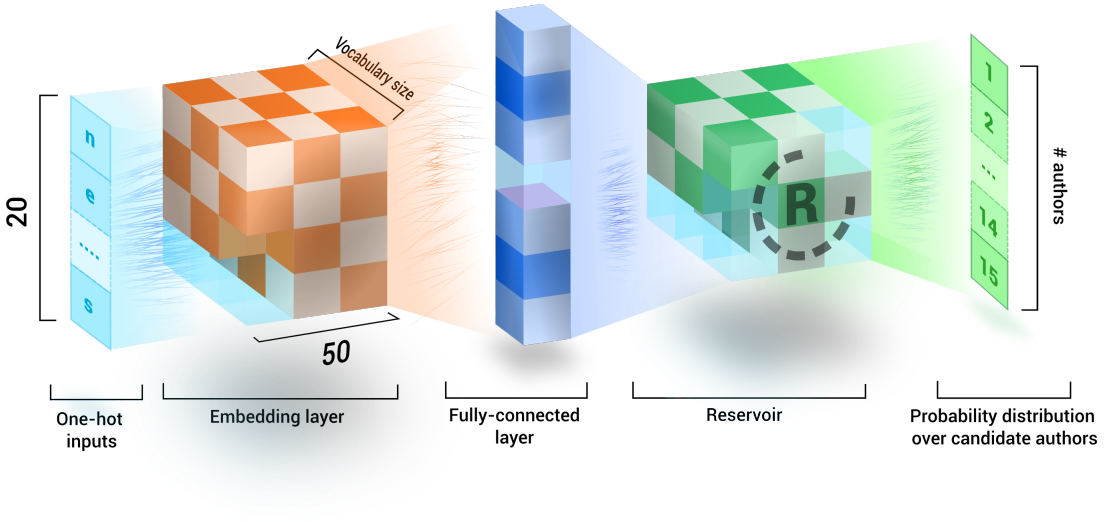


Figure 3.12 – Full view of an Echo State Network (ESN) architecture with a deep character encoder (CE). The input layer is in light blue and accepts twenty consecutive characters. Fully connected, embedding and ESN layers are respectively in blue, orange and green. Embedding and fully-connected layers are trained with classical SGD algorithm and then used as inputs for the ESN layer. The output layer (in green) is the probability distribution over the set of authors.

3.4 Overview

Table 3.6 shows an overview of the different features evaluated in this study along with its type (syntaxic, lexical, etc), its dimension (embedding size n_e or vocabulary size $|V|$) and the training method (trained, pretrained, one-hot) used for each model.

Feature	**Type**	n_e or $\|V\|$	**ESN**	**LSTM**	**GRU**
POS	Syntaxic	16	one-hot	pretrained, trained	pretrained, trained
FW	Syntaxic	250	one-hot	pretrained, trained	pretrained, trained
WV	Lexical	300	pretrained	pretrained, trained	pretrained, trained
C1	Lexical	30	pretrained	pretrained, trained	pretrained, trained
C2	Lexical	60	pretrained	pretrained, trained	pretrained, trained
C3	Lexical	90	pretrained	pretrained, trained	pretrained, trained
CE	Feature extractor	300	pretrained		

Table 3.6 – Overview of all features and models explored in this study with properties including the type, the dimension and the different training procedure.

Part II

Models evaluation

Chapter 4

Authorship Attribution

4.1 Introduction

In this chapter, we seek to evaluate models introduced in the previous chapter and analyse their behaviours in relation to various features and parameters on the authorship attribution task using the Reuters C50 dataset. In Section 4.2, we show how evidences collected by many-to-many RNNs can be visualised and evaluated with practical examples. In Section 4.3, we look for right parameters for ESN models. The leak rate is of particular interest for our study as it defines dynamical properties of highly-dimensional and temporal representations of text. We also explore how training size and reservoir influence ESN's performances. Sections 4.4 explore the impact of different dataset sizes and differences between features. In Section 4.5, we evaluate the performances of LSTMs and GRUs based on trained and pretrained features and the cell size parameter is determined for the different features. In Section 4.6, we compare the different recurrent architectures (ESNs, LSTMS, GRUs). Section 4.7 show how we can retrieve a certainty measure on the prediction made by the different models. Finally, the final sections 4.8 and 4.9 compare all evidences collected in this chapter and draw the necessary conclusions.

4.2 Visualisable evidences

The output $\hat{y}_a(t)$ of the RNN is the probability that tokens in its memory at time t were written by author a. As a result, we have then an output temporal signal of authorship probabilities. As we are dealing with document classification, we choose the predicted $\hat{a}$ of a document as the one with the maximum average probability over time,

$$\hat{a} = \max_a \langle \hat{y}_a(t) \rangle_t \tag{4.1}$$

The angular brackets $\langle \cdot \rangle_t$ indicate the averaging over time t. Figure 4.1 shows an example of output of a trained ESN based on pretrained word embedding (ESN-WV) for three candidate authors and a sample document. The level of red represents the authorship probability for the true author and higher levels of red indicate a higher authorship probability for the true author of the document. With many-to-many architectures, documents are processed as a stream of text and the outputs are probabilities for each token. These ouputs are normalised to sum to one by a sofmax function and at each time step the RNN output is the estimated authorship probabilities that the text stored in the memory has been written by each candidate author.

For ESNs, its architecture opens the possibility to extract information on how the model processes the inputs and how they influence the outputs. This property makes possible the analysis and the visualisation of the attribution process and could pave the way for neural models whose decisions can be justified or explained. First, we can measure the average change in the output probabilities as a specific token or series of tokens are fed into the ESN (bold and underlined words in Figure 4.1). Secondly, we can measure the average distance from the author hyper-plane in the reservoir state space resulting from these inputs. Thirdly, studies have shown that it is possible to analyse the ESN's memory and that the quantity of information resulting from past inputs can be measured [168]. It is then possible to quantify the impact of past inputs on the outputs and then why and how the author's probabilities change as shown with bold and underlined word in Figure 4.1. Furthermore, with many-to-many models we have the possibility to measure the authorship probability of

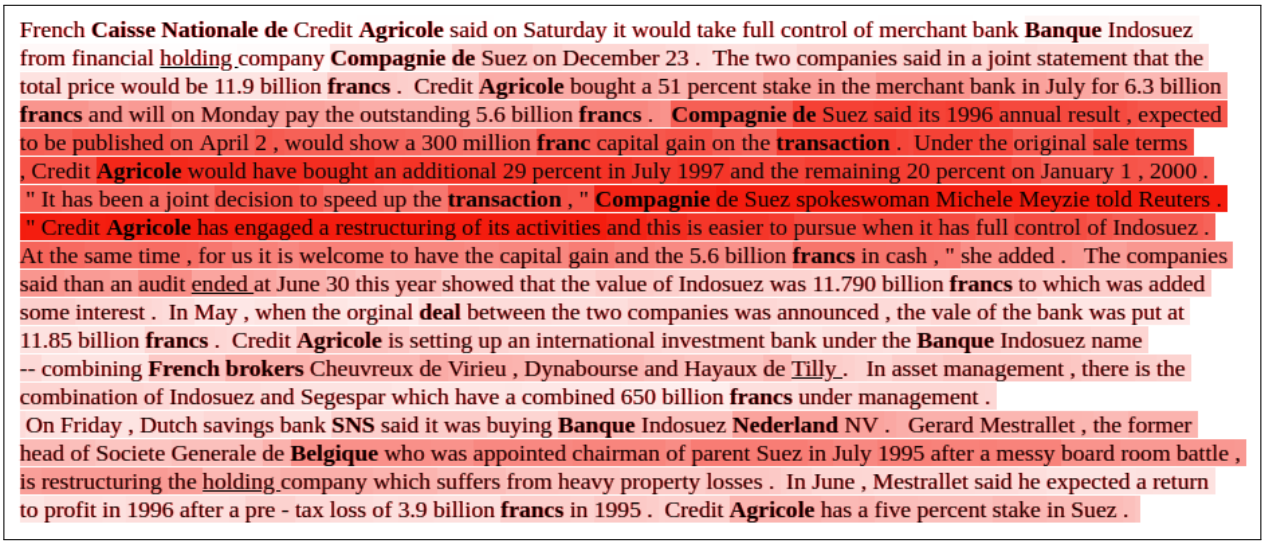


Figure 4.1 – Example of ESN's output on a test sample document. The red level shows the output probabilities for the true author of the document. The bold words represents tokens which resulted in significant increases in the output probability for the true author (two standard deviation). The underlined words represents token which resulted in significant drop in authorship probability for the true author.

any part of a textual document. From document-level to token-level through paragraph and sentence-level, using maximum average probability over time and the same computed outputs. For LSTMs and GRUs, the attention mechanism would allow us to determine which token has influenced the classification at time t. For each token with high output probability, we would retrieve tokens that have a high participation to this classification and save them in a list according to their attention weights. We would then end up with a list of important tokens which are representative of the author' style.

4.3 Echo State Networks

4.3.1 Leak-rate and spectral radius

To analyse the dynamical properties of each textual representation, we run two experiments by evaluating accuracy with leak rate parameter α varying from 0.005 to 1.0, and the spectral radius parameter $\rho(\boldsymbol{W})$ varying from 0.1 to 2.0. For each parameter value, matrices $\boldsymbol{W}$, $\boldsymbol{U}$ and vector $\boldsymbol{b}$ were kept constant. Figure 4.2 shows the 10-fold cross-validation (10-CV) accuracy for the seven features and the various leak rates and spectral radius. Vertical lines at each point also indicate the standard deviation obtained with the 10-fold cross validation.

Leak rate The left side of Figure 4.2 shows accuracy obtained with the different leak rate values. The best results (92.07%, 90.53%, and 89.53%) were obtained respectively for the character encoder (CE), character trigrams (C3) and word embedding (WV) respectively, with leak rates of 0.01, 0.001 and 0.01. These performances have a standard deviation of respectively 1.98, 3.46 and 2.31. The accuracy decreases for CE when dynamics accelerates (leak rate increases) going from 92.07% to 81.47% (-10.6) while standard deviation goes from 2.26 to 5.43 (+3.45). For future research, it would be interesting to know if additional pretrained inputs can make ESN performances more stable over large range of leak rates. Word embedding (WV) is less stable, but stay far above a random classifier (6.67% = 1/15) with an accuracy going from 89.53% to 64.6% (-24.9). The standard deviation slowly increases, going from 1.55 to 3.87 (+1.56). However, the character trigram-based embedding (C3) shows higher instability with various leak rates losing 73.73 points of accuracy going from 90.06% to 16.33%, not so far from a random prediction. The standard deviation goes from 3.46 to 7.29 (+3.83).

Next features in descending order of accuracy are the character bigram-based embedding (C2), function words (FW), Part-Of-Speech (POS) and character-unigram embedding (C1) with respectively an accuracy

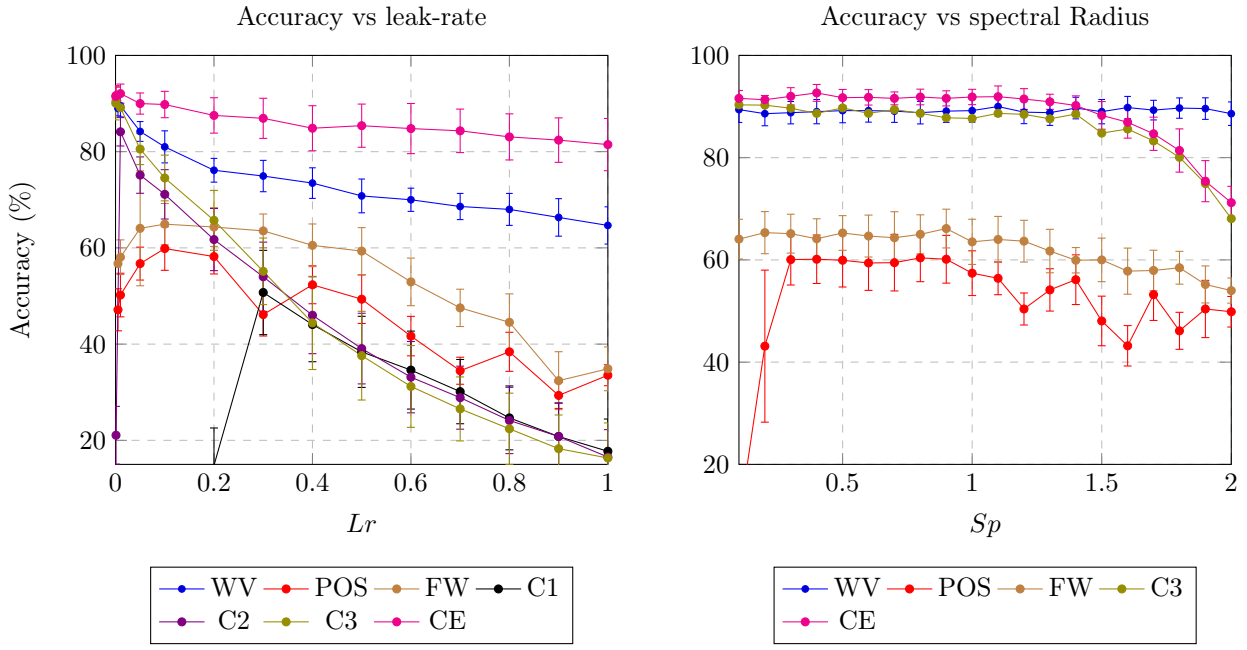


Figure 4.2 – 10-fold cross-validation accuracy of the ESN model ($n_h = 1000$) on 15 authors of the Reuters C50 dataset with varying values for the leaky rate (left) and spectral radius (right) parameters.

of 84.1%, 64.93%, 59.87% and 50.73% with leak rates of 0.01, 0.1, 0.1 and 0.3. They achieved a standard deviation respectively of 2.95, 4.29, 4.53 and 8.76. Regarding their dynamical properties, character bigrams (C2) are the less stable with an accuracy going from 84.1% to 16.5% (-67.6) while the standard deviation went from 2.95 to 5.69 (+2.74). The second less stable feature is C1 with an accuracy going from 50.73% to 17.73% (-33) and a standard deviation going from 8.76 to 6.7 (-2.06). The accuracy and standard deviation of function words (FW) went respectively from 64.93% to 34.86% (-30.1) and from 4.29 to 4.55 (+0.26). For Part-Of-Speech (POS), accuracy and standard deviation went respectively from 59.86% to 33.53% (-26.3) and from 4.53 to 2.18 (-2.18).

Regarding dynamical stability, features in descending order of stability are CE (decrease in absolute value of 10.6), WV (-24.9), POS (-26.3), FW (-30.1), C1 (-33), C2 (-76.6) and C3 (-73.73). The best accuracy is achieved respectively with leak rates 0.01 (CE), 0.01 (WV), 0.1 (POS), 0.1 (FW), 0.3 (C1), 0.01 (C2), 0.001 (C3). All features show a relatively slow dynamics with leak rate below or equal to 0.1, except for C1 which achieved its best accuracy with a leak rate of 0.3. For all features, accuracy decreases with increasing dynamics after their optimal leak rate. For character-based features, we can observe that the optimal leak rate decreases with increasing n-grams going from 0.3 for C1, to 0.001 for C3 through 0.01 for C2. We can also observe that character-based features show less stability across the different dynamics as the three features with largest drops in accuracy are C1, C2 and C3. CE is an exception in this regard as it achieved the highest stability. The impact of feature extraction on ESN stability is an interesting open question for future research. Regarding standard deviation, features in descending order of standard deviation are CE (1.98), WV (2.31), C2 (2.95) C3 (3.46), FW (4.29), POS (4.53) and C1 (8.76). Two of the three top features (CE, WV) have also the lowest standard deviation (1.98 and 2.31 respectively). Across the different dynamics no clear pattern appears, some features showing increasing standard deviation (WV, FW, C2, C3, CE) while others show decreasing standard deviation (POS, C1).

Spectral radius It has been shown in the RC literature that the highest ratio between computational power and memory is reached just below the border of chaos [169]. Formally, it means that ESNs are at their maximum performance with a spectral radius just below 1.0. To evaluate whether this peak in performance before the border of chaos also apply in tasks related to text processing, we designed a second experiment where we tested different spectral radius values for each features using the best leak rates found above (0.01, 0.001, 0.01, 0.1 and 0.1 for respectively CE, C3, WV, POS and FW). The right side of Figure 4.2 shows results of this experiments. We removed two character-based features with lowest accuracy (C2, C1) to save

time.

The best accuracy is reached by the character encoder (CE) with 92.67% for a standard deviation of 1.65 with a spectral radius of 0.4. Accuracy stay stable between 0.1 and 1.2, and slowly decrease above 1.2 going from 91.47% to 71.20% (decrease in absolute value of 20.3). We observe no peak at the border of chaos for this feature but a high sensibility to the chaotic behaviour induces by high spectral radius values. Standard deviation slowly increases after 1.2 going from 2.03 to 4.23, the chaotic dynamic increasing the variability in the performance of this model.

The second best accuracy is achieved by the character tri-grams (C3), this feature has very stable results similar to word embedding (WV) for spectral radius up to 1.4 with a maximum of 90.53% and a standard deviation of 1.41 for a value of 0.1. Above 1.4, the accuracy drops quickly going from 88.53% to 68.08%. The variability in performance also increases after 1.4 going from 3.5 to 5.8.

The third best accuracy is obtained by word vectors (WV) with a spectral radius of 1.1 (90%), but with no significant differences on the whole spectrum of value, and shows no drop and strong stability up to 2.0. We observe no drop in accuracy after 1.0 and no peak in performance at the border of chaos for this feature. The variation in performance stay also stable across the whole spectrum of value staying in a range going from 1.92 to 2.51.

Function words (FW) and Part-Of-Speech (POS) obtained the fourth and the fifth accuracy with respectively 66.1% and 60.4%. The accuracy obtained by FW stay stable up to 0.9 and slowly decreases for higher values, with significant increase in variability. The best accuracy is reached for a spectral value of 0.9 with 66.1% and drop to 54% for 2.0. We observe here a slight increase in accuracy at the border of chaos and a sensibility to the chaotic behaviours induced by high spectral radius values.

For POS, accuracy is very low for spectral radius below 0.3 and stay very stable between 0.3 and 0.9 with no peak at the border of chaos. However, accuracy clearly drops for spectral radius values above 0.9 and become unstable going to 43.2% for 1.6 while the variability stay table across the whole spectral of value. Regarding dynamical stability in this experiment, features in descending order of stability are WV (-1.4), FW (-12.1), POS (14.3), CE (-21.5) and C3 (-22.3).

Best accuracy rate is achieved respectively with spectral radius 1.1 (WV), 0.9 (FW), 0.8 (POS), 0.4 (CE) and 0.1 (C3). Regarding pure accuracy, features in descending order of performance are CE (92.7%), C3 (90.5%), WV (90%), FW (66.1%) and POS (60.4%). This result is coherent with observations made on the first experiment. Regarding the border of chaos well known in the RC community, no clear increases can be observed clearly, however, all features except WV clearly decrease in accuracy for spectral radius above 1.0.

Conclusions We can draw several observations from these experiences. First that the dynamics for this task and this kind of data is very slow. Secondly, that the word vector representation (WV) show an immunity against chaotic behaviours up to a spectral radius of 2.0. A point which is not present with character-based representations. Third, no rise of performance is shown just below the border of chaos (where the spectral radius equals one) but a significant drop for value above this limit. And finally, that POS and FW have higher variability in performance with a standard deviation of respectively 4.7 and 3.8 against 1.41, 1.65 and 1.94 for C3, CE and WV respectively.

4.3.2 Reservoir size and training set

With these two parameters (leak rate and spectral radius) evaluated, we can now evaluate ESN models with different complexity (reservoir size) in order to determine if they are prone to overfitting. Another point to evaluate is the capacity of these models to handle small datasets and then small training sets. As a consequence, we designed two experiments to analyse how accuracy change with reservoir sizes varying from 100 to 1,400, and a dataset size varying from 2 to 100 documents (out of 100 in the original dataset). For each hyper-parameter value, matrices $\boldsymbol{W}$, $\boldsymbol{U}$, $\boldsymbol{b}$ were kept constant if possible. Figure 4.3 shows the 10-fold cross validation accuracy for five features (WV, POS, FW, C3, CE). Vertical lines at each point also indicate the standard deviation with the 10-fold cross validation.

Reservoir size In order to see the influence of increasing the number of units in the reservoir and the possible impact of overfitting, we designed an experiment to evaluate the accuracy of ESNs with various reservoir sizes (n_h) with different features. The left-side of Figure 4.3 shows the 10-fold cross validation accuracy obtained with increasing reservoir sizes from 100 to 1,400 neurons. For small reservoir size ($n_h = 100$), the character encoder (CE) largely outperforms the other features with an accuracy of 87.73% against

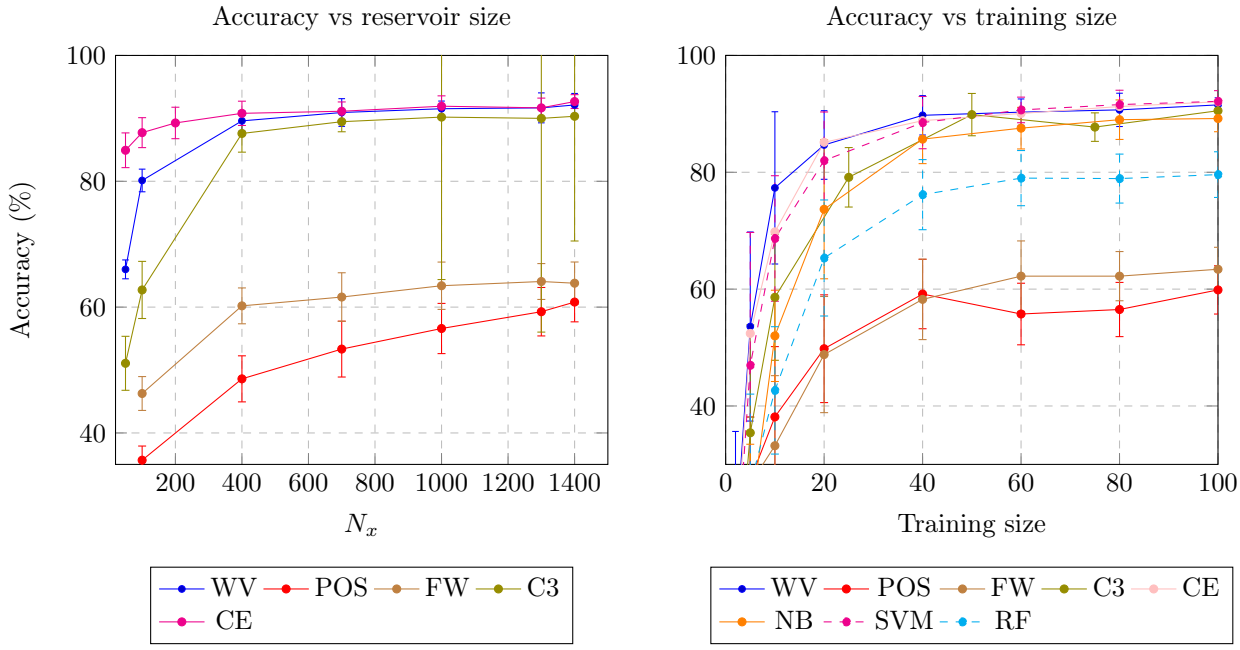


Figure 4.3 – *Left.* 10-fold cross validation accuracy on 15 authors of the C50 Reuters dataset for reservoir sizes (n_h) between 50 and 1,400 on the whole dataset. *Right.* 10-fold cross validation accuracy for different dataset sizes between 2 and 100 documents.

80.1%, 62.7%, 46.3% and 35.7% for WV, C3, FW and POS. This can be explained by the fact that CE has more units on the inputs and then show more computational power independently of the reservoir size. Accuracy with this feature goes from 87.7% for $n_h = 100$ to 92.7% (+4.9) for $n_h = 1400$. Variability in the results slowly decrease going from 2.76 to 1.10, and the best accuracy is achieved with 1,400 units (92.7%).

Word embedding (WV) achieved accuracy going from 80.1% ($n_h = 100$) to 92.1% ($n_h = 1400$) (+12) with a standard deviation staying stable around 1.8. The best accuracy (92.13%) is achieved also with 1,400 units and a standard deviation of 1.83. Character trigrams (C3) has much lower accuracy (62.7%) with small reservoir sizes (100 units) and stable performances up to 700 units. Above 700 units, standard deviation explodes going from 1.6 (700 units) to 34 for 1,300 units. The two other features (FW, POS) obtained much lower accuracy and higher variability. Function words (FW) achieved its best accuracy (64.1%) with 1,300 units and a standard deviation of 2.85. Part-Of-Speech (POS) obtained its best accuracy of 60.8% with 1,400 units and a standard deviation of 3.14. Accuracy obtained with WV and C3 quickly increases up to 400 units and then stay stable around respectively 92% and 90.3% with no sign of overfitting.

Regarding accuracy across the whole spectrum of reservoir sizes, CE achieved the best accuracy (92.7%), followed by WV (92.13%), C3 (90.3%), FW (64.1%) and POS (60.8%). Unlike the two preceding experiments, WV arrives in second position and C3 at the third. Best results are achieved with $n_h = 1400$ except FW which use $n_h = 1300$. Regarding the variability across the whole spectrum of value, CE also achieved the lowest standard deviation (1.10), followed by WV (1.83), FW (2.85), POS (3.14) and C3 (19.82). Based on this experiment, we can observe the poor performances of the character trigrams with small reservoir sizes and its high variability for reservoir size above 700 units. It is reasonable to make the assumption that this feature needs more memory as longer sequences of characters are needed to separate classes. The high instability of C3 above 700 units stay an open question. For reservoir size above 500 units, the results for WV and CE stay stable around 91%, and the C3 remains significantly lower at 90%.

Training size Our experiments show that ESN models have an impressive capacity for fast learning from a small dataset. Right side of Figure 4.3 shows the accuracy for WV, POS, FW, C3, CE and for baseline models (Naive Bayes (NB), Support Vector Machines (SVM) and Random Forest (RF)) for a dataset size varying from 2 files per author to the whole dataset composed of 100 files per author. For each evaluation, we proceeded by doing 10-fold cross validation on a subset of the dataset of the desired size, and this evaluation was done for each possible non-overlapping subsets. The final evaluation is the average accuracy.

Model	100 documents	10 documents	Difference
Linear SVM + Word 1-2 grams	**92.3%**	68.6%	23.7%
ESN-CE	91.8%	69.8%	22.0%
ESN-WV	91.7%	**77.3%**	**14.4%**
ESN-C3	91.6%	58.6%	33.0%
LSTM-WV pretrained	84.5%	50.2%	34.3%
ESN-C2	84.1%	58.6%	25.5%
Random Forest + BOW	80.7%	42.7%	38.0%
ESN-FW	64.2%	33.2%	31.0%
ESN-POS	60.8%	38.1%	22.7%

Table 4.1 – *Second column.* 10 fold cross-validation accuracy averaged for 20 ESN models ($n_h = 1000$) on 15 authors of the Reuters C50 dataset for each features with LSTM (word-based, pretrained, 300 units) and baseline models. *Third column.* 10-fold cross validation accuracy for ESN, LSTM (word-based, pretrained, 300 units) and baseline models with a dataset composed of 10 texts per author. *Fourth column.* Differences between accuracy obtained on the whole data set ($n_d = 100$) and the smaller data set ($n_d = 10$).

For a dataset composed of more than 20 files, CE and WV get the best results with 92.1% and 91.5% respectively for 100 files and stay close for other dataset size above 20. The SVM model stay also close to these two models for dataset sizes above 20 and significantly lower with 20 files (82% against 84.67% and 85.13% for WV and CE respectively). The feature encoder (CE) gets results very close the SVM classifier with however a higher accuracy with 20 files (82% against 85.1%). For a dataset composed of 5 to 20 files of the original dataset, word embedding (WV) outperforms the other approaches, especially with a dataset composed of 10 files with an accuracy of 77.3% against 69.80% for CE, the second best feature for 10 files. Part-Of-Speech (POS) and function words (FW) achieved significantly lower accuracy than the other features staying around 60% with more than 20 files, and around 50% for 20 files while other models stay above 60%. The Naive Bayes baseline (NB) stays close to C3 and significantly lower than CE, WV and SVM.

Regarding variability, the standard deviation is much higher with small datasets and quickly decreases with more training documents. For a dataset of 2 documents, models obtained a standard deviation of 19.2, 7.3, 13.03, 17.3 and 12.2 for respectively WV, FW, C3, SVM and RF. These models obtained for a dataset of 10 files a standard deviation of 13, 12, 10.8, 7.8, 10.7 and 10.9 respectively, and 3.4, 6.9, 3.6, 11.9, 8.3 and 9.9 for a dataset of 40 files. Standard deviation decreases for all models and achieved respectively 1.2, 3.8, 1.6, 2.7, 1.8 and 3.9 on the whole dataset.

4.4 Features and dataset size

Using the best parameter values found in preceding section, we wanted to know which feature can achieve the best accuracy on average on two different dataset sizes. As a first experiment, we randomly generated 20 reservoirs with 1,000 units ($n_h = 1000$) for each feature and evaluated their performances on the whole dataset ($n_d = 100$). As a second experiment, we used the same 20 reservoirs and evaluated their accuracy on a dataset composed of 10 documents ($n_d = 10$). We then used these evaluations to seek statistically significant differences between features and between baseline models (two sided t-tests). The set of baseline models is composed of SVM, Naive Bayes and Random Forest and were implemented with the Scikit-learn framework. Figure 4.4 shows the result of these two experiences.

Features The left side of Figure 4.4 and the second column of Table 4.1 show the average 10-fold cross validation accuracy for each features, baseline models and a LSTM model based on pretrained word-embedding (LSTM-WV-P). Bullets (•) show statistically significant differences with the word-based ESN model and circles (○) indicate statistically significant differences with the LSTM model. The number of bullets and circles show the significant level used for the t-test (• = 5%, • = 1% and • = 0.1%). Best average accuracy is

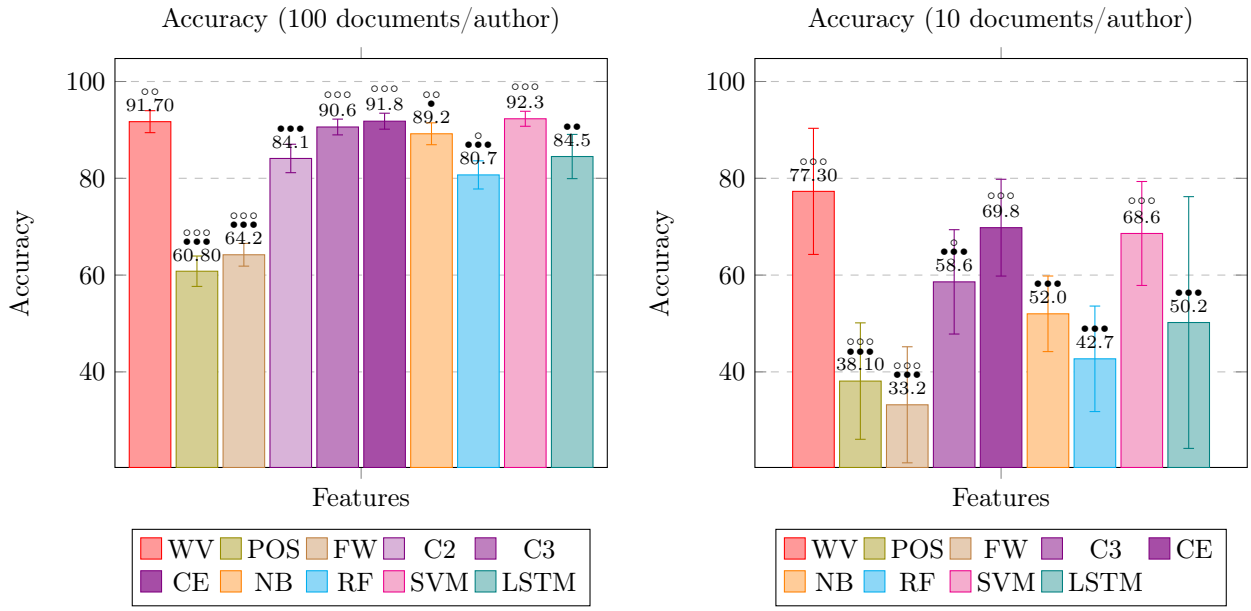


Figure 4.4 – *Left.* 10-fold cross validation accuracy averaged for 20 ESN models ($n_h = 1000$) on 15 authors of the Reuters C50 dataset for each features with LSTM (word-based, pretrained, 300 units) and baseline models. *Right.* 10-fold cross validation accuracy for ESNs, LSTM (word-based, pretrained, 300 units) and baseline models with a dataset composed of 10 texts per author ($n_d = 10$). Bullets (•) represent statistical differences with the ESN-WV models, while circles (○) represent statistical differences with the LSTM model (• = 5%, •• = 1%, •• = 0.1%)

obtained by the word bigrams-based SVM baseline model, character encoder (CE), word embedding (WV) and character trigrams (C3) with 92.3%, 91.8%, 91.70% and 90.6% respectively. However, Figure 4.4 shows that SVM, CE and C3 obtained accuracy not significantly different from WV. Furthermore, character trigram (C3) obtained results significantly (t-test, $\alpha = 0.05$) lower than the SVM baseline and the character encoder (CE). Regarding variability, these features obtained respectively a standard deviation of 1.55, 1.65, 2.27 and 1.63.

The fifth, sixth, seventh and eighth best results are obtained by the Naive Bayes classifier (NB), the word-based LSTM, character bigrams (C2) and Random Forest (RF) with respectively 89.2%, 84.5%, 84.1% and 80.7% and a standard deviation of 2.26, 4.57, 2.94 and 2.91. All these features show significant differences with the word-based ESN and with the other state-of-the-art models (SVM, CE and C3). Furthermore, there is no significant difference between word-based LSTM and character bigrams (C2) while other members of this set obtained significant differences in accuracy. The next best results were obtained by Part-Of-Speech (POS) and function words (FW) with an accuracy of 64.2% and 60.80% and a standard deviation of 3.14 and 2.35 respectively. These two features obtained an accuracy significantly lower than those obtained by the other features evaluated and composed the bottom of the pack of the features evaluated in this experiment.

Once these features have been evaluated, we need to compare all of them with a set of baseline models composed of classical methods used in authorship attribution. The four baseline models we selected obtained (in descending order) respectively 92.3%, 89.2% and 80.7% for SVM, Naive Bayes and Random Forest and a standard deviation of 1.55, 2.26 and 2.91. In comparison, the best baseline model (SVM) obtained an accuracy significantly higher than the character trigrams (ESN-C3) but not significantly higher than ESN-WV and ESN-CE. These three models (SVM, ESN-WV, ESN-CE) are the front runners of our evaluation with no significant differences between them. The second baseline model, the Naive Bayes classifier, obtained an accuracy of 89.2% which is significantly lower than the front runners and ESN-C3 but significantly higher than the LSTM model. This model is therefore situated between the leading peloton and the LSTM model. The third baseline model, the Random Forest (RF), obtained an accuracy of 80.7% and a standard deviation of 2.91. A result significantly lower than the front runners, the ESN-C3 and LSTM model, but also significantly higher than the bottom of the pack (ESN-POS, ESN-FW).

To compare shallow and deep RNN models, we selected the best LSTM model we obtained for this task.

The resulting model is based on the same pretrained word vectors (LSTM-WV-P) as the ESN model and with a dropout layer before the output layer ($rate = 0.5$). This model was implemented with Keras and trained with the Adam gradient descent algorithm for 250 epochs with an hidden layer of size 300. It was trained with same procedure, training data and on the same labels as ESNs. This LSTM model obtained an accuracy significantly lower than ESN-CE, ESN-WV, ESN-C3. However, we found no statistical differences with ESN-C2, and a statistically significant difference with ESN-POS and ESN-FW in favour of the LSTM model. Considering variability, LSTM obtained a standard deviation of 4.6 against 1.97, 2.3 and 1.6 respectively for ESN-CE, ESN-WV and ESN-C3. Our experiment shows that ESN models based on word vectors (WV) and character trigrams (C3) not only outperform deeper models such as LSTM, but also that they have more stable performances.

Dataset size Authorship attribution is applied to a large diversity of corpora, from very short messages (tweets) to datasets composed of long texts such as novels. As shown in [125], dataset and document size are very important parameters in authorship attribution. We therefore decided to evaluate the robustness to a change in data size of ESNs, LSTMs and baseline models by testing these models on two different dataset sizes. The first is composed of the whole dataset (100 documents per author, $n_d = 100$) while the second has only 10 documents ($n_d = 10$) per author. The left and right sides of Figure 4.4 and the third column of Table 4.1 show the average accuracy for $n_d = 100$ and $n_d = 10$ respectively. To evaluate these models on the smaller dataset, we performed 10-CV on non-overlapping dataset composed of 10 documents extracted from the original dataset and computed the average accuracy over the different evaluation. We then used statistical tests to find significant differences.

Unlike evaluations done on the whole dataset, the word-based ESN (ESN-WV) obtained the best accuracy followed with ESN-CE and SVM with respectively 77.3%, 69.8% and 68.6%. However, we were unable to find statistically significant differences between these models on the $n_d = 10$ dataset. These three models composed the front runners of our evaluation. The fourth, fifth, sixth and seventh best accuracy were obtained by ESN-C3, Naive Bayes (NB) classifier, LSTM and Random Forest (RF) models with respectively 58.6%, 52%, 50.2% and 42.7%. These three models obtained an accuracy significantly lower than the SVM, ESN-CE and ESN-WV models, however, we were unable to find statistical differences between NB, LSTM and RF models. The last two models are the ESN-POS and ESN-FW with respectively 38.1% and 33.2%. They obtained an accuracy significantly lower than the other models, but we found no significant difference between them.

Results obtained by LSTMs are disappointing as they are the state-of-the-art in a lot of NLP tasks. Two hypotheses are possible to explain this fact. First the small size of available textual data for each class compared to other NLP application which benefits shallower architectures. Secondly, the possibility that time-related dependencies are not necessary or not long enough in LSTM. Indeed, the ESN's state is the result of the accumulation of a hundred words as the best leak rate we found is 0.01 for this task. These models are then able to create a context vector summarising large parts of the investigated documents. Different LSTM hyper-parameters have been tested without improvements.

In comparison to the results obtained on the whole dataset, unsurprisingly all features showed a significant drop when trained on the smaller dataset. The Random Forest, the naive Bayes classifiers and the LSTM were the most affected by the reduction of training data with respective drops of 37.2, 38 and 34.3 points. The least affected were the ESNs based on word-vectors (ESN-WV) and the character encoder (ESN-CE) features with a drop of respectively 14.4 and 22.0 points. Between features based on words (ESN-WV) and characters (ESN-C3), the character-based feature was more affected by smaller training size with a respective drop of 32 for C3 against 14.4 for WV.

Regarding variability, the standard deviation significantly increases with smaller dataset. For the front runners (ESN-WV, ESN-CE and SVM), standard deviation goes from 2.27, 1.97 and 1.55 on the whole dataset to 13, 10 and 10.7 on the smaller dataset. For the middle-class models (ESN-C3, NB, LSTM and RF), standard deviation goes from 1.63, 2.26 and 4.57 on the whole dataset to 10.8, 7.8, 26 and 10.9 for the smaller dataset. For the bottom of the pack (ESN-POS and ESN-FW), standard deviation goes from 3.1 and 2.4 on the whole dataset to 12 and 12 on the smaller dataset respectively. There is no clear differences in variability between models on the two datasets.

	n_q	n_e	$\|V\|$	α	**ESN-10CV**	**Std. dev**
C1	2	10	4,635	0.3	50.7%	8.76
C2	2	30	32,908	0.01	84.1%	2.94
C3	2	60	167,026	0.001	90.7%	1.63

Table 4.2 – Summary of each character embedding pre-trained on Wikipedia. The context n_q is the number of n-grams on each side used for prediction. The last two columns show the final 10CV accuracy and the corresponding standard deviation.

4.4.1 Lexical and syntactic features

We want now to identify which models between word-based (ESN-WV) and character-based ESNs (ESN-C) are the best to predict the author of a text. The left side of figure 4.4 shows the results for each feature we tested with ESNs. We also performed the same test (20 randomly generated reservoirs, average accuracy) with character unigram-based ESN and performed the same statistical tests (t-test, $\alpha = 0.05$). On one hand, these results show that the word vector-based feature (WV) got an average accuracy of 91.7%. On the other hand, our three character-based vectors respectively got an accuracy of 50.7%, 84.1% and 90.6% for C1, C2, and C3. These results showed that the word vector-based feature (WV) got an accuracy significantly higher than representations based on embedding of unigram and bigram of character (C1, C2).

Secondly, the results obtained with the embedding of character trigrams (ESN-C3) were much closer to word-vectors (ESN-WV) reaching an average accuracy of 90.7% with no significant difference with word-vectors (WV). Our results demonstrated that word embedding produces no significantly higher accuracy rate than character trigrams-based features. The next question is to establish whether an increase in character grams, for example from character bigrams to trigrams, could improve the average accuracy. Table 4.2 shows the average accuracy of each character-based embedding (C1, C2, and C3). The pretrained vectors based on unigrams and bigrams of character reached an accuracy of 50.7% and 84.1% respectively with a significant improvement of 33.4 points. Secondly, in comparison to bigrams, trigrams significantly improved the average accuracy by 6.5 points going from 84.1% to 90.7%. Our results show that increasing the number of consecutive characters used in character-based features significantly improved the accuracy.

Recently, neural networks based on deep architectures such as Convolutional Neural Networks (CNN) and LSTM were very effective in fields such as speech recognition and computer vision. Therefore, the next question we want to answer is: *is it possible to improve ESN models with additional pretrained input layers?* More specifically, we wanted to test whether adding pretrained layers as ESN inputs could improve its performance in this task.

To this end, we compared the average accuracy obtained by ESN based on deep feature encoder (CE) with the same ESN based on classical shallow features. The left side of Figure 4.4 shows that CE obtained an average accuracy of 91.8% compared to 50.7%, 84.1% and 90.6% for the other character-based features. Adding deep feature encoder as ESN inputs significantly improved the average accuracy compared to C1, C2, and C3. Nevertheless, compared to word-based representation (WV), character encoder (CE) has not significantly improved the average accuracy. Moreover, it also increased the training time as few minutes are necessary to train word-based ESN compared to several hours for ESN with the deep character encoder (CE).

Moreover, our results show that adding layers in the ESN's inputs does not significantly improved accuracy compared to word-based ESN, but significantly improved performances compared to other character-based features. Nonetheless, this conclusion should not be apply to other tasks and datasets, as other architectures and loss functions could be tested in addition to the different statistical properties between datasets.

As syntactic features (Part-Of-Speech) and function words (FW) are commonly used in authorship attribution for they capacity to represent unconscious markers of an author's style, it is of a particular interest to compare lexical and content-based features with our ESN models. ESN-FW and ESN-POS respectively reached an average accuracy of 64.2% and 60.8%. These results are significantly lower than content-based features such as word embedding (ESN-WV) and character-based representations (ESN-C1, ESN-C2, ESN-C3, ESN-CE). There is also no significant difference between these two representations.

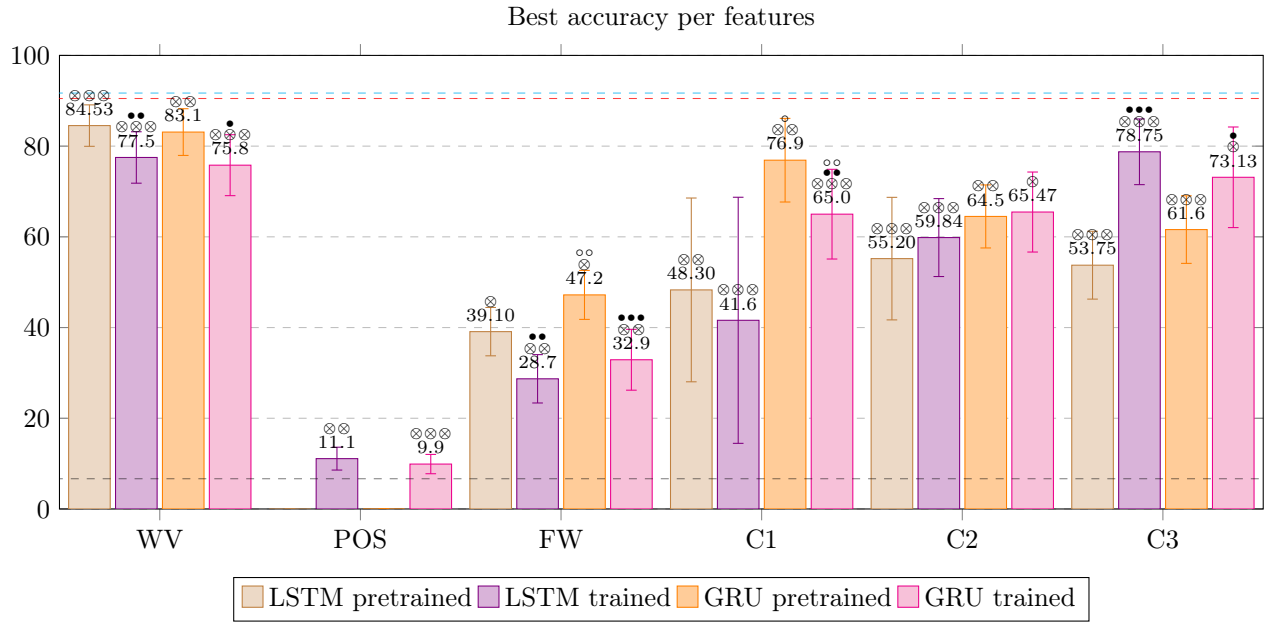


Figure 4.5 – 10-fold cross-validation accuracy for LSTM and GRU with pre-trained and trained features on 15 authors of the Reuters C50 data set. Bullets represent statistical differences between pretrained and trained version for same model and feature ($\bullet = 5\%$, $\bullet\bullet = 1\%$, $\bullet\bullet\bullet = 0.1\%$). Cicles ($\circ$) represent statistical differences between LSTMs and GRUs for the same feature ($\circ = 5\%$, $\circ\circ = 1\%$, $\circ\circ\circ = 0.1\%$). Circle-cross ($\otimes$) represent the size of the cell of the corresponding model ($\otimes = n_h = 100$, $\otimes = n_h = 200$, $\otimes = n_h = 300$). Blue and red lines represent word-based and character-based ESN.

4.5 LSTMs and GRUs

In this section, we explore the performances obtained with two RNN models based on the deep learning paradigm, the Long Short-Term Memory (LSTM) and the Gated Recurrent Units (GRU). The first is introduced in 3.2.8 and the second in 3.2.10. We start by exploring in 4.5.1 results obtained with pretrained and trained versions of each features and put them in comparison with results obtained with ESNs in the preceding section. We then analyse in 4.5.2 the accuracy achieved by LSTMs and GRUs with different cell sizes as the memory capacity and the computational power of these networks are dependent on this parameter.

4.5.1 Pretrained and trained features

As shown in Section 3.3.1, textual data can be represented by one-hot vectors where each dimension corresponds to a token, or by embedding these tokens in a space of lower dimension where tokens with similar contexts end up near each other. This solution increase the number of parameters and then the training time, and increase the size of the dataset necessary to train the model. However, training token embedding directly on the task can be effective if the dataset has a very different vocabulary than the one used to train the embedding layer. We then wanted to determine if training words embedding is effective on this authorship attribution task or if it increases model's sensitivity to overfitting.

To answer this question, we designed another experiment where we trained LSTM and GRU models with trainable or pretrained embedding layer on a set of different features composed of words (WV), Part-Of-Speech (POS), function words (FW), character unigrams, bigrams and trigrams (C1, C2, C3). We then used statistical tests to find significant differences between LSTMs and GRUs using trainable and pretrained embedding layers. Figure 4.5 shows the result of this experiment.

Word embedding The best result is obtained with an LSTM based on pretrained word embedding (LSTM-WV-P) with an accuracy of 84.5% compared to 77.5% (-7) for the same model with a trained version of this feature (LSTM-WV-T). This difference is significant. The GRU model with the same feature (GRU-WV-P)

obtained the second best result with an accuracy of 83.1% (-1.4) while the same trained GRU (GRU-WV-T) obtained an accuracy of 75.8% (-7.3 against pretrained GRU). The difference between trained and pretrained GRU is significant, while differences between LSTM and GRU models are not. The pretrained GRU model uses a cell size of only 200 units compare to the LSTM model which uses 300 units. Our results show that training word embedding with LSTMs and GRUs significantly degraded their performances.

Character embedding The next best result is obtained with an LSTM based on trained vectors of character trigrams (LSTM-C3-T) with an accuracy of 78.8% against only 53.8% for the same pretrained version (LSTM-C3-P) of this model (significant difference). The same phenomenon can be observed with GRUs as the trained version (GRU-C3-T) obtained 73.1% against only 61.6% for the pretrained (GRU-C3-P) version (significant difference). Unlike WV, character trigrams (C3) are more effective if trained directly on the task at hand, there is maybe too much differences between the vocabulary used to pretrained these vectors and the one found on this task. As with WV, there is no significant differences between LSTMs and GRUs and variability is higher with C3 compared to WV (4.57, 5.67, 5.15 and 6.72 for WV against 7.48, 7.24, 7.45 and 11.09 for C3). However, the trained GRU model achieved its accuracy with only 100 units.

The next interesting result is obtained by a GRU based on pretrained embedding of character unigrams (GRU-C1-P) with an accuracy of 76.9% against 65% (-11.9%) for the same model with a trained (GRU-C1-T) embedding layer (significant difference). Unlike WV and C3, GRU models obtained significantly higher accuracy than LSTM with the same number of units. The LSTM model obtained respectively 48.3% and 41.6% with the pretrained and trained version of the feature (no significant differences) and a very high variability (20.3 and 27.1 respectively) in comparison to GRUs. This result is pretty surprising as the GRU model based on the pretrained version of this feature obtained a very high accuracy on comparison to character bigrams for example (76.9% against 65.47% respectively). For both LSTMs and GRUs, pretraining the embedding layer achieved an higher accuracy but the difference is significant only for GRUs. The reason behind the much higher accuracy of GRU models compared to LSTM is an open question but we can propose the hypothesis that on this feature the higher number of parameters of LSTM models make them more sensitive to overfitting.

Regarding character bigrams (C2), the best accuracy obtained with models using this feature is achieved by a GRU with the trained version of this feature (GRU-C2-T) with 65.5% against 64.5% (-1%) for the pretrained version (GRU-C2-P). The LSTM models achieved respectively 55.2% and 59.8% for the pretrained (LSTM-C2-P) and trained version (LSTM-C2-T). However, it must be noted that we were unable to find any statistically significant differences between these models. In consequence, no clear conclusions can be done out of this experiment on this feature.

Function words and Part-Of-Speech Concerning function words (FW), the best accuracy is obtained by the GRU model based on pretrained embedding layer (GRU-FW-P) with 47.2% against 32.9% for the trained (GRU-FW-T) version (significant difference). The same observation can be made for LSTMs as the pretrained version (LSTM-FW-P) obtained 39.10% against 28.7% for the trained (LSTM-FW-T) version (significant difference). GRU models achieved a better accuracy for both versions of the feature but we were able to find a significant difference only for the pretrained version. Finally, Part-Of-Speech (POS) has only a trained version of this feature as we did not have pretrained embedding available. However, LSTMs (LSTM-POS-T) and GRUs (GRU-POS-T) respectively obtained 11.1% and 9.9% with no significant difference between these two models.

Features The next question is to know which features obtained significantly higher or lower accuracy and to determine if we obtained the same pattern than with ESN models. The best model for each feature obtained 84.5%, 11.1%, 47.2%, 76.9%, 65.5% and 78.5% for respectively WV, POS, FW, C1, C2 and C3. The two highest accuracy were obtained by pretrained word embedding (LSTM-WV-P) with 84.5% and trained character trigrams (LSTM-C3-P) with 78.5%. We were unable to find any statistically significant difference between these two models. The next best feature is the pretrained character unigrams (GRU-C1-P) with 76.9%. We found significant differences with word representations (WV) but not with character trigrams (C3). These observations allow us to categorise these three features as the front runners of this experiment in the following order: WV, C3 and C1. The bottom of the pack is constituted of character bigrams (GRU-C2-T), function words (GRU-FW-P) and Part-Of-Speech (LSTM-POS-T) with an accuracy of 65.47%, 47.2% and 11.1% respectively. We found statistically significant differences between these three features and with

the three front runners specified above. In descending order of accuracy, WV become first, then C3 and C1, followed by C2, FW and POS.

In comparison to the order we found above with ESN models (WV, C3, C2, FW, POS, C1), C1 obtained much higher accuracy with LSTM/GRU models than with ESN coming in second position after C3. However, LSTMs and GRUs can train their embedding matrix while ESNs can only use pretrained representations. If you compare accuracy obtained with ESN to the results obtained with LSTM and GRU but only with pretrained features, the new order of features by accuracy is WV, C1, C2, C3, FW and POS. Results obtained with LSTMs and GRUs based on pretrained features show that character-based features are ordered in reverse order compared to character-based ESNs. C1 being the most effective feature with 76.9% against 64.5% and 61.6% respectively for C2 and C3. We found no significant difference between this two pretrained models (C2 and C3), but a clear difference between these two and pretrained character unigrams (C1). Why deep models are more effective on character unigrams while ESNs are better on bigrams, trigrams or even more, is an open question. It could be related to the capacity of ESN to model longer dependencies while GRUs and LSTMs are constrained by the vanishing of the gradient after a few dozen time steps. ESNs achieved a higher accuracy with all features except for C1 (50.1% for ESNs against 76.9% for GRUs).

4.5.2 Cell sizes

In Equation 3.19 (Section 3.2.8), $\boldsymbol{c}$ represents the memory cell used by the LSTM to keep the necessary information for the task at hand. For GRU, the hidden vector $\boldsymbol{h}$ keep these information (Equation 3.31 in Section 3.2.10). The size of these two vectors is essential as it specifies the sizes of the gates and then the complexity of the model. With cell/output too small, the model shows high bias and is not able to keep enough information for the task at hand. With cell/outputs too big, the model will overfit quickly due to high variance. In this subsection, we explore the performance of LSTMs and GRUs with different cell/hidden sizes ($n_h = 100$, $n_h = 200$ and $n_h = 300$) to evaluate how the complexity of the models influence the accuracy. One way to prevent overfitting with deep neural networks is to add dropout layers which put some inputs coming from lower layers to zero with a specific probability. Here we added a dropout layer before the output with a probability of 50%. We will then be able to determine if this strategy is effective on this task and on small datasets such as ones found in authorship attribution.

To answer these questions, we made an experiment where we trained LSTM and GRU models with different cell/hidden sizes and performed statistical tests (t-tests) in order to find significant differences. Results of this experiment are presented in Figures 4.6 and 4.7. These figures show the accuracy per cell/hidden sizes for each feature for LSTMs and GRUs respectively. Figure 4.6 shows the accuracy for the different pretrained features and the different cell/hidden sizes. Figure 4.7 show the accuracy for the different trained features and the different/hidden cell sizes. Bottom and top plots show respectively the accuracy obtained with LSTMs and GRUs. Bullets ($\bullet$) represent statistical differences with the $n_h = 100$ model for the same feature (type, trained/pretrained). Circles ($\circ$) represent statistical differences with the $n_h = 200$ model for the same feature (type, trained/pretrained). One, two and three of these symbols represents a statistical tests with respectively $\alpha = 5\%$, $\alpha = 1\%$ and $\alpha = 0.1\%$. A cross ($\times$) indicates that we were not able to find statistically significant difference between the corresponding model and a random classifier.

Word embedding Pretrained word embedding (LSTM-WV-P) obtained an accuracy of 82.6%, 83.1% and 84.5% respectively for each cell/hidden sizes for the LSTM model, and 75.4%, 83.1% and 72.2% for the GRU model (GRU-WV-P). No significant differences were found between the different cell sizes for LSTM, while the GRU model with 200 units obtained an accuracy significantly higher than its versions with 100 and 300 units. For trained word embedding (LSTM-WV-T), LSTM models obtained an accuracy of 74.5%, 72.5% and 77.5% while GRU models (GRU-WV-T) achieved an accuracy of 70.9%, 70.2% and 75.8%. It might seem that there is an improvement in accuracy for LSTM and GRU with 300 units, but no significant differences were found for these two models for different cell/hidden sizes. Therefore, only GRU shows improvement in accuracy with a specific size ($n_h = 200$) for pretrained word embedding and no drops in accuracy can be observe for other models.

Character embedding Pretrained character trigrams obtained an accuracy of 42.5%, 52.2% and 53.8% respectively for each cell/hidden sizes for LSTM models (LSTM-C3-P), and 56.6%, 58.8% and 61.6% for GRU models (GRU-C3-P). No significant differences were found for GRU models but the LSTM with 300 units achieved an accuracy significantly higher than the 100 units version. In this case, increasing LSTM's cell

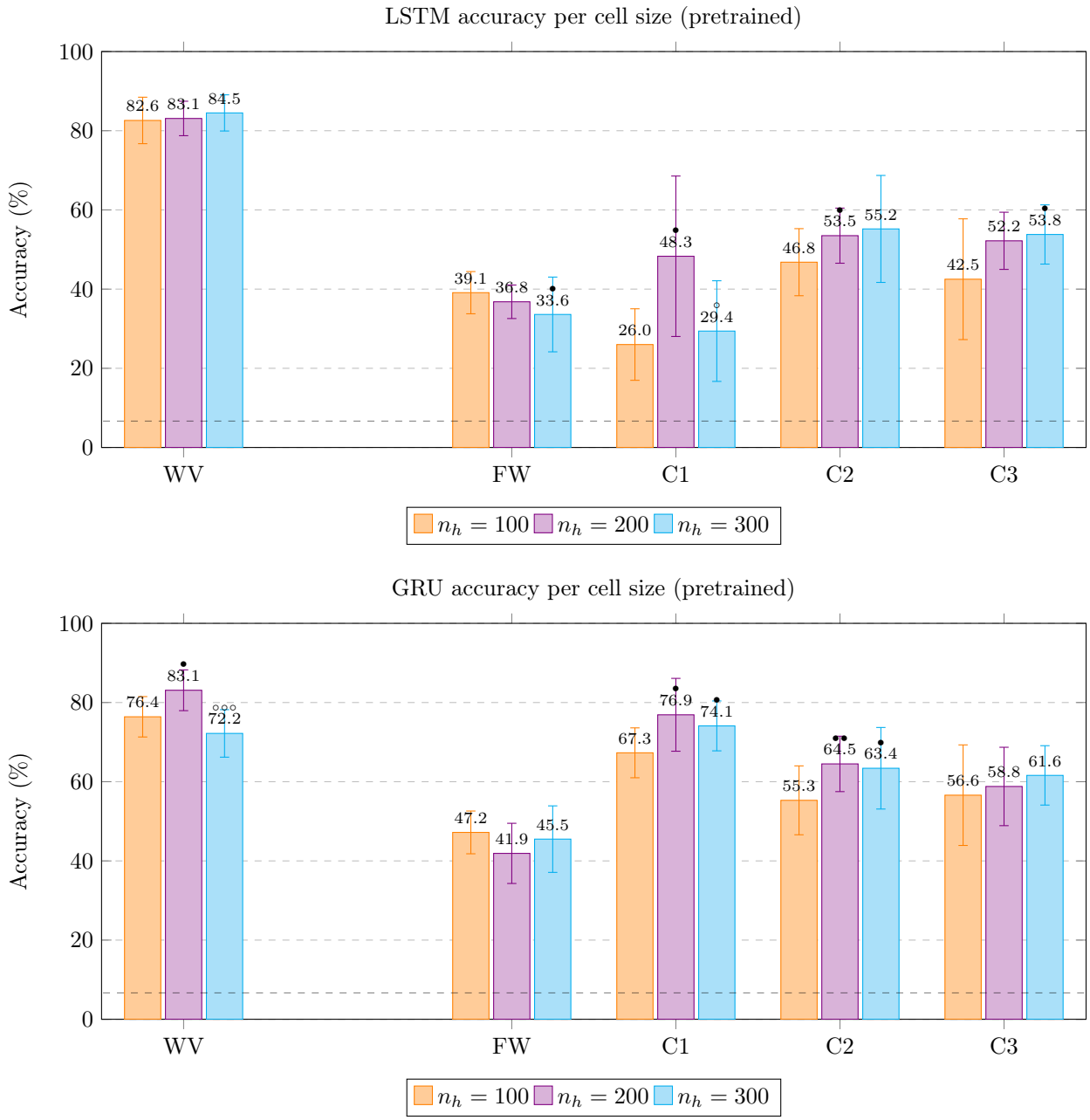


Figure 4.6 – 10-fold cross validation accuracy for LSTM (top) and GRU (bottom) with pretrained features and different cell sizes on 15 authors of the Reuters C50 dataset. Bullets (•) represent the statistical differences with $n_h = 100$ models for each features (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent the statistical differences with $n_h = 200$ models for each features (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%).

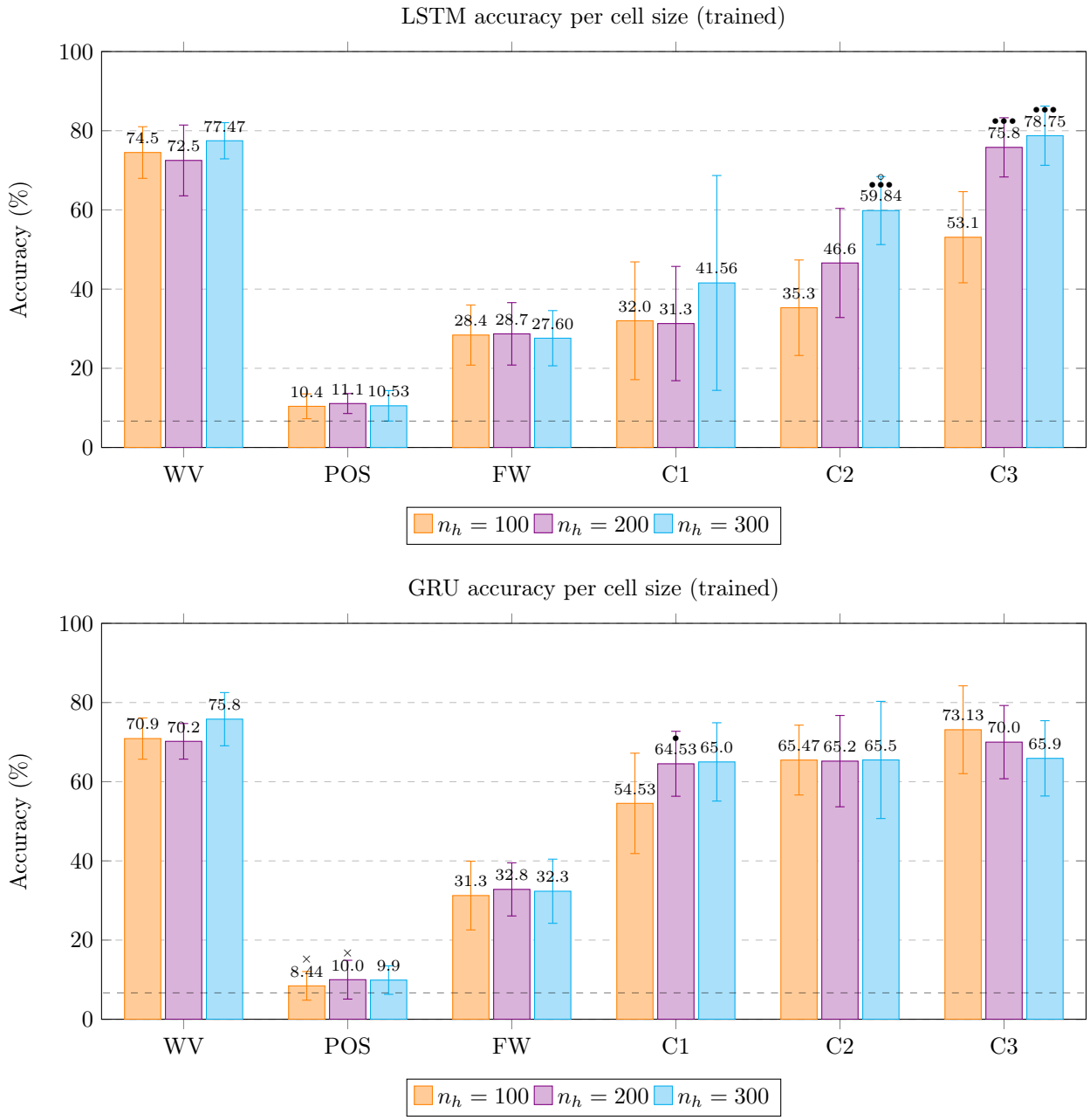


Figure 4.7 – 10-fold cross validation accuracy for LSTM (top) and GRU (bottom) with trained features and different cell sizes on 15 authors of the Reuters C50 dataset. Bullets (•) represent the statistical differences with $n_h = 100$ models for each features (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent the statistical differences with $n_h = 200$ models for each features (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%).

sizes improved accuracy. For trained character trigrams, LSTM models (LSTM-C3-T) obtained an accuracy of 53.1%, 75.8% and 78.8% while GRU models (GRU-C3-T) achieved an accuracy of 73.13%, 70.0% and 65.9%. There is a clear improvement for LSTM with cell sizes above 100 units and a significant difference between LSTM with 200 or 300 units and LSTM with 100 units. For GRU, it might seem that there is a drop in accuracy for GRU with increasing hidden sizes but we were unable to find any statistically significant differences between these three models. We can observe that for this feature we found significant improvements with increasing cell/hidden sizes only for LSTMs.

For pretrained character bigrams, LSTM (LSTM-C2-P) models achieved an accuracy of 46.8%, 53.5% and 55.2% while GRU (GRU-C2-P) models obtained an accuracy of 55.3%, 64.5% and 63.4%. For both LSTMs and GRUs, we found a significant improvement with bigger cell/hidden sizes, more specifically for GRUs with 200 and 300 units. No drop in accuracy can be found with increasing cell/hidden sizes. For trained character bigrams, LSTMs (LSTM-C2-T) obtained an accuracy of 35.3%, 46.6% and 59.8% while GRUs(GRU-C2-T) achieved an accuracy of 65.5%, 65.2% and 65.5%. A clear and significant difference can be observed for LSTM with 300 units compared to LSTMs with smaller cell size ($n_h = 100$), while GRUs show no sign of significant improvement with increasing hidden sizes. For this feature, all models except GRUs based on trained features show significant improvement with increasing cell/hidden sizes.

Pretrained character unigrams obtained an accuracy of 26.0%, 48.3% and 29.4% respectively for each cell sizes for LSTM models (LSTM-C1-P), and 67.3%, 76.9% and 74.1% for GRU models (GRU-C1-P). We found significant improvement for the LSTM with 200 units and a drop for LSTMs with more units (300). We also found significant difference for GRUs with more than 100 hidden units. For trained character unigrams, LSTMs (LSTM-C1-T) obtained an accuracy of 32%, 31.3% and 41.6% while GRUs (GRU-C1-T) achieved an accuracy of 54.5%, 64.5% and 65%. Only GRUs show sign of significant improvement with increasing number of hidden units. For this feature, we can observe a more complex pattern of improvements and drops. However, outputs of our experiment on this feature show a high variability, and we therefore think that more evaluation should be done here. More specifically for LSTM based on pretrained features (LSTM-C1-P) with 200 units showing an important improvement but also a drop for LSTM with 300 units while having a high variability (20.3 for 200 units, 12.7 for 300 units). The other models using character unigrams show signs of improvement with increasing cell/hidden sizes.

Part-Of-Speech and Function Words For pretrained function words, LSTM (LSTM-FW-P) models achieved an accuracy of 39.1%, 36.8% and 33.6% while GRU (GRU-FW-P) models obtained an accuracy of 47.2%, 41.9% and 45.5%. We found significant difference only for LSTM with 300 units compared to the same model with 100 units. As accuracy is lower for the 300 units model, it seems that LSTM based on this feature faces overfitting. For trained function words, LSTMs (LSTM-FW-T) obtained an accuracy of 28.4%, 28.7% and 27.6% while GRUs (GRU-FW-T) achieved an accuracy of 31.3%, 32.8% and 32.3%. We found no significant difference between difference cell/hidden sizes for LSTMs and GRUs. Regarding POS, only the trained version is available and LSTM (LSTM-POS-T) based on this feature obtained an accuracy of 10.4%, 11.1% and 10.5% while GRUs (GRU-POS-T) achieved 8.44%, 10% and 9.9%. Unlike LSTMs, GRUs with 100 and 200 units are not able to achieved significantly better results than a random classifier.

Conclusions Several observations can be made from the overall results presented in figures 4.6 and 4.7. First trained versions of each feature for both LSTMs and GRUs have less significant improvements with increasing cell/hidden sizes. We found significant improvements only for character bigrams (C2-T) and trigrams (C3-T) for LSTMs, and for character unigrams (C1-T) for GRUs. Although few in number, these differences are very significant ($\alpha = 0.1\%$) for character bigrams and trigrams-based LSTMs. As consequence, we can conclude that increasing n_h for character trigrams and bigrams can improve the accuracy of LSTM-based models. This is not the case with GRUs as we found no increase in accuracy for these features. This could be explained by the fact that GRUs are known to be more powerful than LSTM with the same number of units. Furthermore, we were unable to find any significant difference with these models, what allows us to think that, thanks to the regularisation of the dropout layer, increasing the number of units does not imply overfitting.

For pretrained features, we found some significant increases especially for character-based features but with weaker evidences than for the trained features. We can observe only two significant drops, the first with character unigrams-based LSTM (LSTM-C1-P) which goes from 48.3% to 29.4% with respectively 200 and 300 units. This difference is pretty strange and need further investigations. We also observe a drop with word embedding-based GRUs (GRU-WV-P), going from 83.1% for 200 units to 72.2% with 300 units. This could

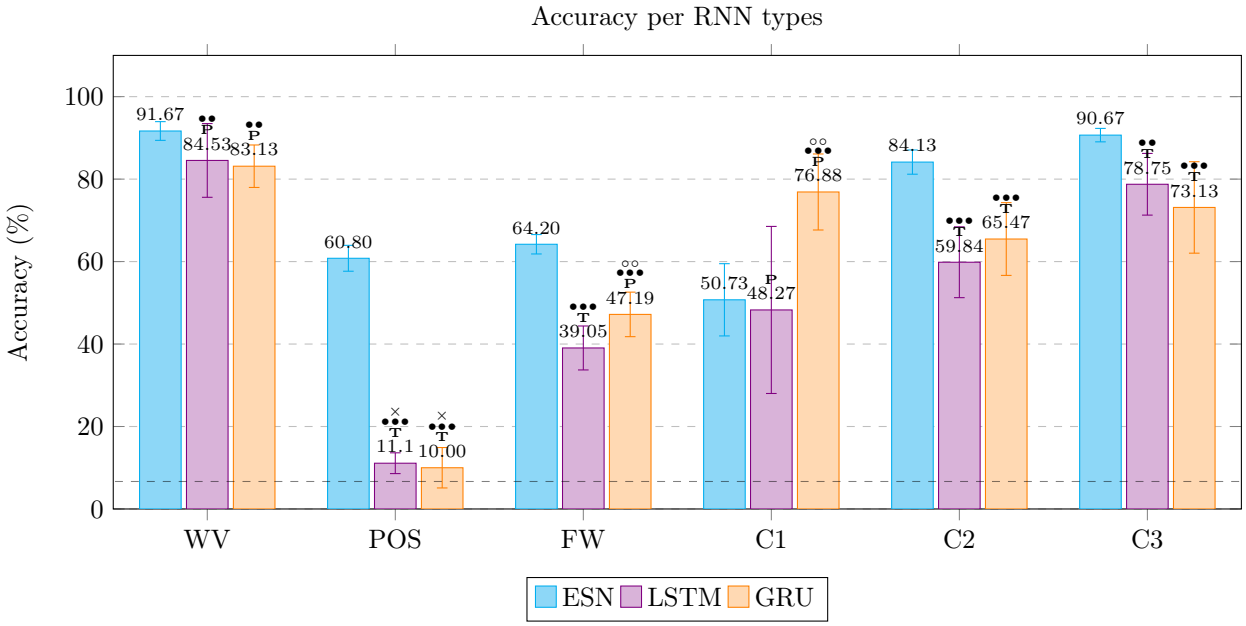


Figure 4.8 – Best 10-fold cross-validation accuracy per features for LSTM (violet), GRU (orange) and ESN (blue) on 15 authors of the Reuters C50 data set. Bullets (•) represent the statistical difference with ESN (• = 5%, •• = 1%, ••• = 0.1%) while circles (∘) represent the statistical difference with LSTM (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%). The cross (×) shows models with no statistically significant difference with the random classifier. *P* and *T* symbols indicate that the corresponding model uses respectively pretrained and trained feature.

be a sign of overfitting for this feature and further tests should be effectuated with different regularisation methods.

4.6 Comparison between ESNs, LSTMs and GRUs

Using results of the preceding experiments, we wanted to determine which model is the most effective for the different features. For ESNs, we used results of the experiments done in Section 4.4 averaged over samples in order to extract the average accuracy for each fold. For LSTMs and GRUs, we used the results of experiments done in Section 4.5 and selected the best model for each feature between the different cell/hidden sizes. Using 10-CV evaluation for each model, we computed statistical tests to compare each models and find significant differences. The result is presented in Figure 4.8. Average accuracy for ESNs, LSTMs and GRUs are presented respectively in blue, violet and orange. Bullets (•) represent statistical differences with the ESN model for the same feature (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent the statistical differences with the LSTM model for the same feature (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%). The cross (×) indicates that we were unable to find any significant differences with a random classifier. Finally, symbols P and T represent, for LSTMs and GRUs, whether the corresponding accuracy was obtained with trained (T) or pretrained (P) version of the feature.

Word embedding For word embedding-based RNN models, the highest accuracy is achieved by the ESN model (ESN-WV) with an accuracy of 91.7% against 84.5% and 83.1% for respectively LSTMs (LSTM-WV-P) and GRUs (GRU-WV-P). These last two models were significantly less effective than the ESN model but we were unable to find any difference between the LSTM and the GRU models. Both GRU and LSTM models used pretrained word embedding. Regarding variability, ESN has the lowest variation of its performance with a standard deviation of 2.27 against 8.94 and 5.15 for the LSTMs and GRUs. ESN models are clearly and significantly more effective on word embedding than LSTMs and GRUs.

Character embedding Regarding character trigrams (C3), the highest accuracy is achieved by the ESN model (ESN-C3) with an accuracy of 90.7% against 78.8% and 73.1% respectively for LSTMs (LSTM-C3-T) and GRUs (GRU-C3-T). We found evidences that these two models obtained an accuracy significantly lower than the accuracy achieved by the ESN. Both LSTM and GRU used trained character trigrams. Regarding variability, ESN obtained a standard deviation of 1.6 while LSTMs and GRUs are located at 7.5 and 11.09. ESN models are clearly and significantly more effective on character trigrams than LSTMs and GRUs.

For character bigrams (C2), the highest accuracy is also obtained by the ESN model (ESN-C2) with an accuracy of 84.13% against 58.8% and 65.4% for LSTMs (LSTM-C2-T) and GRUs (GRU-C2-T). The difference between ESN and both GRUs and LSTMs is significant while the difference between the LSTM and GRU models is not. Regarding variability, the standard variation of ESN's accuracy is 2.9 against 8.6 and 8.8 for respectively LSTMs and GRUs. Again, the ESN model is clearly and significantly better than LSTMs and GRUs on character bigrams.

Results are different with character unigrams (C1) as the highest accuracy is obtained by GRUs with pretrained vectors (GRU-C1-P) with an accuracy of 76.9% against 50.7% and 48.3% for respectively ESNs and LSTMs. This outcome is surprising regarding results obtained on other features. It could be the result of a problem in the implementation or in the methodology, but Figure 4.6 shows that similar results are obtained with different n_h values. Indeed, GRUs based on pretrained character unigrams (GRU-C1-P) with 100 and 300 units achieved an accuracy of 67.3% and 74.1% respectively. Why GRUs based on character unigrams are able to achieve such high accuracy compared to other RNN models and other features is an open question. Regarding variability, ESN stay first with a standard deviation of 8.8 against 20.3 and 9.2 respectively for LSTM and GRU.

Part-Of-Speech and Function Words For both POS and function words (FW), ESN models (ESN-POS, ESN-FW) achieve significantly higher accuracy than LSTMs and GRUs. For function words, ESN achieved an accuracy of 64.2% against 39.1% and 47.2% for LSTMs and GRUs respectively. Unlike other features, best accuracy is achieved with trainable function words for LSTMs and pretrained function words for GRUs. Regarding variability, standard deviation for ESN is 2.35 against 5.3 and 5.4 for LSTMs and GRUs respectively.

Regarding POS, ESN models is the only model to perform significantly better than a random classifier. ESNs achieved an accuracy of 60.8% against 11.1% and 10% for LSTMS and GRUs respectively. This result rises an interesting question : *why ESNs are able to achieve such as performance while LSTMs and GRUs are the state-of-the-art models in a lot of NLP tasks ?* One possible answer is that POS contains much less information as we removed all word meaning and as tokens are distributed along few dimensions. As the amount of information is reduced, the capacity of ESN to handle small dataset and its high capacity to generalise become more important. Regarding variability, standard deviation for ESN is 3.1 against 2.5 and 4.9 for LSTMs and GRUs.

4.7 Prediction certainty

The output of an RNN model gives us an authorship probability for a document, but a new question arise : *can we say how certain we are about a specific prediction ?* In order to give an empirical answer, we computed the probability of having the true author A knowing the output $\hat{y}_a$ which is the average output over time t of the predicted author a.

$$\hat{y}_a = \langle \hat{y}_{t,a} \rangle_t \tag{4.2}$$

Figure 4.9 presents the probability of having the true author according to output $\hat{y}_a$. We computed these probabilities for the top three features (WV, C3) found in the evaluation of ESNs on the test set during the 10-fold cross validation. We used the resulting 1,500 model outputs to evaluate their distribution according to the fact that the author had been correctly predicted or not.

Regarding pretrained word embedding (WV), the probability of being correct is of 50% for the lowest $\hat{y}_a$of 0.11 and rises to 99.999% for output values equal or above 0.23. For character trigrams (C3), the lowest $\hat{y}_a$ output of 0.09 gives a probability of being correct of 42.86% and rises up to 99.999% for output values equal or above 0.23. The average $\hat{y}_a$ for each feature is respectively 0.174 ($\pm$0.038) and 0.149 ($\pm$0.032) for word embedding (WV) and character trigrams (C3). For WV, the average prediction has 93.54% probability of being correct compared to 84.78% for C3.

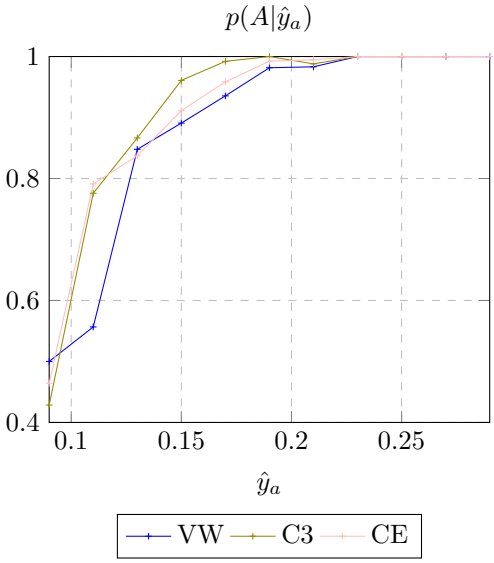


Figure 4.9 – Empirical probability of having the true author A according to the output authorship probability $\hat{y}_a$ of the predicted author a for the word-based and character-based features.

4.8 Comparison

To compare the differences we obtained on the various experiments, we selected interesting models of each type and feature. Table 4.3 shows the result of this selection and various information about these models. We added baseline models such as Support Vector Machines (SVM), Random Forest (RF) and Naive Bayes classifier baselines based on word and word bigrams. In addition to these baselines, we also evaluated a pretrained BERT model as it is currently the state-of-the-art model in document classification. These baseline models were implemented using the Sklearn framework and the BERT model with PyTorch. The document-level and the sentence-level columns show the accuracy of each model on the tasks of predicting the author of documents and the author of sentences respectively in two-authored documents.

The third column of Table 4.3 shows the corresponding standard deviation obtained with the 10-fold cross validation. The fourth column shows the number of parameters to be learned for each model. For the second task, we created two author-documents by concatenating two documents from the dataset which had different authors. The model had to predict the true author of each sentence and also use accuracy as the evaluation measure. To get our prediction at the sentence-level with ESN models, we followed exactly the same method as for document-level, with the exception of the last phase where we computed the maximum average over time not for the whole document but for each sentence.

The number of parameters to learn of an ESN model is the size of the output matrix ($\boldsymbol{W}^{out}$). In comparison to baseline models, ESN models, except ESNs based on character encoder (CE), have a lot fewer parameters to learn. In addition to the fact that they can be learned with linear methods, this shows that ESNs are much easier to train compared to other applications of neural models to NLP. For each ESN model, we used grid-search to find the best parameter values for the task. The resulting parameter values turned out to be the same as for the document-level task.

4.8.1 Document-level

At the document-level, the best model is an ESN with a reservoir of 1,500 units ($n_h = 1500$) using word-based representation (WV). To find this model, we generated 50 ESNs with random internal weights matrix ($\boldsymbol{W}$) and selected the one with the best 10-CV accuracy. This model obtained an accuracy of 93.40%, a model with 1,000 units ($n_h = 1000$) found with the same principle got the second best result (92.80%). The third position is occupied by the linear SVM baseline based on word 1-gram and 2-grams with 92.27%. The average reservoir ($n_h = 1000$) based on character encoder (CE), obtained the fourth place with 92.1%. However, we were unable to find any evidence of statistically significant differences between these models. They then composed the front runners of our evaluation with a respective standard deviation of 1.02, 2.59, 1.97 and

Classifier	Document	Sentence	Std. dev	Nb. params
Best of 50 ESN-WV (1,500)	**93.40 %**	76.88 %	**1.02**	1,125,000
Best of 50 ESN-WV (1,000)	**92.80** %	**84.76 %**	2.59	750,000
Linear SVM + Word 1-2 grams	92.27 %	72.09 %	**1.55**	317,940
ESN-CE (1,000)	92.07 %	80.00 %	1.97	325,800
ESN-WV (1,000)	91.67 %	83.27 %	2.27	**15,000**
Pretrained BERT	91.52 %		3.49	100 millions
ESN-C3 (1,000)	90.67 %	75.90 %	**1.63**	**15,000**
Naive Bayes TF-IDF, Word 1-2	89.20 %	74.86 %	2.26	3,559,500
LSTM-WV-P	84.53 %	**83.45** %	4.57	725,715
ESN-C2 (1,000)	84.13 %		2.94	**15,000**
GRU-WV-P	83.13 %	**86.57** %	5.15	303,615
Random Forest + BOW	80.67 %	67.47 %	2.91	
LSTM-C3-T	78.75 %		7.48	1,064,625
GRU-C1-P	76.88 %		9.22	129,615
GRU-C3-T	73.13 %	61.01 %	11.09	
GRU-C2-T	65.47 %	65.07 %	8.82	
ESN-FW	64.20 %		2.35	**15,000**
ESN-POS	60.80 %		3.14	**15,000**
LSTM-C2	59.84 % (T)	66.38 % (P)	8.59	444,995
ESN-C1	50.73 %		8.76	**15,000**
LSTM-C1-P	48.27 %	44.80 %	20.26	172,000
Random baseline	6.67 %	6.67 %	0	0

Table 4.3 – Comparison of 10-fold cross validation accuracy of baselines and RNN models.

1.55.

The next place is occupied by average ESN models based on word vectors (WV) with a reservoir size of 1,000 units with an accuracy of 91.67%. A pretrained BERT model arrives at the sixth place with an accuracy of 91.52%. The next place is occupied by average ESN models based on character trigrams (C3) with a reservoir size of 1,000 units with an accuracy of 90.67%. We found no significant differences between these tree models.

The next place is occupied by the Naive Bayes baseline based on TF-IDF and word unigrams and bigrams with an accuracy of 89.20%. Pretrained word-based LSTMs, character bigrams-based ESN and word-based GRUs come next with respectively 84.53%, 84.13% and 83.13%. We found no significant differences between these models. These models also outperform the BOW-based Random Forest baseline which obtained an accuracy of 80.67%. This model is the last baseline in our comparison. Of all RNNs types, word-based LSTMs and GRUs, and ESN based on words, character trigrams and character encoders outperformed this last baseline.

Character-trigrams LSTM (C3), unigrams GRU (C1) and trigrams GRU (C3) come next with respectively 78.75%, 76.88% and 73.13% and no significant differences between them. ESN models based-on function words (FW) and Part-Of-Speech (POS) come next with respectively 64.20% and 60.80%. The bottom of the pack is composed of LSTM based on character bigrams (C2), ESNs and LSTMs based on character unigrams (C1)

with respectively 59.84%, 50.73% and 48.27%. We found no differences between these models. All models we selected obtained an accuracy significantly better than the random classifier (6.67%).

4.8.2 Sentence-level

At the sentence-level, the best model on document-level (50 best ESNs with 1,500 neurons) reached an accuracy of only 76.49%. Compared to the best model, a GRU based on pretrained word embedding, which obtained an accuracy of 86.57%. This showed a strong overfitting of bigger reservoir sizes on this task. The second and third models were the best of 50 ESNs (reservoir size 1,000) based on pretrained word embedding and a LSTM also based on pretrained word embedding (WV) with an accuracy of respectively 84.76% and 83.45%. We were not able to find significant differences between these three models and they therefore compose the front runners of this evaluation on this task.

The fourth is the average ESN ($n_h = 1000$) based on pretrained word vectors (WV) with an accuracy of 83.27%. In comparison, ESN based on character encoder (ESN-CE) achieved an accuracy of 80.00% which is surprising as this model obtained a better accuracy than the word-based ESN (ESN-WV) on the document classification task. We found a significant difference between these two models.

The model mentioned before (best of 50 ESNs with 1,500 neurons) achieved an accuracy of 76.49% while the character trigrams-based ESN (ESN-C3) achieved 75.90%. The Naive Bayes classifier and the character unigrams-based GRU obtained respectively an accuracy of 74.86% and 72.44%. We were not able to find any significant difference between these four models. The best baseline is the Naive Bayes classifier based on TF-IDF and word bigrams and unigrams with 74.86%.

The second and third baselines are the Support Vector Machines (SVM) based on word bigrams and unigrams and he Random Forest based on BOW with respectively 72.09% and 67.47%. These results showed that the RNN's capacity to take textual documents as streams of text strongly increases the prediction accuracy at the sentence-level, as compared to classical models, and that the shallow architecture of ESN and its capacity for very long-term contextualisation gives it an edge over deeper architectures on smaller datasets at the document-level.

The bottom of the pack is composed of the LSTM and GRU based on character bigrams (C2) followed also by GRU and LSTM but based on character trigrams (C3) with respectively an accuracy of 66.38%, 65.07%, 61.01% and 49.11%. GRUs and LSTMs are better ranked at the sentence-level (first and third respectively) while SVM are less efficient than on document-level with 72.09% and is ranked second of baseline models compared to its first place on the document classification task. This could show a memory effect as LSTMs and GRUs can model shorter time dependencies but with higher accuracy than ESNs. Models outperformed by all baselines are basically the same as at document-level. As for document-level, models with high number of parameters don't have significantly higher accuracy than shallower models as the set of front runners is composed of word-based GRU, best-of ESN, LSTM and average ESN with 86.57%, 84.76%, 83.45% and 83.27% with respectively 303,615, 750,000, 725,715 and 15,000 parameters to be learned. Similarly to document-level, best results are obtained by models based word embedding.

4.9 Conclusions

In this chapter, we investigated the performance of ESNs, LSTMs and GRUs on two authorship attribution tasks with 15 authors of the C50 Reuters dataset at the document and sentence levels. We demonstrated how RNN models can be used to extract evidences out of classification decision and how to visualise them. We designed a set of experiments to evaluate models performances and determine how their parameters influence their accuracy using 10-fold cross validation and statistical tests. We first presented the results of two experiments which were designed to evaluate respectively how the leak rate and the spectral radius influence ESN's accuracy. We then analysed how two hyperparameters, the reservoir size and the training set size, influence ESN's performances. To this end we computed the 10-CV accuracy for different values of the reservoir size and for different training sizes.

To have a precise evaluation for each features, we computed the average accuracy for each features over 20 ESNs and we determined which features perform best on the whole data, and on a smaller dataset composed of 10 files per author. We then analysed how lexical and syntactic features and which ones outperform the other. Regarding LSTMs and GRUs, we set up an experiment to precisely compare trained and pretrained feature for the various representations. We use the same method to compare RNN models with different cell sizes and we compared their respective performance. Furthermore, the main question was to determine

which RNN architecture obtained the highest accuracy, we then used results from the various experiments we performed to compare ESNs, LSTMs, GRUs and the various baseline methods.

Regarding leak rate, we observe that the dynamics for the task of authorship attribution and textual data is very slow. Furthermore, we showed that pretrained word embedding (WV) is immune against chaotic behaviours up to a spectral radius of 2.0. A property that character-based features do not have. Regarding spectral radius, we did not observed any rise of performance just before the border of chaos but a significant drop of value above this limit. For reservoir size, we observed that very poor performances of the character trigrams with small reservoir sizes (below 200 units) and we made the assumption that this feature needs more memory as longer sequences of character are needed.

We showed that for most feature accuracy does not significantly improved for reservoir size above 500 units. Our experiment designed to evaluate the various models on a smaller dataset composed of 10 documents per author showed the impressive capacity of ESN for fast learning from a small dataset compared to baseline models and other RNN architectures (LSTM). For ESNs, we showed that increasing the number of characters, going from character unigrams to character bigrams, significantly increases accuracy. Regarding features, our results show that the word-based features (WV) and deep character encoder (CE) are the most efficient features, closely followed by character trigrams. We evaluated the possibility to use pretrained deep feature extractor as ESN inputs and the results of our experiments show that it does not significantly improved the accuracy in comparison to simple word-based embedding layer. Furthermore, it also strongly increased the training time from few minutes to several hours. On another side, syntactical features such as POS and function words got the worse results compared to word-based and character-based representations with all types of RNNs. With ESNs, POS and function words got results below the other features except for the single character embedding (C1).

For LSTMs and GRUs, we showed that pretrained word embedding (WV) provides the most efficient feature. We also demonstrated that pretrained features are more efficient that there trained counterparts except for character bigrams (C2) and trigrams (C3). This is probably due to the increasing number of parameters to be learned with trainable embedding layer and the small size of the dataset. Both LSTMs and GRUs are the most efficient with word-based features, especially with pretrained word embedding. GRUs and LSTMs based on this feature is the only one able to outperform one of the baseline, the Random Forest, while the other features for these networks are all outperformed by our baselines. GRUs got similar results to LSTMs on all features except character unigrams (C1) but with a smaller number of parameters. Character-based features slightly increase with increasing n-grams except with GRU where the single character embedding receives the highest accuracy. Regarding syntactic features, models based on POS were not able to learn the task with accuracy significantly higher than a random classifier when based on POS or function words.

For LSTMs and GRUs, we analysed the importance of cell and hidden sizes respectively by designed an experiment evaluating gated models with different parameter value. We concluded that increasing cell size for LSTM for character trigrams and bigrams can improve accuracy and that increasing the number of units does not imply overfitting on this task, thanks to the regularisation of dropout layers. For pretrained features, we found some significant increase for character-based representations but with weaker evidences and some problem of overfitting with word-based GRUs.

Compared to ESNs, gated networks are clearly outperformed with all features except single character pretrained vectors. This is surprising as ESN is much less powerful compared to deep networks such as LSTMS and GRUs. This can be explained by the fact that ESNs are less powerful but less subject to overfitting, which is important in the case where we want to predict a properties based on a small amount of data. Another hypothesis is the capacity of ESNs to model very long-term relationships. This hypothesis is supported by the fact that gated networks achieved better results at the sentence-level where the necessity to model long-term links is less important.

By comparing models, we showed that four models compose the front runners of our evaluations, best ESNs (with 1,000 and 1,5000 units) out of 50, ESN based on character encoder (CE) and the linear SVM based on word unigrams and bigrams. We were not able to find any differences between these models which show that RNN architectures such as ESN can compete with state-of-the-art methods such as SVMs and Random Forest while having a very small number of parameters and a very short training time. In addition to these results, we demonstrated the possibility to extract visual evidences out of ESN's predictions, a mandatory property in the field of authorship attribution.

Chapter 5

Author Profiling

In this chapter, we present results of the evaluation of RNN models introduced in Chapter 3 on the Author Profiling task using the PAN@CLEF 2017 dataset. Here models must predict if a set of 100 tweets has been written by a man or a woman. The performance is evaluated by accuracy rate. We compared different RNN models with a baseline composed of a random classifier, a Naive Bayes classifier and character bigrams-based convolutional neural model (CNN), and used statistical tests to find significant differences.

In Section 5.1, we introduce the methodology used to train RNNs and test their predictions. In Section 5.2, we analyse the results of the CNN baseline and the impact of different features on the accuracy. Sections 5.3 and 5.4 analyse respectively results obtained by ESN and GRU models and the impact of their various parameters. Section 5.5 will presented results obtained with various features and neural architectures. Finally, Sections 5.6 and 5.7 analyse results obtained with the different RNN architectures, compare their accuracy with those obtained by baseline models and draw some general conclusions on the overall results obtained in this chapter.

5.1 Methodology

The output of the RNN on this task is $\hat{y}(t) = \{p(class = male), p(class = female)\}$ where $p(class = male)$ is the estimated probability that the word-tokens in the memory at time t have been written by a male. Similarly, $p(class = female)$ is the estimated probability that the word-tokens at time t have been written by a female and is equivalent to $1 - p(class = male)$. To process data from a Twitter profile with many-to-many architectures, we concatenate all tweets into a single document and we use that as an input for the RNN, we then end up with an output timeseries of estimated gender probabilities.

To evaluate and compare models based on RNNs with different parameters and baseline, we evaluated accuracy on the PAN17 dataset using 10-fold cross validation (CV-10). For ESNs, once the optimal leak rate found, we tested different reservoir sizes and we choose the optimal size for this task. Finally, we used bilateral t-tests to search for significant differences in evaluated accuracy for different models and parameters.

PAN is a series of scientific events and shared tasks on authorship analysis and text forensics while has been operated for many years at the CLEF conference. The dataset and task analysed in this chapter come from the 2017 edition of the PAN laboratory. Different models have been proposed and evaluated by different teams and some were selected for presentation. Compared to the methodology used in this chapter, PAN used the whole dataset to develop models and for training, and an independent training set to evaluate proposed methods. We use here the training set from PAN for our 10-fold cross validation.

Table 5.1 shows some models proposed during this shared tasks and the associated evaluated accuracy on the English subset of the original dataset. We were not able to recover which model has been used and their properties except for three of them, including the method which obtained the best accuracy. Table 5.1 show that the best model is an SVM (the type of kernel is not specified) based on character n-grams with TF-IDF features. In comparison with our results, SVM also obtained the best accuracy but our was based on words unigrams and bigrams also with TF-IDF features. Two other methods proposed in 2017 at PAN were based on logistic regression and SVM both with various text transformation and feature combinations. To our knowledge no proposition were made with deep learning or neural network-based methods at PAN 2017 except the same CNN baseline (CNN-C2) we used in this study and which obtained an accuracy of 74.8%, compared to the 78.3% obtained here.

Ranking	Classifier	Test accuracy
1	**SVM character n-grams TF-IDF** [170]	**82.3** %
2	Lopez-Monray et al. [171]	81.7 %
3	Markov et al. [171]	81.3 %
4	Logistic regression character n-grams, POS n-grams, emoji [172]	80.7 %
5	MicroTC (SVM, various text transformation and feature combinations) [173]	80.5 %
6	Franco-Salvador et al. [171]	79.6 %
7	Kodiyan et al	78.9 %
8	Ogalstov & Romanov [171]	78.8 %
9	Poulston et al. [171]	78.3 %
10	Ganesh [174]	78.3 %
11	Sierra et al. [171]	78.2 %
12	Ciobanu et al. [171]	76.42 %
13	CNN 2-grams + starting-grams + ending-grams [175]	74.8 %
14	LDR baseline [171]	72.2 %
	Random baseline	50.0 %

Table 5.1 – 10-CV success rates of baselines and both models on PAN17 gender profiling task.

For our baseline neural model (referred as CNN-C2), we choose a Convolutional Neural Network (CNN) based on a feed-forward architecture. A CNN is a variety of FFNNs inspired by the visual cortex [176]. We applied a CNN to a representation matrix containing the relative frequency of character bigrams of all tweets in a Twitter profile. Figure 5.1 shows the structure of the representation matrix. For each letter, one can find

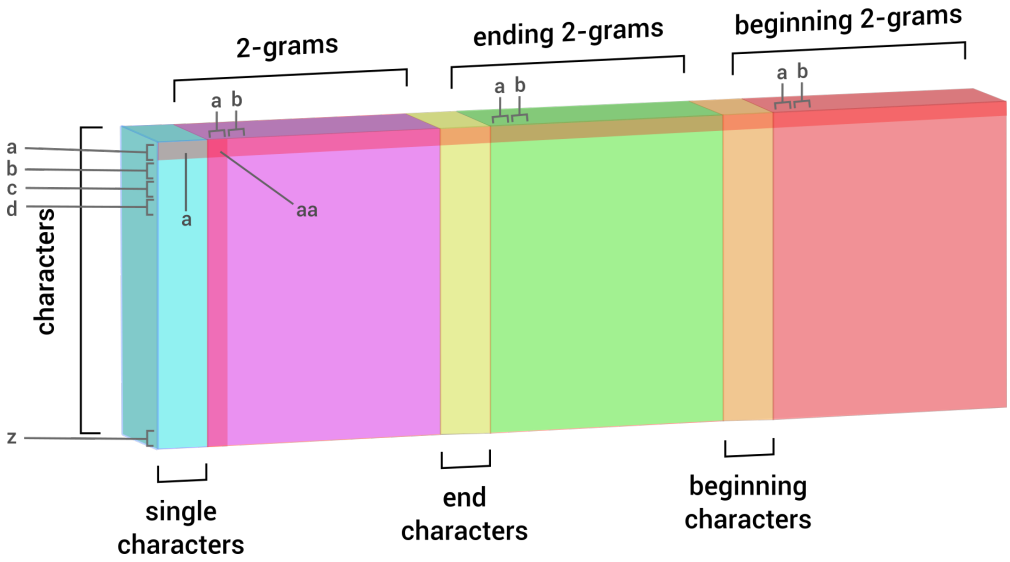


Figure 5.1 – Structure of the input features matrix. From left to right, the matrix represents frequencies of character 2-grams, frequencies of starting character 2-grams, and frequencies of ending character 2-grams.

one row. In the first position, the relative frequency of this letter is provided. Then, from left to right, the matrix is composed of the relative frequencies of each character bigram (e.g., at row "t" and column "h", the relative frequency of the bigram "th" is given). The third part is optional and contains the relative frequency of character bigrams at the end of words. Finally, the last part is the same optional matrix representing the relative frequency of character bigrams at the beginning of words. This parameterised matrix represents relative frequencies of character bigrams for a single author and is used as input for the CNN.

The first two layers are convolution layers composed of 20 and 10 convolution kernels respectively, with a size of 5×5. These layers are followed by a dropout layer. The last two are linear layers based on ReLU. Outputs are finally obtained by a softmax function which model a probability distribution over the two possible classes (men/women). The predicted class is therefore the class with the highest corresponding output from this function. The training set is composed of 90% of the dataset and the remaining 10% is use to estimate the performance. This procedure is repeated 10 times with non-overlapping test sets to obtain the 10-fold cross validation estimator. To train this model, we used the classical SGD algorithm with a learning rate and a momentum respectively set to 0.01 and 0.5. We run the training for 100 epochs with a batch size of 64, and the Negative Log-Likelihood (NLL) as the loss function.

For GRU models, we train each model to maximise the average output probability for the true class using a Cross-Entropy loss function. GRUs take as input a single tweet and the average output probability is computed over the 100 tweets of a Twitter profile to give a final classification. The same network process each tweet but is trained to maximise the average over the whole set of tweets for a profile. Different models are evaluated on different features with an input embedding layer for trained features with dictionary sizes corresponding the number of words/character found in the dataset. All models were evaluated with a dropout layer inserted before the output layer. This layer set values obtained from connections with the previous layer to zero with a probability of 50%. We trained each model for different number of epochs going from 100 to 250 depending on the model requirements. We used a learning rate of 0.001 with a batch size between 64 and 128 depending on memory size required by the evaluated feature.

5.2 Convolutional Neural Network Baseline

Table 5.2 shows results obtained with this deep learning CNN model based on character bigrams (CNN-C2) with different vocabulary and starting and ending character bigrams (second row). Concerning starting bigrams, we can observe that the results obtained with this additional feature is above alone bigrams for the three possible vocabulary (English, punctuation and punctuation + smiles).

Alone bigrams obtained an accuracy of respectively 75.26%, 76.16% and 76.51%, for English, punctuation and punctuation with smilies, against 76.02%, 77.63% and 77.50% with additional starting bigrams. Statistical tests ($*$) indicate that the starting bigrams can significantly improve the performance compared to the bigrams model (first row) with the three alphabets.

Regarding ending bigrams (third row), Table 5.2 shows the results obtained by the CNN-C2 model with this additional feature. With this feature, the CNN-C2 model obtained an accuracy of 75.94%, 77.22% and 77.25% respectively for vocabulary composed of the English alphabet, additional punctuation, and additional

Matrix / Alphabet	**English**	**+ Punctuation**	**+ Punctuation & Smilies**
Bigrams	75.26%	76.16%	76.51%
+ starting bigrams	76.02%*	77.63%*†	77.50%*
+ ending bigrams	75.94%	77.22%†	77.25%
+ starting & ending bigrams	76.12%	77.83%†	78.33%*†

Table 5.2 – 10-CV accuracy rates of different feature matrices and alphabets. Symbols $*$ and † mean that there is a significant difference on the vocabulary (columns) and grams (rows) axis respectively, with a 5% significance level (t-test) compared to the previous line/column.

smilies. Statistical tests (∗) show that ending bigrams cannot significantly improve the performance compared to starting bigrams (second row) with all tested alphabets.

Concerning the combination of starting and ending bigrams, Table 5.2 shows that CNN-C2 models with these two additional features obtained 76.12%, 77.83% and 78.33% respectively for the three different vocabularies. Statistical tests (∗) indicate that this additional feature can significantly improve the accuracy only in the case one use the punctuation and smilies in addition to the basic English alphabet.

For the different alphabets, CNN-C2 models based on the basic English alphabet (second column) obtained an accuracy of 75.26%, 76.02% and 76.12% respectively for the three different features evaluated. In comparison, the addition of punctuation (third column) obtained an accuracy of 76.16%, 77.63% and 77.83% respectively for the three features. Statistical tests (†) show here that the addition of punctuation only significantly improved performance with starting bigrams but not for the other features. The addition of smilies (fourth column) obtained an accuracy of 76.51%, 77.50% and 77.25% respectively for the three features. The statistical tests (†) demonstrated that smilies can significantly improve performance when one use starting and combination of starting and ending bigrams as features.

5.3 Echo State Networks

To analyse dynamical properties of each textual features on this task, we evaluated the performance of ESNs with various leak rates from 0.0001 to 1.0. For each leak rate, matrices $\boldsymbol{W}$, $\boldsymbol{U}$ and $\boldsymbol{b}$ were kept constant to have similar models with different properties.

The left plot in Figure 5.2 shows the six features (WV, POS, FW, C1, C2 and C3) with a leak rate between 0.0001 and 1.0. Pretrained word vectors (WV) obtained the highest accuracy with 80.81% for a leak rate of 0.01. The accuracy obtained by this feature goes from 80.81% with a slow dynamics (0.01) to 73.86% with a fast dynamics (1.0). For POS, the highest accuracy (70.2%) is obtained with a leak rate of 0.05, while normal dynamics (leak rate of 1.0) obtained an accuracy of 62.08%. For function words (FW), best accuracy is obtained (71.91%) with a leak rate of 0.05 against 68.17% for normal dynamics (leak rate of 1.0).

For character based features, pretrained embeddin of character unigrams (C1) obtained an accuracy of 65.45% for a leak rate of 0.4 against 59.81% for a normal dynamics (1.0). For pretrained embedding of character bigrams (C2), the best accuracy (68.5%) is obtained with a leak rate of 0.01 against 55.5% for a normal dynamics (1.0). Finally, for pretrained embedding of character trigrams (C3), the best accuracy

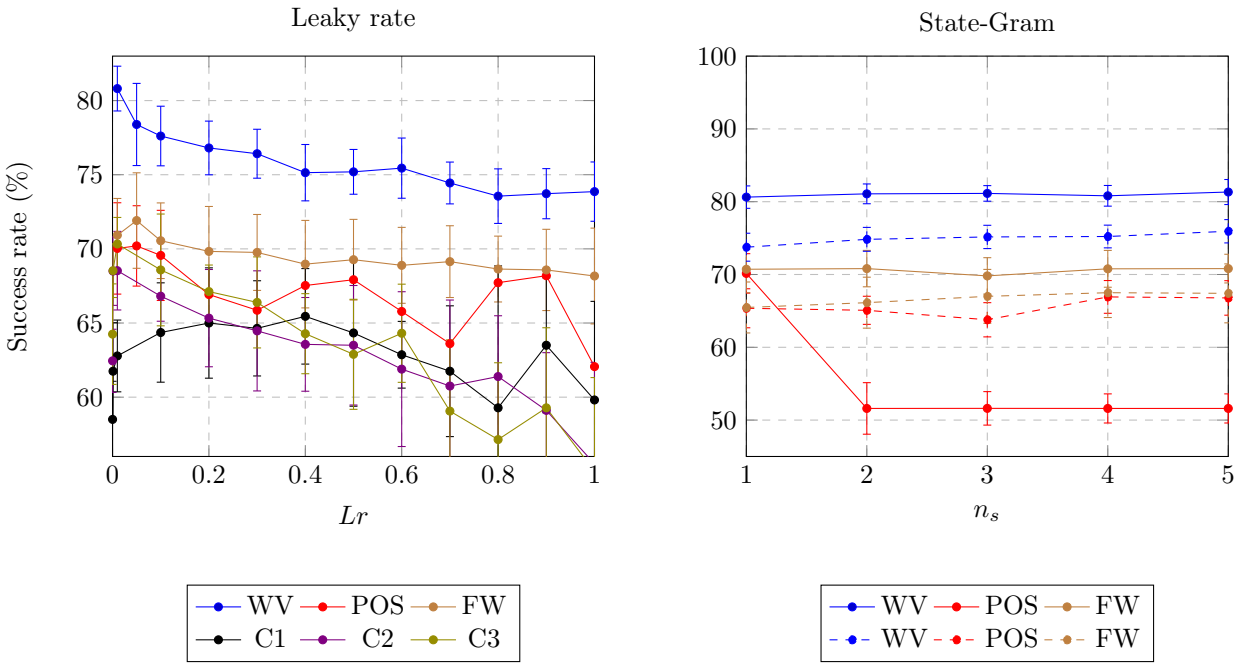


Figure 5.2 – 10-fold cross validation success rate on the PAN17 tweet collection for each leaky rate and state-gram with $n_h = 1000$.

(70.3%) is obtained with a leak rate of 0.01 against 55% with a normal dynamics (1.0). Between different character-based features, pretrained embedding of character trigrams obtained the highest accuracy with 70.33% against 68.5% and 65.45% for C1 and C2 respectively. We found significant differences between C1 and C2, and between C1 and C3, but not between C2 and C3.

Pretrained word vectors (WV) obtained the best result with an accuracy of 80.81% while the second and third best are the function words (FW) and character trigrams (C3) with respectively 71.91% and 70.33%. POS arrives after with an accuracy of 70.2%. Character bigrams and unigrams arrives last with respectively 68.5% and 65.45%. We were not able to find any significant difference between function words (FW), character trigrams (C3), Part-Of-Speech (POS) and character bigrams (C2). Our results showed that features can be separated in three groups, first pre-trained word embedding (WV) which achieved an accuracy significantly higher than the other features, followed by a group composed of function words (FW), character trigrams (C3), POS and character bigrams (C2) with no significant differences. Finally, character unigrams (C1) arrives last with an accuracy significantly lower than the other features.

Comparing dynamical properties of different features, we can observe that the best accuracy is obtained for the each feature (WV, POS, FW, C1, C2, C3) with a leak rate of respectively 0.01, 0.05, 0.05, 0.4, 0.01 and 0.01. For all features except character bigrams (C2), we can observe that they display very slow dynamics on this task.

In addition, the output layer of ESNs can take account of more than one state to estimate the class probabilities. A *state-gram* value of 2 means that the training is performed, not only on a single $x(t)$, but on the union of the current state $x(t)$ with the previous state $x(t-1)$ ($x(t-1) \cup x(t)$). Such a model was effective for handwritten digits recognition [177]. The right plot in Figure 5.2 shows the accuracy obtained state-grams going from 1 (normal ESN) to 5 for pretrained word vectors (WV), POS and function words (FW). Slide lines shows accuracy obtained for ESN with the corresponding best leak rate, while dashed lines indicate accuracy obtained with ESN and leaky rate set to 1.0.

For pretrained word vectors (WV), accuracy shows no sign of significant improvement for fast dynamics (1.0) and slow dynamics (0.01). For POS, we observe no significant improvement for fast dynamics (1.0) but a significant drop of accuracy for slow dynamics (0.05), going from 70.15% for normal ESN to 51.60% with a state-gram value set to 5. For function words (FW), we also observe no significant improvement of the accuracy with higher state-gram values for both fast (1.0) and slow dynamics (0.05). In conclusions, while this parameter can improve accuracy on tasks such as image classification, we observe on this task no significant improvement with this additional parameter.

5.4 Gated Recurrent Units (GRU)

Gated Recurrent Units (GRUs) can use pretrained features but also, unlike ESNs, trained features. In the case of trained features, the projection matrix $\boldsymbol{W}^e$ is trained directly on the task at hand. A main question is then : *which representation can achieved the highest accuracy ?*

Therefore, we did an experiment to answer this question which consists to evaluate the accuracy of GRUs based on trained and pretrained version of each feature while using statistical tests to find significant differences. Also for GRUs, the size of the hidden layer influence the memory and complexity of the model. As a consequence, we did a second experiment in which we evaluated the accuracy of GRUs with different hidden sizes in order to find significant differences based on this parameter.

Pretrained/trained features Figure 5.3 shows accuracy obtained by GRUs based on trained and pretrained embedding layers. Red, orange, violet and green showing respectively accuracy rates obtained with word embedding (WV), character unigrams (C1), character bigrams (C2) and character trigrams (C3). Bullets (•) represent the results of statistical tests performed between trained and pretrained version of each feature (• = 5%, •• = 1%, ••• = 0.1%).

For word-based feature (WV), the accuracy rate obtained with pretrained layer (GRU-WV-P) is slightly higher than with a trained layer (GRU-WV-T) with respectively 76.6% against 75.5% but with no significant difference. For character unigrams (C1), the trained version (GRU-C1-T) clearly outperform the pretrained version (GRU-C1-P) with respectively 53% and 63.8% with a strong significant difference ($\alpha = 0.1\%$). Regarding character bigrams (C2) and character trigrams (C3), trained versions (GRU-C2-T) also outperform their pretrained counterparts (GRU-C2-P) with 55.00% and 66.44% for C2, and 61.28% and 68.83% for C3. For this two features, the difference is strongly significant ($\alpha = 0.1\%$).

Regarding accuracy obtained by various features, we can observe that pretrained and trained version of word embedding (WV) obtained the highest accuracy with 76.2% and 75.5% respectively. In comparison, the best accuracy obtained by each character-based feature are respectively of 63.8%, 66.4% and 68.8% for C1, C2 and C3. We can also observe that accuracy seems to improve with increasing character n-grams going from 63.8% to 66.4% for C1 and C2, and from 66.4% to 68.8% for C2 and C3. We did not look for significant differences at this point because that is the aim of Section.

Hidden sizes Regarding GRUs, an important parameter is the hidden size conditioning the memory capacity and the complexity of the model. It basically sets the number of units used in the memory cell. Figure 5.4 shows the accuracy obtained by GRUs with different hidden sizes for each feature. Accuracy per size is represented for each feature, words (WV), character unigrams (C1), character bigrams (C2) and character trigrams (C3). For all test models, we inserted a dropout layer, with a probability of 50% for a unit to be put to zero, between the output of the GRUs and the output linear layer. This is a well-known method to reduce overfitting.

Orange, violet and blue bars shows respectively the accuracy obtained by GRUs with 25, 50 and 100 units. Bullets (•) represent the statistical difference with the 25-units model for the corresponding feature (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent statistically significant differences with the 50-units model ($n_h = 50$) for the same feature (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%).

For word-based features, accuracy increases with increasing cell size for pretrained models (GRU-WV-P) with 66.5%, 69.9% and 76.2% for respectively cell with 25, 50 and 100 units. The model with 100 units ($n_h = 100$) obtained an accuracy significantly higher than GRUs with 25 and 50 units. This observation is reversed with trained models (GRU-WV-T), as accuracy is going down with 75.5, 75.3 and 71.8 for 25, 50 and 100 units. Inversely to the pretrained version, the model with 100 units achieved an accuracy significantly lower than the GRUs with 25 and 50 units.

For pretrained character unigrams (GRU-C1-P), accuracy slightly increases with increasing number of units in the memory cell with 51.5%, 52.8% and 53% respectively but with no significant improvement. For the trained version (GRU-C1-T), we observe a significant decrease in accuracy between GRUs with 25 and 100 units. For all other feature, we observe no significant differences between GRUs with different hidden sizes.

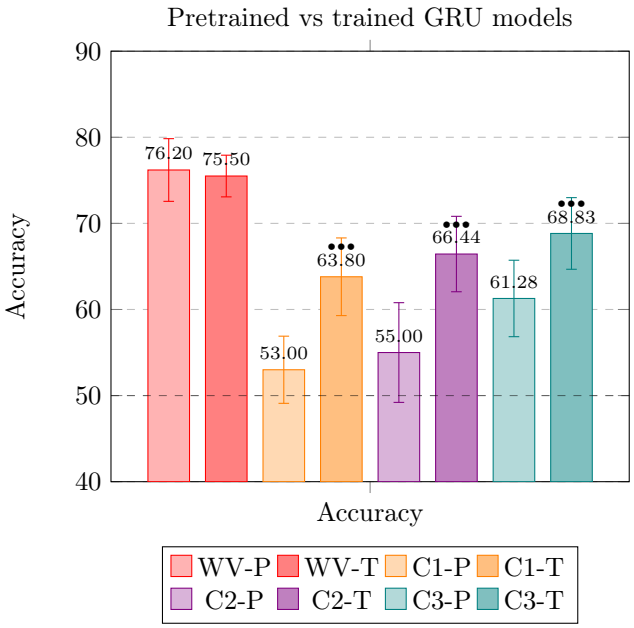


Figure 5.3 – Comparison of 10-fold CV accuracy between trained and pre-trained features for GRU models. For each feature, we selected the best model between the different cell sizes (25, 50, 100). Bullets (•) represent statistical significant difference between the pretrained and trained version of the same feature (• = 5%, •• = 1%, ••• = 0.1%).

Conclusions A main observation can be made from figure 5.4. As the accuracy increases or shows no significant improvements with increasing cell sizes with pretrained features, the phenomenon is reversed with trained models as their accuracy show signs of lower performances with increasing cell sizes for word and character-based models. A straightforward explanation for this phenomenon is that trained models have much more parameters as each embedding vectors is an additional parameter to train. This increases linearly the number of parameters as a function of embedding dimensions and vocabulary size. These models have then a much higher number of parameters, and are as a consequence more subject to overfitting.

5.5 Features and neural models

5.5.1 Features

The next step to investigate is to determine which feature between word-based and character-based features are the most effective for each model and in comparison with baseline models. Figure 5.5 shows accuracy obtained by the different models with word-based and character-based features. For each type of model, we selected the one which achieved the highest accuracy with the corresponding feature (word-based (WV) or character-based (C)). Bullets (•) represent significant statistical differences with respectively ESN-WV and ESN-C for word-based and character-based models (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent significant statistical differences for the same model between its word-based and character-based version (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%).

Word embedding For word-based feature, the best result is achieved by the SVM with 81.3% against 80.6% for the ESN is in second position. However, we found no significant differences between these two models. The third and fourth models are GRUs based on pretrained and trained word embedding with an accuracy of respectively 76.2% and 75.5%. The fourth and fifth highest accuracy is obtained by the Naive Bayes and the Random Forest classifier with respectively 75.5% and 74.56%. We observed no significant differences between these last four models.

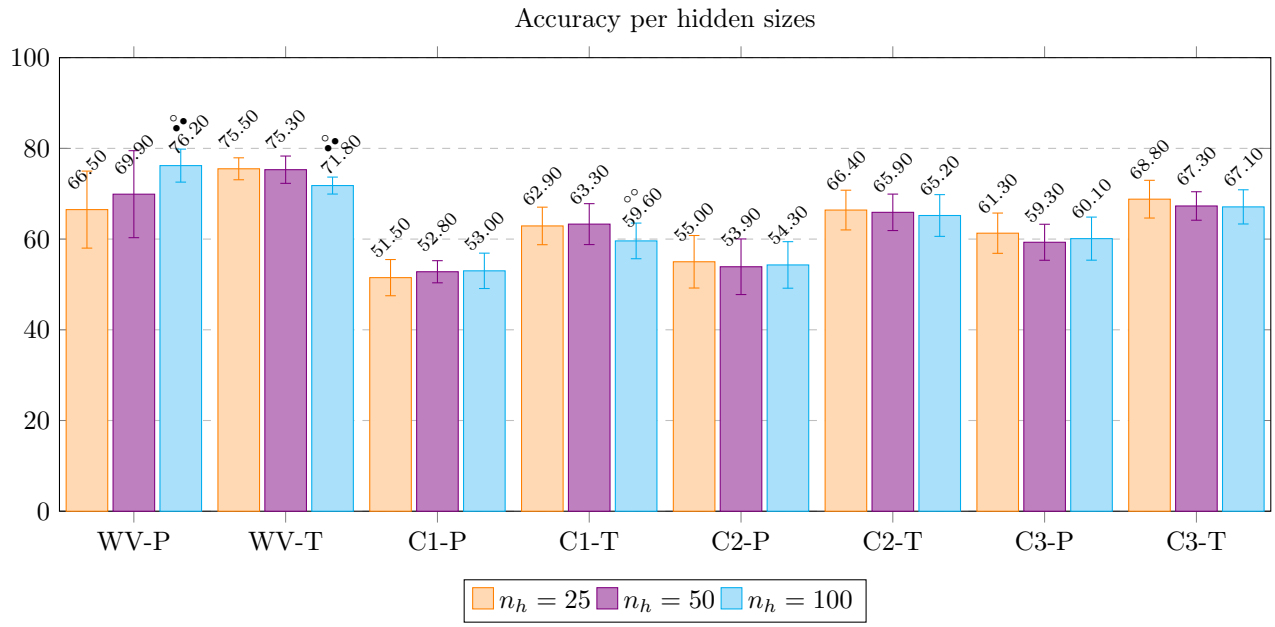


Figure 5.4 – Comparison of 10-fold cross validation accuracy on PAN17 dataset between GRU models with 25, 50 and 100 units in the memory cell. Bullets (•) represent statistically significant differences with the model 25 units model ($n_h = 25$) with the same feature (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent statistically significant differences with the model 50 units model ($n_h = 50$) with the same feature (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%).

Character embedding For the character-based feature, the best result is achieved also by the linear SVM with an accuracy of 79.92% against 78.3% for the deep learning CNN model with no significant differences. Random Forest classifiers and character trigrams-based ESN (ESN-C3) arrive after with respectively 75.47% and 70.33%. The bottom of the pack is composed GRUs based on trained and pretrained character unigrams (GRU-C3-P, GRU-C3-T) with respectively 68.83% and 61.28%. We found no significant differences between the trained character trigrams-based GRU (GRU-C3-T) and the pretrained character trigrams-based ESN (ESN-C3).

Word versus character Regarding differences between word-based and character-based features, Figure 5.5 indicates the significant statistical differences for the same model between its word-based and character-based version by a circle (∘). For all models tested, word-based feature outperformed the same model based on character-based feature or showed no significant difference. Word-based feature showed significantly higher accuracy only for RNN models (pretrained, trained GRUs and ESNs). We were unable to find any significant differences between word-based and character-based features for baseline models.

Another observation is that while RNNs are competitive with the word-based feature, they are not when used with character-based feature. For example, ESNs achieved an accuracy of 80.6% with pretrained word vectors and 70.33% with pretrained character vectors. This observation does not apply to the deep learning CNN model which is based on character feature but is able to achieve a competitive accuracy of 78.3%.

5.5.2 RNN architectures

Using results of the preceding experiments, we wanted to determine which model is the most effective for different features. For ESNs, we used results of experiments done in Section 5.3. For GRUs, we used the results of the experiments done in section 5.4 and selected the best model for each feature between the different hidden sizes. Using 10-fold evaluation for each model, we performed statistical tests to compare each models and find significant differences. The result is presented in Figure 5.6. Accuracy obtained by ESNs, GRU with pretrained embedding layer, GRU with trained embedding layer is shown respectively in

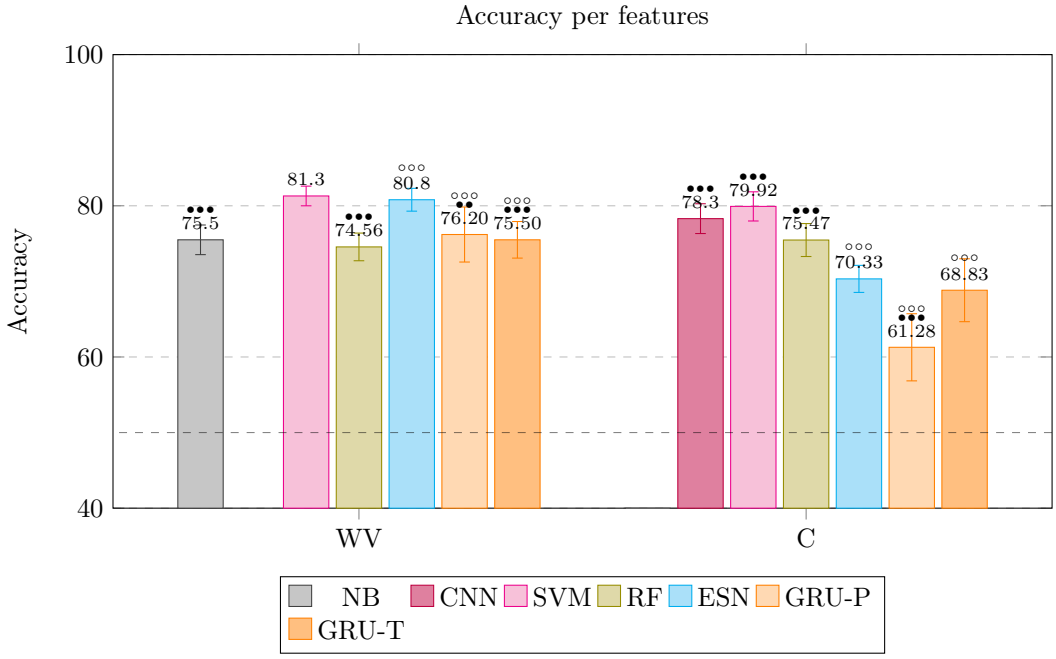


Figure 5.5 – Comparison of 10-fold cross validation accuracy on PAN17 dataset between word-based and character-based models. Best pretrained and trainged GRU models are selected for the comparison in all possible features (C1, C2, C3 for character based) and cell sizes ($n_h = 25, n_h = 50, n_h = 100$). Bullets (•) represent significant statistical difference with respectively ESN-WV and ESN-C for word-based and character-based models (• = 5%, •• = 1%, ••• = 0.1%). Circles (∘) represent significant statistical differences between version of the same model based on characters and words (∘ = 5%, ∘∘ = 1%, ∘∘ = 0.1%).

blue, orange and pink for word-based feature (WV), character unigrams (C1), character bigrams (C2) and character trigrams (C3).

For the word embedding (WV), ESNs (ESN-WV) achieved the highest accuracy with 80.6% against 76.2% and 75.5% for pretrained GRU (GRU-WV-P) and trained GRUs (GRU-WV-T). The accuracy obtained by ESN-WV is significantly higher than those obtained by GRUs, no significant difference was found between the two GRU models. Regarding character unigrams, ESNs (ESN-C1) got again the highest accuracy with 65.45% against 53% and 63.83% for pre-trained (GRU-C1-P) and trained GRUs (GRU-C1-T). There is no significant difference between the ESN model and the trained GRU model (GRU-C1-T), but the pre-trained GRU model got an accuracy significantly lower.

For character bigrams, ESNs (ESN-C2) still achieved the highest accuracy with 68.53% against 55% and 66.44% for pretrained (GRU-C2-P) and trained GRUs (GRU-C2-T) but with no significant difference between ESN and the trained GRU (GRU-C2-T) while the pre-trained GRU was significantly lower. Finally, for character trigrams (C3), ESNs (ESN-C3) still obtained the best accuracy with 70.33% against 61.28% and 68.83% for pretrained (GRU-C3-P) and trained GRUs (GRU-C3-T). Again, no significant difference was found between the ESN and the trained GRU model (GRU-C3-T) while the pretrained model was still significantly lower.

Between the different models, ESNs significantly outperforms pretrained and trained GRU models on word embedding with 80.6% against 76.2 and 75.5%. Regarding character-based features, ESNs and trained GRUs obtained similar result with no significant differences while pretrained GRUs stayed significantly behind. Between trained and pretrained GRUs, GRUs with trained embedding layer clearly outperform GRUs with a pretrained layer with a gain of 10.83, 11.44 and 7.55 for C1, C2 and C3 in favour of trained GRUs.

Comparing features, Figure 5.6 shows that the word-based feature is significantly the most effective with all RNN models, outperforming other features regardless of the model used. Concerning character-based features, character trigrams (C3) is the most effective outperforming character bigrams (C2) and unigrams (C1) regardless of the model used. Going from C1 to C2 significantly improved accuracy for ESNs but not for trained and pretrained GRUs. Going from C2 to C3 improved accuracy significantly for pretrained GRUs but not for trained GRUs and ESNs. Accuracy significantly improved when going from C1 to C3 for all models.

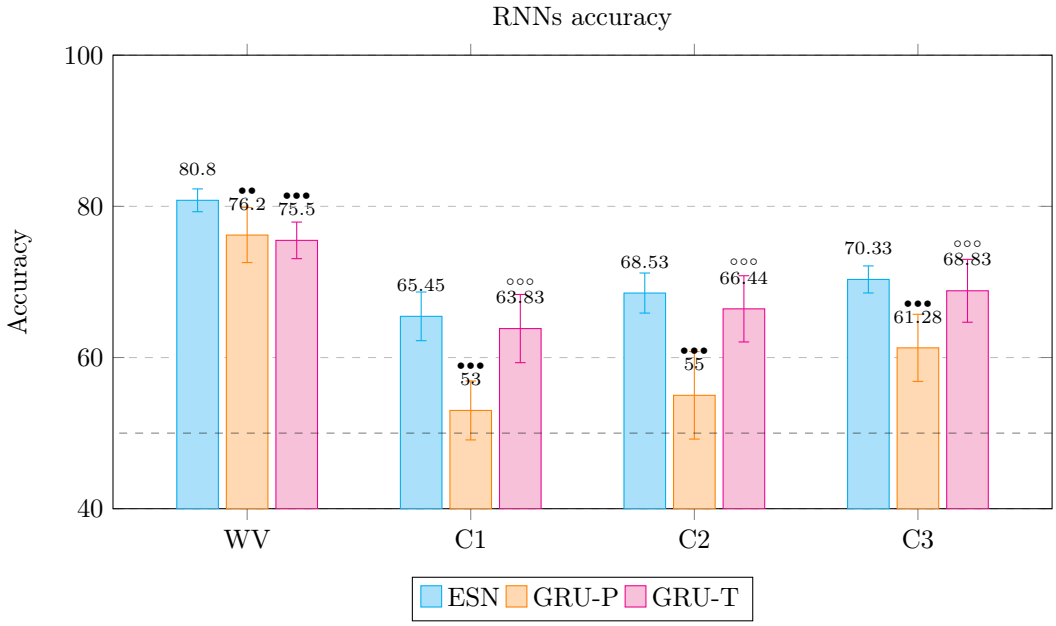


Figure 5.6 – Comparison of 10-fold cross validation accuracy on PAN17 dataset between RNN type (ESN, GRU pretrained, GRU trained) for each feature. For GRUs, we selected the best model for the different cell sizes (25, 50, 100). Bullets (•) represent statistical significant differences with the ESN model (blue) (• = 5%, •• = 1%, ••• = 0.1%), while circles (○) represent statistical significant differences with the pretrained GRU model (○ = 5%, ○○ = 1%, ○○ = 0.1%).

5.6 Models

We now want to compare all models evaluated so far using outputs of experiments we did in previous sections. Figure 5.7 shows the 10-CV accuracy obtained with different models and associated features on the PAN17 dataset. Furthermore, table 5.3 shows the various models with the evaluated accuracy on the profiling task (second column), the standard deviation (third column) and the number of parameters to be learned during the training phase (fourth column). For each model we selected the best accuracy obtained with this model and the corresponding feature over all possible parameters. The baseline is composed of word-based Naive Bayes classifier (NB-WV), SVMs based on bag-of-words (SVM-WV) and bag-of-characters (SVM-C), and Random Forest based on bag-of-words (RF-WV) and bag-of-characters (RF-C). We choose two reference models, namely the word-based ESNs and the character unigrams-based ESNs (ESN-C1). Bullets (•) are here to indicate significant differences with the first reference model (ESN-WV) while circles (○) represent significant differences with the second reference model (ESN-C1).

The highest accuracy is obtained by a linear SVM based on BOW (1 and 2-grams) with TF-IDF (SVM-WV) with an accuracy of 81.3%. The second best model is an ESN based on pretrained word vectors (ESN-WV) with an accuracy of 80.6% against 79.92% for respectively the third best model, the linear SVM based on bag-of-character. We found no significant differences between these three models and they composed therefore the set of front runners of our evaluations. Furthermore, they have respectively about 2 millions, 2,000 and 670 thousand parameters to be learned.

The character bigrams CNN model (CNN-C2) arrives in the fourth position with 78.3%. GRU based on pretrained word vectors (GRU-WV-P), Random Forest based on BOW (RF-WV), GRU based on trained word vectors (GRU-WV-T), Naive Bayes classifier based on bag-of-words and Random Forest based on bag-of-characters come after with respectively 76.17%, 75.92%, 75.5%, 75.5% and 75.47%. However, these results are so closed that we cannot observe any statistical difference between these models. Regarding variability, these models have a 10-fold cross validation standard deviation of respectively 3.6, 1.8, 2.4, 2.0 and 2.2. In addition, they have respectively 1.1 millions, 120 thousand, 9 millions, 4 millions and also 9 millions parameters to estimate.

All of the following models are based on neural networks (ESNs and GRUs) and character features. ESNs based on pretrained character trigrams (ESN-C3), GRU based on trained character trigrams (GRU-C3-T),

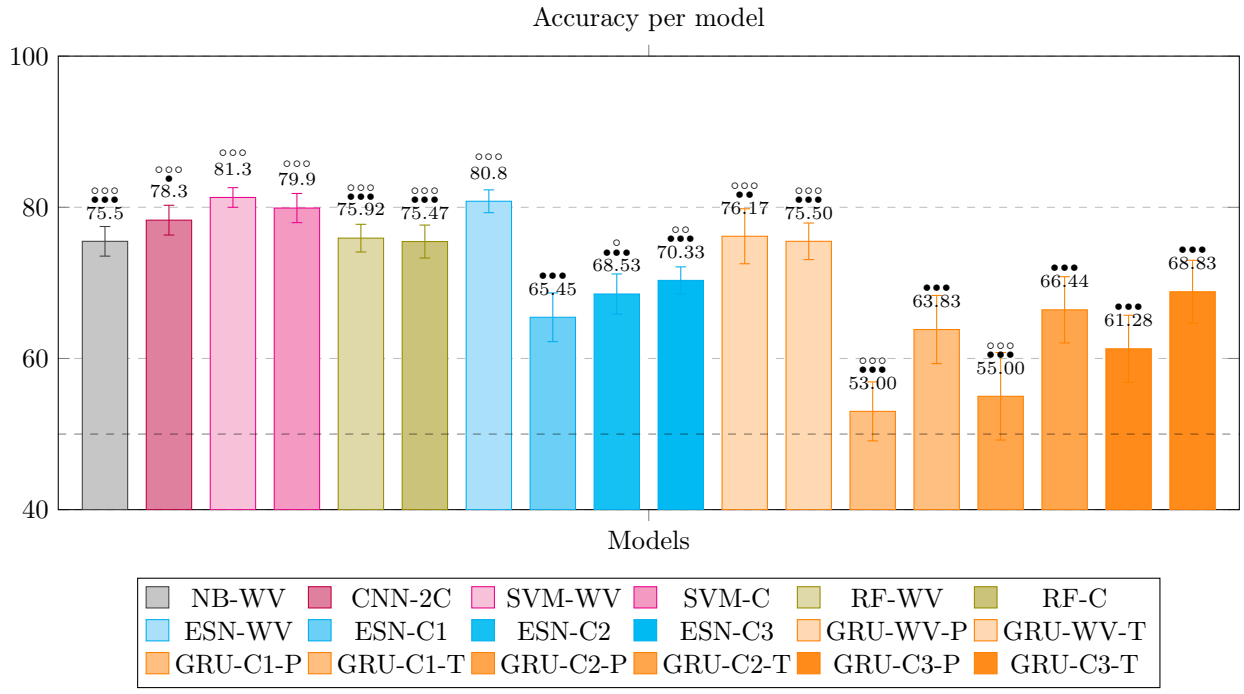


Figure 5.7 – 10-fold cross validation accuracy on PAN17 dataset for each model and features. Bullets (•) represent significant statistical differences with the model ESN-WV (• = 5%, •• = 1%, ••• = 0.1%) while circles (○) represent significant statistical differences with the model ESN-C1 (○ = 5%, ○○ = 1%, ○○○ = 0.1%).

ESNs based on pretrained character bigrams comes after with respectively 70.33%, 68.83% and 68.53%. At the bottom of the pack, we find GRUs based on trained character trigrams (GRU-C3-T), ESNs based on pretrained character unigrams (ESN-C1), GRUs based on trained character unigrams (GRU-C1-T), GRUs based on pretrained character trigrams (GRU-C3-P), pretrained character bigrams (GRU-C2-P) and unigrams (GRU-C1-P) with respectively 66.44%, 65.45%, 63.83%, 61.28%, 55% and 53%. Of all models tested, GRUs based on pretrained character features achieved the worse accuracy and two of them (GRU-C1-P, GRU-C2-P) struggle to get results significantly higher than a random classifier.

For ESN models, Figure 5.7 shows the different results achieved with these models (in blue). The best ESN model is based on pretrained word vectors (ESN-WV) and achieved an accuracy of 80.6% against 65.45%, 68.53% and 70.33% for respectively ESNs based on character unigrams (ESN-C1), character bigrams (ESN-C2) and character trigrams (ESN-C3). The difference between ESNs based on word and character features are important and significant which allow us the observe that word-based features are more effective with these models. Between character-based features, we can observe that accuracy increases with increasing n-grams going from 65.45% for ESN-C1, to 68.53% and 70.33% for ESN-C2 and ESN-C3 respectively. This increase is not significant only for C2 to C3.

Compared to other models, ESNs based on pretrained word vectors (ESN-WV) outperform all other neural and baseline models except word-based SVM with 80.6% against 81.3%. However, there is no significant difference between these two models. These models clearly outperform other neural models based on GRU with an accuracy of 80.6% for word-based ESNs against 76.17% for the best GRU model found for this task (GRU-WV-P). Even if ESNs are much faster and easier to train than GRUs.

For GRU models, the best accuracy of 76.17% is achieved by pretrained word vectors (GRU-WV-P), not very far from the second one, trained word vectors (GRU-WV-T), with an accuracy of 75.5%, and no significant differences. Regarding character-based features, trained ones achieved significantly higher accuracy with 63.83%, 66.44% and 68.83% respectively for trained character unigrams (C1), bigrams (C2) and trigrams (C3). In comparison, pretrained versions of these features obtained only 53%, 55% and 61.28%. Differences between trained and pretrained character-based features, and between n-grams being significant, we can make two observations. First, training embedding layers are effective for GRUs using character-based features, and second, that increasing n-grams from character unigrams to character trigrams is also effective to improve the performance of these models. In comparison to word-based features, characters are less effective with 68.83% for the best GRU model based on characters (GRU-C3-T) against 76.17% for the best GRU based on words (GRU-WV-P). This difference being significant, we can conclude that word-based features are also more effective with these models.

Compared to the baseline, neural models such as ESNs and GRUs can achieve competitive results with 80.6% and 76.17% respectively for the best ESN and GRU models. The ESN model outperforms all baseline models except SVMs based on BOW, while the GRU model is outperformed only by BOW based SVMs, the CNN neural model based on character bigrams (CNN-C2), and the bag-of-characters based SVMs. Between word-based and character-based features, character-based features are less effective than word-based features when the type of model is kept constant. Generally, accuracy is much more determined by the type of model used than by the type of features.

On the overall results, we can observe a first part of Table 5.3 composed of state-of-the-art baseline methods such as linear SVM and Random Forest, and RNN models using word-based features. On this three models (ESN-WV, GRU-WV-P, GRU-WV-T), ESNs based on pretrained word vectors achieved an accuracy of 80.6% with only 2,000 parameters to evaluate, while the best baseline method, an SVM based on word n-grams obtained 81.3% for 1.9 millions parameters, for a difference of only 0.7 points. The two others RNN models, GRUs based on trained and pretrained word vectors, achieved accuracy similar to state-of-the-art baseline methods such as Random Forest and Naive Bayes classifier with respectively 76.17% and 75.50% but with many more parameters to learn compared to ESNs. A second part is composed of RNN models based on character embedding trained or pretrained with accuracy going from 70.33% at the top to 53% at the bottom of the pack. In the set of RNN models we tested, only three are competitive with state-of-the-art methods in authorship analysis, ESNs and GRUs, based on word vectors. Of these three models, two of them, GRUs, have a very high number of parameters to learn, while ESN is the only one being competitive on both accuracy and complexity.

Classifier	10-CV accuracy	Std.	Parameters
SVM (linear) word 1-2 grams TF-IDF	**81.30 %**	**1.3**	**1,985,718**
ESN-WV (1,000 units)	80.60 % ‡	1.5	2,000
SVM poly-3 character 1-4 grams TF-IDF	79.92 % ‡	1.9	670,379
CNN 2-grams (CNN-C2) + starting-grams + ending-grams	78.30 % 2‡	2.0	1,158,732
GRU-WV-P (100 units)	76.17 %	3.6	120,802
Random Forest word 1-4 grams 200 trees	75.90 %	1.8	9,061,946
GRU-WV-T (25 units)	75.50 %	2.4	43,709,277
Naive Bayes classifier baseline	75.50 % 3‡	2.0	9,061,946
Random Forest character 1-4 grams 200 trees TF-IDF	75.47 % 3† 3‡	2.2	672,330
ESN-C3 (1,000 units)	70.33 %	1.8	2,000
GRU-C3-T (25 units)	68.83 %	4.2	21,442,717
ESN-C2 (1,000 units)	68.53 % ‡	2.7	2,000
GRU-C2-T (25 units)	66.44 %	4.4	776,497
ESN-C1 (1,000 units)	65.45 %	3.2	2,000
GRU-C1-T (50 units)	63.83 %	4.5	26,342
GRU-C3-P (25 units)	61.28 %	4.4	6,577
GRU-C2-P (25 units)	55.00 %	5.8	4,327
GRU-C1-P (100 units)	53.00 %	3.9	33,802
Random baseline	50.00 %	0	

Table 5.3 – 10-CV success rates of baselines and models on the PAN17 gender profiling task. ‡ indicates significant differences with the model positioned two lines below. † represents significant difference with the model on the line below.

5.7 Conclusion

In this chapter, we evaluated the performance of RNN models to perform the Author Profiling task referred as gender profiling where the model has to predict if a Twitter profile composed of 100 tweets has been written by a man or a women (binary classification). To this end, we used the dataset developed for the 2017 edition of the PAN shared task at the CLEF conference. This dataset is composed of 3600 Twitter profiles with 100 tweets per profile, all written in English.

Our goal was to evaluate two kinds of RNN models, a shallow one referred as Echo State Networks (ESNs), and Gated Recurrent Units (GRUs). We evaluated these models with a set of different textual representations based on word and character n-grams trained during the learning phase or pretrained with GloVe, for words, and according to 3.3.3 for characters. These character-based representations were pretrained on a dataset extracted from Wikipedia and composed of 200 millions samples. The pretrained model has to predict the next word from its context, the embedding layer being used afterward as an embedding matrix to transform tweets into time series.

To evaluate our models, we use 10 fold cross validation to have a faire evaluation. For GRUs models, we split a fold into a validation set and a test set (50% / 50%). We then used statistical test (t-test $\alpha = 0.05$) to extract significant differences between models. As a baseline we used a variety of state-of-the-art methods in

authorship analysis such as SVMs, Random Forest and Naive Bayes classifiers. Models tested here come from a different paradigm. While traditional methods consider textual document as a single block (bag-of-words), RNNs compute documents as a stream of text, allowing the extraction of evidence from their predictions.

Results indicate that ESNs based on pretrained word vectors are competitive (80.6%) with state-of-the-art methods such as SVMs (81.3%) on this task while keeping a very low number of parameters and a fast training time compared to common neural network models (minutes against hours). Furthermore, this model outperformed methods commonly used in authorship analysis studies such as SVMs based on character (79.92%), Random Forest (75.9%) and Naive Bayes classifiers based on the BOW paradigm (75.5%).

However, an important observation must be made here. Pretrained word vectors used here with ESNs (GloVe) have a large vocabulary but are not trained on data similar to text found in tweets and social networks. Indeed, the vocabulary used on Twitter is composed of a lot of different hashtags and smilies. As a consequence, lot of tokens found in this dataset are not part of the vocabulary of the embedding matrix used here to transform text into timeseries. It is therefore highly possible that ESNs with word vectors pretrained on data extracted from a similar source (Twitter) to this task could achieve higher performances compared to the one found in this study. In the future, we plan to investigate performances of ESN models with word vectors pretrained on social networks, specifically on data coming from Twitter.

Another main result that we want to highlight is the very slow dynamics of ESN on this task regardless of the kind of representations used as inputs. The dynamics of an ESN is specified by its leak rate and must be fine-tuned for the task at hand. It depends on statistical properties of the input timeseries and low leak rates drive the ESN to a slow dynamics and inversely for high leak rates. Here, we observe that the leak rate corresponding to the highest accuracy is found below 0.05 for all features except the character unigrams. Best accuracy is obtained with a leak rate of 0.01, 0.05, 0.05, 0.01, 0.01 for pretrained word vectors (WV), Part-Of-Speech (POS), function words (FW), pretrained character bigrams (C2) and trigrams (C3). Only one feature, pretrained character bigrams, achieved their best accuracy with a leak rate higher than 0.1 (0.4). This observation is important for further application of Reservoir Computing (RC) related methods to NLP and document classification, it is also an important observation for those who would like to apply ESNs to tasks involving natural languages.

Regarding features, pretrained words vectors clearly outperform other features when used with ESNs with a 10-CV accuracy of 80.6% against 70.2%, 71.91%, 65.45%, 68.5% and 70.33% for POS, function words (FW), character unigrams (C1), bigrams (C2) and trigrams (C3) respectively. As these representations are pretrained on data which are substantially different from those used in this task (different words, hastags, characters and smilies), further investigations should be made in order to evaluate ESNs mode with word and character vectors pretrained on data coming from similar source such as tweets and social network posts.

Regarding GRUs, only models based on trained and pretrained word vectors were able to compete with state-of-the-art models and ESNs. GRUs based on trained and pretrained word vectors achieved similar performances and no significant differences were observed. However, GRUs have also a much higher number of parameters to be learned and the training time rises from several minutes with ESNs to several hours for GRUs.

Similarly to ESNs, word-based representations are more effective with GRUs than character-based representations on this task. Also similar to the ESNs is the observations that increasing character n-grams significantly increased the accuracy. Between pretrained and trained version of GRUs, trained ones were significantly better and achieved higher accuracy. This result could be due to two possible causes which are not mutually exclusives. First, the dataset is pretty big and composed of several millions of tokens, which could be enough to train on-the-fly representations which are more efficient than the pretrained ones. The second is that the actual pretrained representations are trained on data which are substantially difference from those use on this task as the character-based word2vec model introduced in 3.3.3 is based on data extracted from Wikipedia. Textual data from this source does not contains information such as hashtags, similes, mentions and other social network related tokens. This could make the trained version more efficient by developping representations which are adapted to the task at hand.

Finally, we think that results presented in this chapter show that RNN models, especially ESNs, can significantly be competitive with state-of-the-art methods on gender profiling tasks. Furthermore, they demonstrated that a simple and shallow neural model such as ESNs, thanks to its learning algorithm based on simple linear regression, can achieve a higher accuracy than deeper and more complex models such as GRUs and character-based CNNs. ESN's higher accuracy can be explained by the recurrent architecture that allow ESN to take into account token order, and its small number of parameters, which makes it less eager for training data. As shown by Table 5.1, combination of features are effective on this task and we therefore

advice the exploration of feature such as combination of words and characters, with syntactic ones such as POS.

Chapter 6

Author Verification

In this chapter, we seek to evaluate models introduced in Chapter 3 on the two author verification tasks. In the first task referred as Author Extraction (AE), RNNs must identify the author in a sequence of tokens. The output of the recurrent model is the probability that the token at time t has been written by the evaluated author. In the second task referred as Author Verification in Multi-Authored Documents (AVMA), we seek to identify if a part written by an author exists in a target document. For both task, we used the F_1 (defined in 2.4.2) score averaged over three authors extracted from the SFGram dataset (Isaac Asimov, Robert Silverberg and Philip K. Dick). We evaluated two RNN models, namely ESNs and GRUs, with a set of features based on word embedding (WV) and character embedding with n-grams ranging from one to three (C1, C2, C3). For each feature, we evaluated the F_1 score with pretrained (-P) or trained vectors (-T). We also investigated various model parameters such as hidden layer / reservoir sizes in order to show how various models tend to overfit. Section 2.4 of Chapter 2 introduced formally the tasks and the dataset used in this chapter.

In Section 6.1, we introduce the methodology used with RNNs to compute a threshold and identify documents or parts of document containing text written by a target author. Section 6.2 compare results obtained with the different models based on ESNs. Section 6.3 analyse results obtained by different RNNs based on GRUs. Section 6.4 compares RNN architectures. Section 6.5 compare results obtained by RNNs and baseline models. Section 6.6 analyse some output examples for the three authors and show how these outputs can be used to interpret the context of the predictions. Finally, Section 6.7 compare the overall results obtained in this chapter and draw some general conclusions.

6.1 Methodology

The output of the RNN is $\hat{\boldsymbol{y}}(t)$, the estimated probability that the sequence of tokens at position t has been written by the author. The result is then an output timeseries of authorship scores. Figure 6.1 shows ESN's outputs of two issues of *IF Science Fiction* published respectively in March 1955 and in September 1959. The first contains the novel *War Veteran* and the second *Fair Game* both written by Philip K. Dick. The blue lines show the outputs at time step t (x-axis). At each time step, ESN outputs represent a score for the presence of the target author. The blue areas show the position of the section written by the true author. With ESNs, magazine issues are processed as text streams and the output is normalised to have mean and variance equal respectively to zero and one. This normalisation is necessary with ESNs as the outputs are computed with simple linear regression methods. With GRUs, we have the possibility to add a sigmoid function at the output in order to learn directly an output value between zero and 1.

To separate tokens and documents written by the target author from those with negative authorship, we have to determine an optimal threshold during the training phase. For the first task (Author Extraction), once the output is computed for the whole training dataset, we look for the best threshold which allows to separate the two classes on a validation set. Each position with an output value $\hat{y}(t)$ above the threshold was considered part of a section written by the target author. For the test phase, we computed the F_1 score based on the calibration obtained from the validation set. Blue dotted horizontal lines (present in Figure 6.1) shows the chosen thresholds while blue points represent positions predicted as belonging to the author section. The training, validation, and test sets represent respectively 80%, 10% and 10% of the whole dataset.

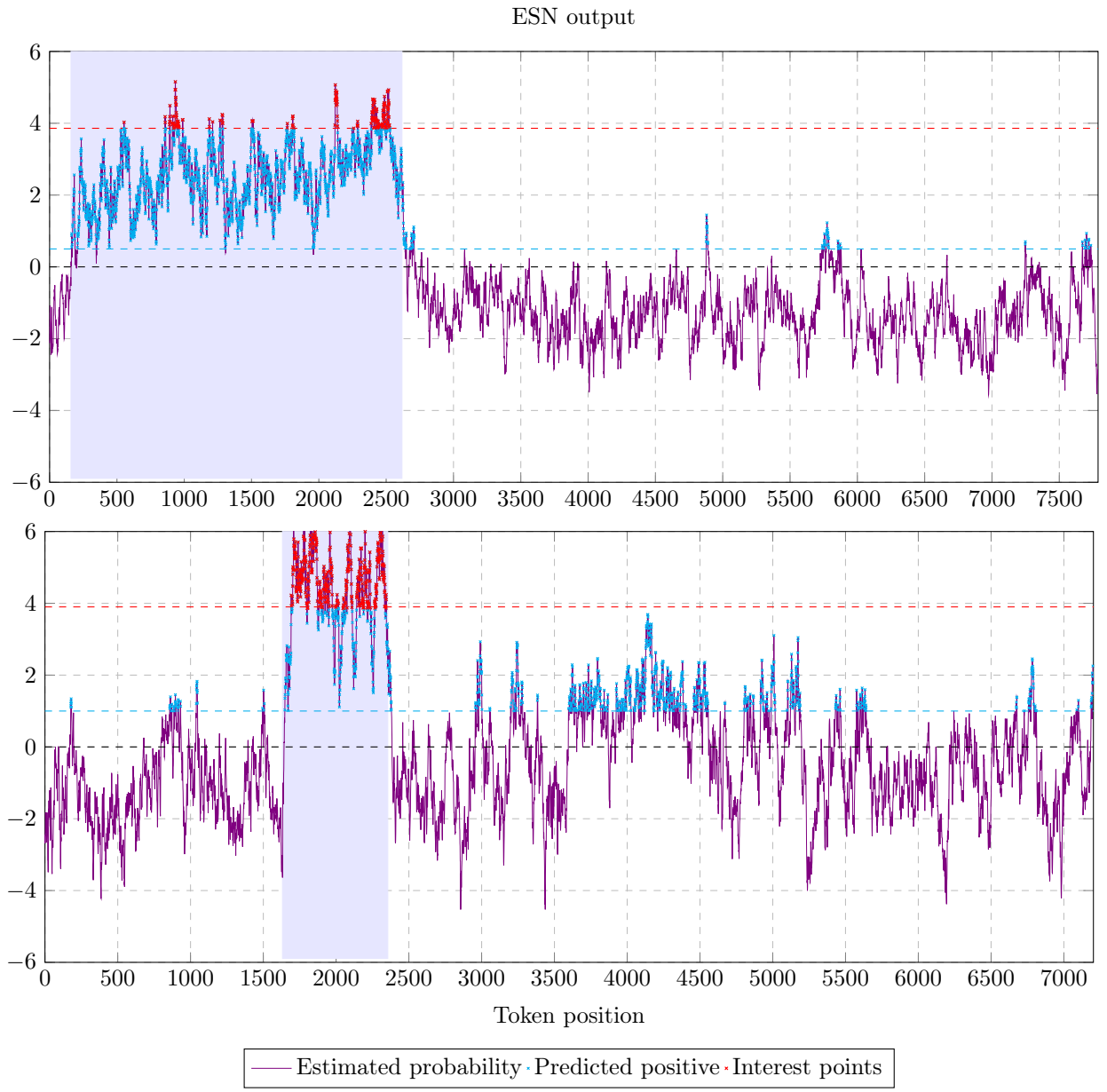


Figure 6.1 – Two examples of outputs after normalisation of an *Echo State Network* with a pretrained word embedding layer on the author verification task. The violet line shows the estimated authorship score. The blue horizontal dotted line is the threshold and points above it are marked in blue. The red points are interest points and the red dotted line is the threshold used to find them.

We used the same principle at the document level (AVMA). We define the threshold based on validation set, a threshold that best separate issues in which the target author collaborated and those which do not contain a text written by this author. In Figure 6.1, red dotted lines show thresholds used to find interest points in an issue belonging to the test set. If interest points are present, the author is considered to have collaborated on that issue.

For GRUs, we used the SGD algorithm based on Adam, with a learning rate of 0.0001. All models were trained for 50 epochs. A main point of design was the definition of the loss function as with deal with a very unbalanced dataset. Indeed, we have much more negative examples in each document than positive ones (4.25% for Asimov, 3.26% for K. Dick, and 11.25% for Silverberg). With a traditional loss function based on Mean Squared Error (MSE), the SGD algorithm can achieve very low values just by converging network outputs to zero. We then used a different loss function which allow us to tune the weight put on negative (zero) on positive (one) example.

$$loss = \alpha\mathbb{E}[\hat{Z}^2] + (1-\alpha)\mathbb{E}[(\hat{Z}-1)] \tag{6.1}$$

where α is the weight with a value between zero and one. This parameter allows to put a different importance on the negative (author not present) and positive (author present) samples to be learned. This is useful for example as both tasks do not put the same importance on this balance. For example, at the document level, we want to have high output values only if the true author is certain at this position and then a very low number of false positives. Whereas, false positives are not so important at the token level. In our experiments, we tested different values for α and we determined that this parameter has no significant importance at the token level and we then choose a value of 0.5 (same importance for negative and positive samples), but we determined that the best weight at the document level was 0.56, which is coherent with our preceding intuition.

6.2 Echo State Networks

We first wanted to analyse the performances of the shallower RNN architecture referred as Echo State Networks (ESN). To this end, we evaluated the F_1 score at the document and token levels of ESN models based on pretrained word embedding (ESN-WV) and pretrained embedding of character trigrams (ESN-C3). Results of these evaluations are shown in Figure 6.2 with results obtained with GRUs. Bullets ($\bullet$) represent statistically significant differences with the random classifier ($\bullet = 5\%$, $\bullet\bullet = 1\%$, $\bullet\bullet\bullet = 0.1\%$). Dashed black lines represent the F_1 score of the random classifier.

We evaluated the F_1 score using 5-fold cross validation (5-CV) according to the three authors, for words and character-based ESN models. To have a broader evaluation, we also computed the F_1 averaged over the three authors. The random baseline is a simple classifier predicting always yes for all points in the stream which obtained at the token level respectively an F_1 score of 0.08, 0.06, 0.20 and 0.11 for Asimov, Dick, Silverberg and the average F_1. These results depend on the amount of positive samples for each author. For example, the random classifier obtained the lowest F_1 score with Dick (0.06) as he is the author with the least amount of text present in the corpus. At the document level, the random classifier obtained an F_1 score of respectively 0.35, 0.36, 0.65 and 0.47 for Asimov, Dick, Silverberg and the average F_1. These results depend on the total number of issues available for each author. For example, the random classifier obtained the lowest F_1 score with Asimov (0.35) as he is the author with the least amount of issues containing one of his texts in the corpus.

Token level For Isaac Asimov, the best F_1 score is reached by an ESN based on pretrained vectors of character trigrams (ESN-C3) with 0.71 against 0.58 for the pretrained word embedding version (ESN-WV). This difference is significant ($\alpha = 0.1\%$). For Philip K. Dick, the best F_1 score is reached also by the character-based ESN (ESN-C3) with 0.81 against 0.73 for the pretrained word embedding version. This difference is significant ($\alpha = 5\%$). For Silverberg, the best F_1 score where obtained by the ESN based also on character trigrams (ESN-C3) with 0.65 against 0.62 for the ESN based on pretrained word embedding (ESN-WV). This difference is not significant.

Philip K. Dick seems easier to identify at token level than Asimov and Silververg for both models despite a smaller training set. The reason for this greater ease to identify Dick remains to be identified in future research. As possible explanation, we can assume that Dick tends to reuse the same stylistic construction.

To determine which model performs best, we computed the average F_1 score over the three authors. ESN-C3 surpasses the word-based ESN with an average F_1 score of 0.73 against 0.64 respectively. Differences between these two models are significant for Asimov and Dick, but not for Silverberg.

Regarding variability, the 5-CV standard deviation for ESNs based on pretrained word embedding (ESN-WV) and character trigrams (ESN-C3) is respectively 0.35 and 0.30 for Asimov, 0.12 and 0.14 for Philip K. Dick, and 0.17 and 0.22 for Silververg. As with the F_1 score, models have more stable prediction when working on Philip K. Dick in comparison with Asimov and Silverberg. Results achieved by all ESN models tested at the token-level are significantly higher than F_1 scores obtained by the random classifier.

Document level At the document level, (see bottom part of Figure 6.2), we asked models to determine whether an author's work is present in a issue or not. To compare their effectiveness, we added two models as a baseline.

The learning stage of these models is based on a training set where the label indicates the probability that the target author participated to the given issue. This probability is fixed to one if the author is present, zero otherwise. Once trained, we used the same procedure as for ESNs by determining the best threshold to separate both classes. For Isaac Asimov, the best F_1 score is achieved by a character-based ESN (ESN-C3) with 0.69 against 0.41 for word-based ESN (ESN-WV). The difference is not significant ($p = 6.38\%$). For Philip K. Dick, the best F_1 score is achieved by a character-based ESN (ESN-C3) with 0.95 against 0.89 for the pretrained word embedding ESN (ESN-WV). The difference is not significant. On Silverberg's novels, the best score is achieved by the word-based ESN (ESN-WV) with an F_1 score of 0.78 against 0.77 for the character trigrams-based ESN (ESN-C3). The difference is not significant.

Of three authors, Philip K. Dick is the easier to detect at the document-level. This is coherent with what we found at the token level. The character-based ESN even reaches an F_1 score of 0.95 Dick's novels. Regarding variability, the 5-CV standard deviation for ESNs based on pretrained word embedding (ESN-WV) and character trigrams (ESN-C3) is respectively 0.30 and 0.31 for Asimov, 0.19 and 0.02 for Philip K. Dick and 0.13 and 0.21 for Silverberg. Again, Philip K. Dick can be identified at the document-level with more stable predictions.

On Asimov's novels, only the character-based ESN is able to perform significantly better than the random classifier. On Dick, two models outperform clearly and significantly the random classifier. For Silverberg, only the word-based ESN obtained a F_1 score significantly higher than the random classifier.

6.3 Gated Recurrent Units

6.3.1 Features

We now analyse performances of the other type of RNN architecture (GRUs) with different authors and various features. We then designed an experiment whose results are presented in Figure 6.2 alongside those related to ESNs. For GRUs based on word embedding, we selected the best model across the different hidden layer sizes. For GRUs based on character embedding, we selected the best model across various n-grams (C1, C2, C3) and the different hidden layer sizes. On Figure 6.2, symbols C1, C2 and C3 shows which n-gram worked best. Bullets ($\bullet$) represent statistically significant difference with the random classifier ($\bullet = 5\%$, $\bullet\bullet = 1\%$, $\bullet\bullet\bullet = 0.1\%$). Dashed black lines represent the F_1 score of the random classifier.

Token-level Figure 6.2 (top) shows the average F_1 score obtained at the token-level by different GRU models with different features. For Asimov, the best GRU model is the one based on trained character bigrams (GRU-C2-T) with 0.46 against 0.35, 0.38 and 0.25 for GRUs based on respectively pretrained word embedding (GRU-WV-P), trained word embedding (GRU-WV-T) and pretrained character unigrams (GRU-C1-P). However, we found no significant difference between the two word-based GRUs (GRU-WV-P, GRU-WV-T). For word-based GRUs, results were significantly higher than a random classifier for only one author. For character-based GRUs, character unigrams-based GRU (GRU-C1-P) did not performed better than random for any author. Compared to ESNs, GRUs performed significantly worse with 0.46 for the best GRU against 0.71 for the best ESN.

For Dick, two GRU models outperformed the other GRU models, these models are based on trained word embedding (GRU-WV-T) and trained character trigrams (GRU-C3-T) which both obtained an average F_1 score of 0.55 against 0.42 and 0.29 for corresponding pretrained models. We found a significant difference

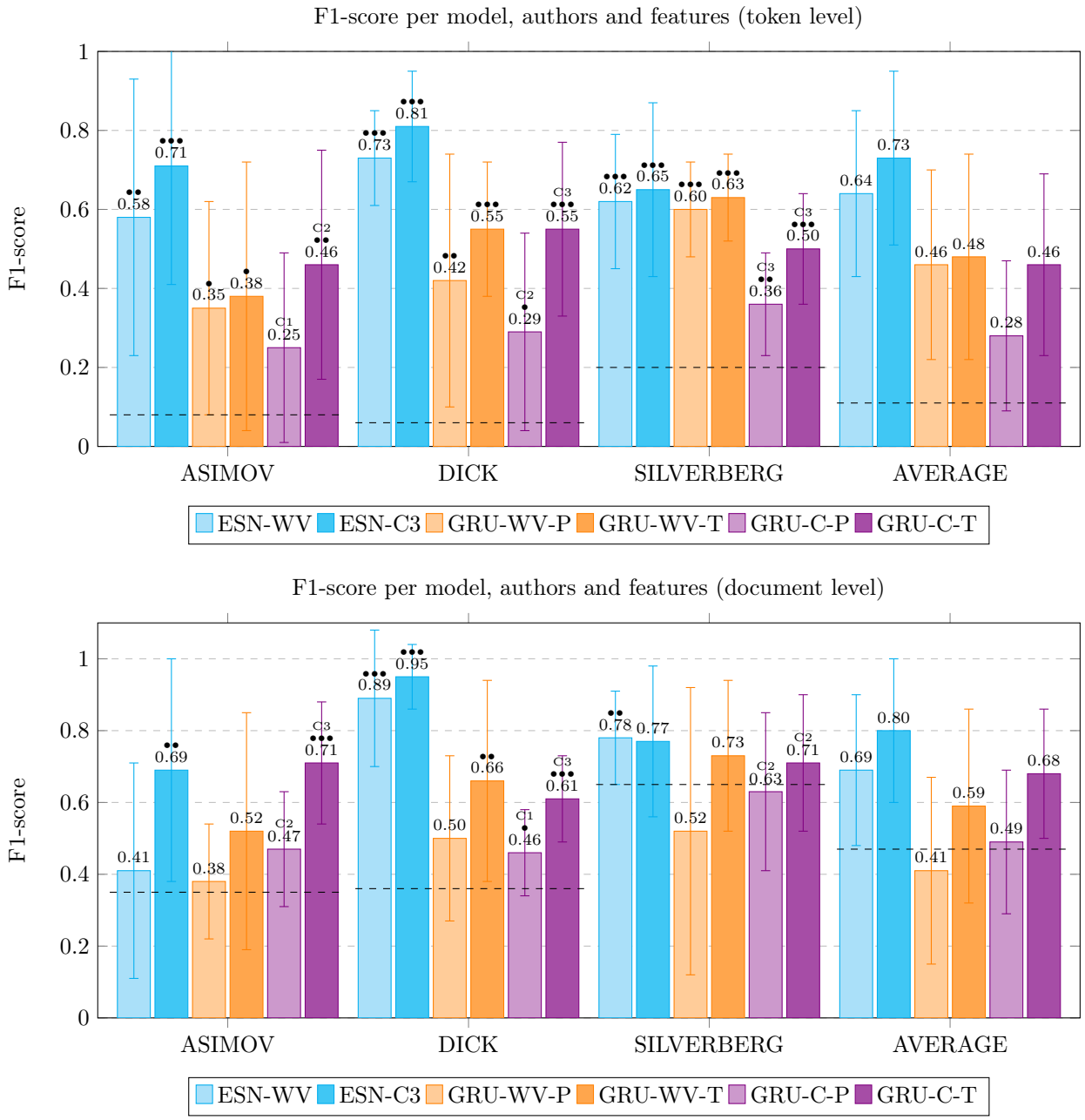


Figure 6.2 – *Top.* Best token-level 5-fold cross-validation F_1 score for each feature and RNN type on the 3 chosen author of the SFGram dataset. *Bottom.* Best document-level 5-fold cross-validation F_1-score for each feature and RNN type on the 3 chosen author of the SFGram dataset. Bullets (•) represent statistically significant differences with a random classifier (• = 5%, •• = 1%, ••• = 0.1%). Dashed black lines represent the random classifier. For character-based GRUs, we added n-grams used with the symbols C1, C2 and C3.

between the worse model (GRU-C2-P, 0.29) and three other GRU models (GRU-WV-P, GRU-WV-T and GRU-C3-T). However, we found no significant difference between these three models. Compared with ESNs, GRU performed significantly worse with 0.55 for best models against 0.81 for ESNs.

For Silverberg, the best GRU model obtained an F_1 score of 0.63 and is based on trained word vectors (GRU-WV-T), against respectively 0.60, 0.36 and 0.50 for GRU models based respectively on pretrained word vectors (GRU-WV-P), pretrained (GRU-C3-P) and trained character embedding (GRU-C3-T). There is no significant difference between pretrained and trained word-based GRUs (GRU-WT-P, GRU-WV-T), but a significant difference exists between these two models and the character-based GRUs. Compared to ESNs, character-based ESN (ESN-C3) obtained a higher F_1 score (0.65) compared to the top GRU model (GRU-WV-T). However, there is no significant difference between ESN models and the two word-based GRUs, and between the two ESN models. Futhermore, we can observe here that Silverberg is more difficult to identify at token level compared to Dick and Asimov as the best model obtained an F_1 score of 0.65 against 0.81 and 0.71 for respectively Dick and Asimov. Surprisingly, more samples are available for Silverberg.

For the F_1 score averaged over three authors, the best GRU model is based on trained word vectors with 0.48 against 0.46, 0.28 and 0.46 for respectively pretrained word vectors (GRU-WV-P), pretrained (GRU-C3-P) and trained character embedding (GRU-C3-T). Pretrained character embedding (GRU-C-P) significantly underperforms compared to other models and are even not significantly better than the random classifier on Asimov's novels. The high variability of outputs makes the analysis difficult and inconclusive for other GRU models. For Asimov, GRUs based on words and trained character embedding (GRU-WV-T, GRU-C-T) have similar F_1 scores. We can make the same observation on the results obtained with Dick's novels. However, on the Silverberg dataset, the trained trigrams-based GRU (GRU-C3-T) underperforms compared to word-based GRUs. Word-based GRUs seems to perform better than character-based GRUs but more experiments should be performed to confirm this hypothesis. ESNs obtained a higher F_1 score than GRU models on two authors (Asimov and Dick) out of the three we tested.

Document-level Bottom part of Figure 6.2 (bottom) shows the average F_1 score obtained by different GRU models with different features at the document level. For Asimov, the best GRU model is based on trained character trigrams (GRU-C3-T) with 0.71 against 0.47, 0.52 and 0.38 for GRUs based respectively on pretrained character bigrams (GRU-C2-P), trained (GRU-WV-T) and pretrained word embedding (GRU-WV-P). This GRU based on trained character trigrams (GRU-C3-T) performs significantly better than GRUs based on pretrained word embedding (GRU-WV-P) and pretrained character bigrams (GRU-C2-P). The output variability of the trained word embedding GRU is to high to confirm any significant differences with other models. Compared to ESNs, character trigrams ESN (ESN-C3) have a very high output variability, and we can only confirm a higher significant F_1 score with the pretrained word-based GRU (GRU-WV-P), the worse GRU model, but not with other models. For the pretrained word embedding ESN (ESN-WV), which has also a very high output probability, we can only confirm that it is significantly outperformed by the trained character trigrams GRUs (GRU-C3-T), the best GRU model. Compared to the random classifier, only two models performed significantly better, the pretrained character trigrams (ESN-C3) and the trained character trigrams GRU (GRU-C3-T).

For Dick, the best F_1 is obtained by a GRU model based on pretrained word vectors with 0.66 against 0.61, 0.50 and 0.46 for GRUs based respectively on trained character trigrams (GRU-C3-T), pretrained word embedding (GRU-WV-P) and pretrained character unigrams (GRU-C1-P). We found significant differences between pretrained and trained version of each features (example GRU-WV-P against GRU-WV-T). Compared to ESNs, all GRU models obtained an F_1 score significantly lower than the two ESN models (word and character-based). Compared to the random classifier, the two trained features performed significantly better. Pretrained character unigrams-based GRU is the only pretrained model identifying Dick's novels significantly better than randomly.

For Silverberg, the best GRU model is based on trained word embedding (GRU-WV-T) with 0.73 against 0.52, 0.63 and 0.71, for respectively pretrained word embedding (GRU-WV-P), pretrained (GRU-C2-P) and trained character bigrams (GRU-C2-T). We were not able to find any significant differences between GRU models. Compared to ESNs, ESN models performed significantly better than the two GRU models based on pretrained features (GRU-WV/C-P). No significant differences where found with GRUs based on trained features. Compared to the random classifier, only the ESN based on pretrained word embedding (ESN-WV) performed significantly better.

On the average F_1 score, the best model is the GRU based on trained character embedding (GRU-C-T) with 0.68 against 0.41, 0.59 and 0.49 for GRUs based respectively on pretrained word embedding (GRU-

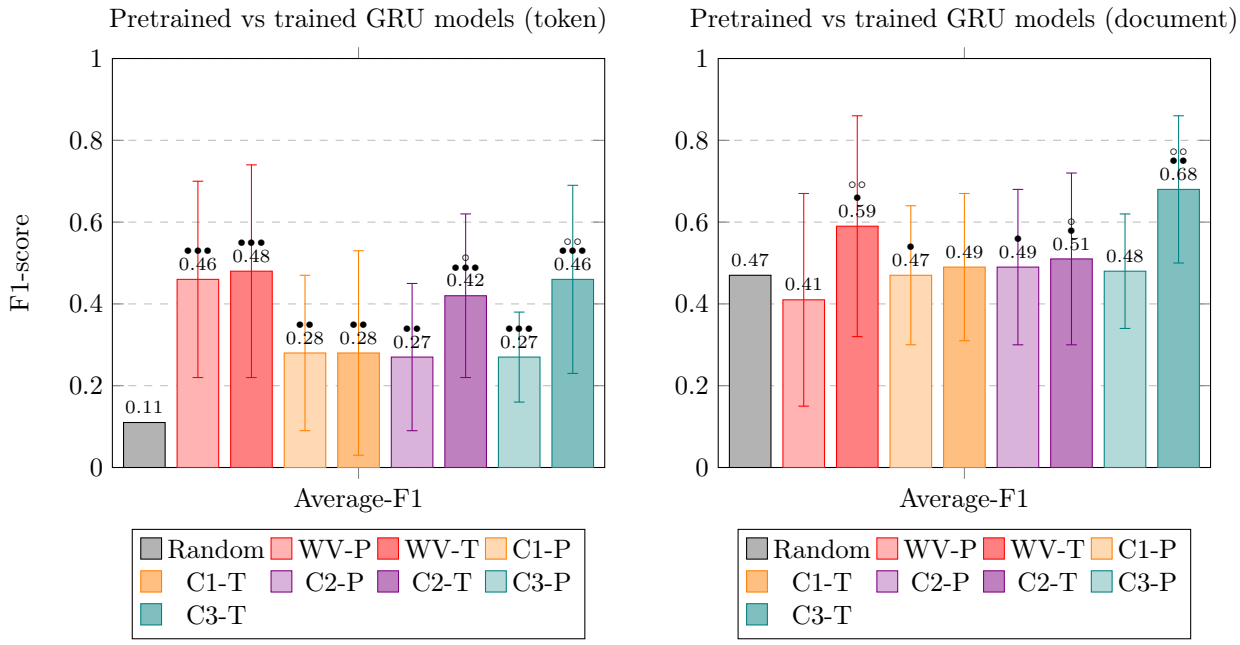


Figure 6.3 – *Left.* Best token-level 5-fold cross validation F_1-score for pretrained and trained models on the 3 chosen author of the SFGram dataset. *Right.* Best document-level 5-fold cross validation F_1-score for pretrained and trained models on the 3 chosen author of the SFGram dataset. F_1 score and standard deviation are averaged over the three authors. Bullets (•) show how many author are predicted significantly better than randomly while circles (∘) indicate significant differences with the pretrained counterpart of the feature. (∘ = 5%, ∘∘ = 1%, ∘ ∘ ∘ = 0.1%

WV-P), trained word embedding (GRU-WV-T) and pretrained character embedding (GRU-C-P). Trained features obtained higher average F_1 score compared to their pretrained counterparts. Compared to ESNs, GRU models achieved lower average F_1 score, but ESNs significantly outperformed all GRU models only on Dick's novels.

Conclusions Between the two tasks, the best GRU at the token level is the trained word vectors (GRU-WV-T) with an average F_1 of 0.48, while the best feature at the document level is the trained character embedding (GRU-C-T) with 0.68. This could indicate that the two tasks requires different kind of representation. However, the high variability makes difficult any convincing conclusions but we can observe that in general pretrained features seems to achieve significantly lower F_1 score than their trained counterparts at the token and document levels. This hypothesis is investigated in 6.3.2 below. We can also observe that ESN models achieved significantly higher F_1 scores on two authors at the token level (Asimov and Dick) and one author at the document level (Dick). This additional observation is investigated in more details in Section 6.4.

6.3.2 Pretrained versus trained features

To confront pretrained and trained features, we gathered results from the previous experiment and we plotted average F_1 scores obtained by each version for each feature (WV, C1, C2, C3). Figure 6.3 shows the result of this investigation. Word embedding (WV), character unigrams (C1), character bigrams (C2) and character trigrams (C3) are shown respectively in red, orange, violet and green. The left plot of Figure 6.3 presents F_1 scores obtained by pretrained and trained version at the token level while the right plot presents these scores at the document level. F_1 scores and standard deviations are both averaged over the three authors. For both plots, bullets (•) indicate how many authors are predicted significantly better than randomly while circles (∘) show significant differences with the pretrained counterpart of the feature.

Token-level The left part of Figure 6.3 shows average F_1 scores of pretrained and trained features at the token level. More transparent bars show scores obtained by pretrained features. At the token level, the

random baseline obtained an F_1 score of 0.11.

For word-based feature (WV), trained word vectors (GRU-WV-T) obtained the best score with 0.48 against 0.46 for pretrained word vectors (GRU-WV-P) with no significant difference. Both were able to identify the three authors significantly better than randomly. For character unigrams (C1), the pretrained (GRU-C1-P) and trained (GRU-C1-P) versions obtained the same score of 0.28, the trained version being less stable. Both pretrained and trained version of C1 where able to identify two authors significantly better than the random classifier.

For character bigrams (C2), the trained version clearly outperforms the pretrained version with 0.42 against 0.72. The difference being significant ($\alpha = 5\%$). The trained version being able to identify three authors significantly while the pretrained version was able to identify significantly only one. Furthermore, the trained version obtained an average F_1 score significantly higher than its pretrained counterpart for one of the three authors. Finally, for character trigrams (C3), the trained version (GRU-C3-T) also outperforms the pretrained one (GRU-C3-T) with 0.46 against 0.27. The difference being very significant ($\alpha = 1\%$). Indeed, the trained version achieved an average F_1 significantly higher than the pretrained version for two of the three authors. All the features show scores significantly higher than the random baseline.

Document-level The right part of Figure 6.3 shows average F_1 scores of pretrained and trained features at the document level. More transparent bars show scores obtained by pretrained features. At the document level, the random baseline obtained an F_1 score of 0.47.

For word-based feature, trained word vectors (GRU-WV-T) obtained the best score with 0.59 against 0.41 for pretrained word embedding (GRU-WV-P) with no significant difference. Only one was able to identify one of the three authors significantly better than randomly. For character unigrams (C1), the trained version (GRU-C1-T) achieved a slightly better score with 0.49 against 0.47 for the pretrained version (GRU-C1-P) but with no significant difference. Only the pretrained version was able to identify one of the author better than randomly.

We can make the same observation for character bigrams (C2) as the trained version (GRU-C2-T) obtained an average score slightly better than the pretrained version (GRU-C2-P) with 0.51 against 0.49 with a significant difference only for one author. Both version were able to identify one author better than the random classifier. For character trigrams (C3), the trained version (GRU-C3-T) clearly outperforms the pretrained version (GRU-C3-P) with 0.68 against 0.48. The difference being significant for two authors. The trained version is able to identify two authors significantly better than randomly while the pretrained version is unable to identify at least one author. All features show significant differences for at least one author with the random baseline for at least one version (pretrained, trained).

Conclusions At the token level, we can observe that trained version achieved average scores significantly higher for character bigrams (C2) and trigrams (C3). As we are dealing with very unstable outputs, we were unable to find any significant differences for other features. We found the same result at the document level as character bigrams (C2) and trigrams (C3) show average scores significantly higher than their pretrained counterparts.

6.3.3 Hidden sizes

In Equation 3.31 (3.2.10), $\boldsymbol{h}$ represents the hidden layer used by GRU to keep the necessary information for the task at hand. The size of this vector is essential as it specifies the size of gates and the complexity of the model. In this section, we explore the performance of GRUs with various hidden sizes ($n_h = 25$, $n_h = 50$, $n_h = 100$) to evaluate how the complexity of GRUs influence the F_1 score. As the computational power of GRUs increases with n_h, we expect better results with increasing n_h. However, GRUs with more hidden units can also be the victim of overfitting. To avoid this situation, we added a dropout layer before the output with a probability of 50% to put a unit output to zero. We will then be able to determine if this strategy is effective on this task and on small datasets with small number of positive examples.

To tackle this problem, we gathered results from the previous experiment where we trained GRUs with different hidden sizes and we performed statistical tests to find significant differences. The results of this analysis is presented in Figure 6.4. These figures show the F_1 score per hidden sizes for each features. The bottom and top plots show respectively the accuracy obtained at the token and document level. Bullets (•) represent how many authors got an F_1 score significantly different than the $n_h = 25$ units. Circles (∘)

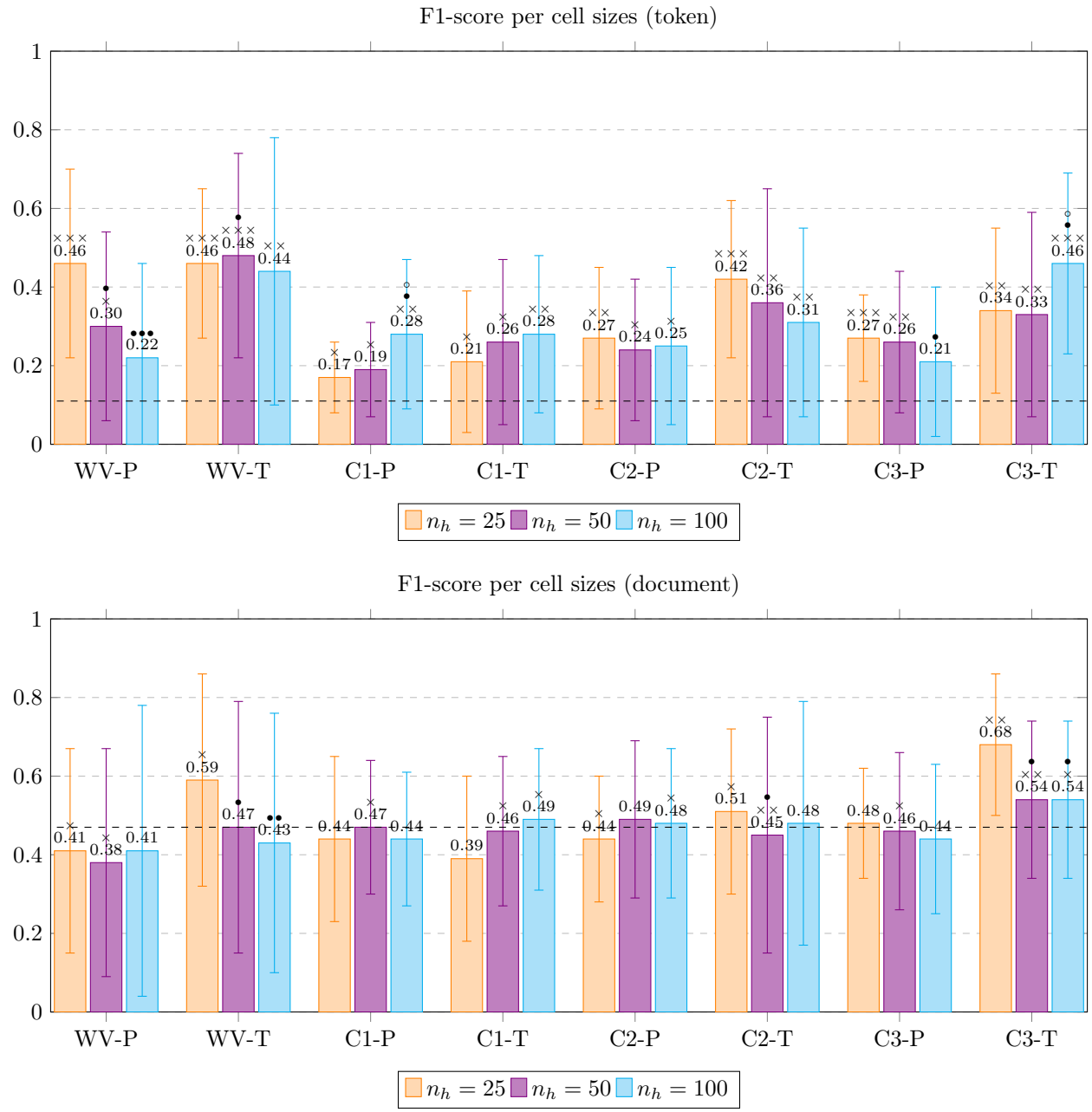


Figure 6.4 – *Top.* Best token-level 5-fold cross-validation F_1 score for each feature and RNN type on the 3 chosen authors of the SFGram dataset. *Bottom.* Best document-level 5-fold cross validation F_1 score for each feature and RNN type on the 3 chosen authors of the SFGram dataset. Crosses (×) represent how many author obtained an F_1 score significantly higher than the random classifier. Bullet (•) represent how many authors got an F_1 score significantly different than the $n_h = 25$ units. Circles (○) represent how many authors got an F_1 score significantly different than the $n_h = 50$ units.

represent how many authors got an F_1 score significantly different than the $n_h = 50$ units. Crosses ($\times$) represent how many authors got an F_1 score significantly different than the random classifier.

Token-level Figure 6.4 (top) shows the F_1 score for different cell sizes at the token level. For pretrained word vectors (GRU-WV-P), GRU with 25 units largely outperform the other models with 0.46 against 0.30 and 0.22 for GRU with 50 and 100 units. Only the GRU with 25 units achieved an F_1 score significantly higher than the random classifier for the three authors. The F_1 score significantly drops with increasing hidden sizes. We can observe in this case a strong overfitting occurring even with the addition of a dropout layer. For trained word embedding (GRU-WV-T), the three GRU models achieved respectively 0.46, 0.48 and 0.44 with no significant differences.

For pretrained character unigrams (GRU-C1-P), GRU with 100 units slightly outperforms other models with 0.28 against 0.19 and 0.17 for GRU with 25 and 50 units respectively. The F_1 score significantly increases with increasing hidden size. For trained character unigrams (GRU-C1-T), we can observe a slight improvement in F_1 score and a significant increase for the model with 100 units.

For pretrained character bigrams (GRU-C2-P), GRU with 25 units slightly outperform other models with 0.27 against 0.24 and 0.25 but with no significant differences. For trained character bigrams (GRU-C2-T), the model with 25 units slightly outperforms other model but with no significant differences.

For pretrained character trigrams (GRU-C3-P), GRU with 25 units slightly outperforms the other models with 0.27 against 0.26 and 0.21. We were unable to find any significant differences between these models. For trained character trigrams (GRU-C3-T), GRU with 100 units outperforms other models with 0.46 against 0.33 and 0.34 for 50 and 25 units respectively. The difference between the GRU model with 100 units and the two other models is significant.

Document-level Figure 6.4 (bottom) shows the F_1 score for different cell sizes at the document level. GRU models at this level show a very high output variability and most models are not able to outperform a random classifier. This make any conclusion difficult and very inconclusive. For pretrained and trained word embedding (GRU-WV), two models seems to stand out, the model with 50 and 25 units respectively, which are able to predict at least one author.

For character unigram (GRU-C1), we can observe a slight improvement with increasing hidden sizes. The average F_1 score does not change a lot with 0.44, 0.47 and 0.44 for the pretrained feature (GRU-C1-P), and 0.39, 0.46 and 0.49 for the trained one (GRU-C1-T). However, we found significant differences for one author for the pretrained feature, and for two authors for the trained feature.

For character bigrams (GRU-C2), GRU models achieved respectively an F_1 score of 0.44, 0.49, 0.48 and 0.51, 0.45, 0.48 for the pretrained (GRU-C2-P) and trained (GRU-C2-T) feature. We can observe a slight improvement for the pretrained feature with a significant improvement for one and two authors respectively for the 50 and 100 units compared to the 25 units model. For the trained feature, we can observe the opposite as we see a significant difference for the 50 units models compare to the 25 units model. Furthermore, the 100 units model is not able to perform better than randomly on any author.

For character trigrams (GRU-C3), GRU models achieved respectively an F_1 score of 0.48, 0.46, 0.44 and 0.68, 0.54, 0.54 for the pretrained (GRU-C3-P) and trained (GRU-C3-T) feature. For the pretrained feature, we can observe a slight decrease with increasing hidden sizes as the F_1 score decreases with significant difference with one authors compare to the 25 units model. For the trained feature, we can observe a clearer drop for models with more than 25 units as the F_1 score is going down with significant differences for one authors compare to the 25 units model. Furthermore, the 100 units model is able to identify only one author significantly better than randomly compared to the two other models which are able to do better than randomly for two authors.

Conclusions It is very difficult to draw any conclusions out of these experiments as we have a very high output variability or to observe a clear pattern. At the token-level, pretrained word embedding seems more sensitive to overfitting while character-based models seem to benefit from the increase of the hidden layer. At the document-level, some models get increasing F_1 with bigger hidden layer size, while other show a drop. Due to the high variability, we are unable to draw any conclusions or interesting patterns at the document-level.

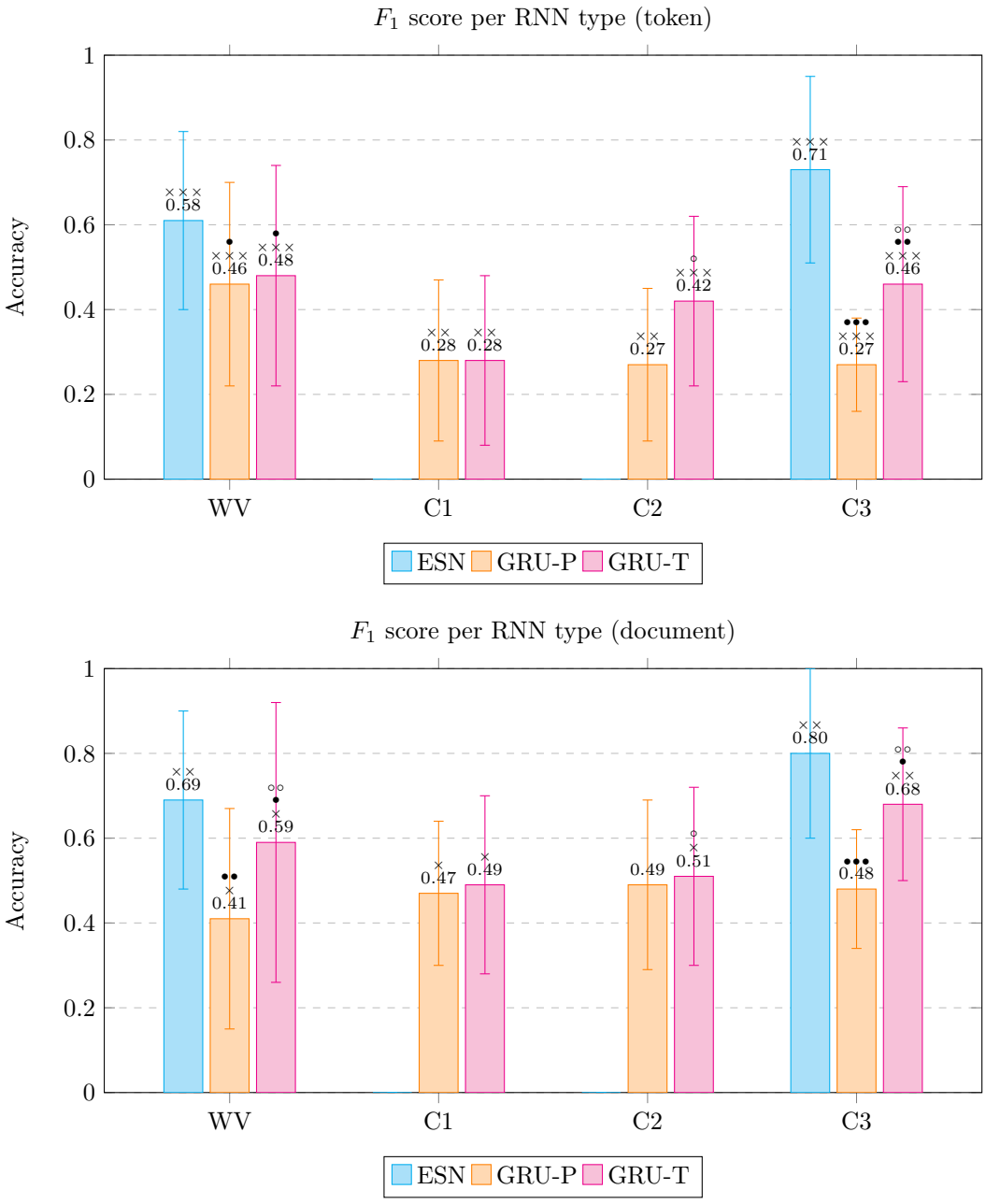


Figure 6.5 – *Top.* Average token-level 5-CV F_1-score for each RNN architecture. *Bottom.* Average document-level 5-CV F_1-score for each RNN architectures. Crosses ($\times$) represent how many authors got a F_1 score significantly higher than the random classifier. Bullets ($\bullet$) represent how many authors are significantly different with the ESN model for the same feature. Circles ($\circ$) represent how many authors are significantly different with the pretrained GRU for the same feature.

6.4 RNN architectures

Using results obtained from previous experiments, we now focus our attention on which RNN architecture is the most effective for the different features. Using 5-fold cross validation for each model, we used statistical tests to compare each model and find significant differences. The output is presented in Figure 6.5 with ESNs, pretrained GRUs and trained GRUs respectively in blue, orange and red. The top and bottom plot show respectively the resulting F_1 scores for respectively token and document levels. Crosses ($\times$) show how many authors are significantly predicted better than randomly for each model. Bullets ($\bullet$) represent how many authors are significantly different with the ESN model for the same feature. Circles ($\circ$) represent how many authors are significantly different with the pretrained GRU model for the same feature. For GRUs, we selected the model with highest average F_1 over the various hidden sizes. Here, average score is computed over the three authors (Asimov, Dick, Silverberg).

Token-level Top plot of Figure 6.5 presents the average F_1 score obtained by the different RNN architectures for the various features (WV, C1, C2, C3). For word-based models, ESNs outperform GRUs with 0.58 against 0.46 and 0.48 for respectively pretrained and trained GRUs. We found significant differences between GRU and ESN models but not between GRUs themselves. The three models were able to identify the three authors at token level significantly better than randomly.

For character unigrams (C1), pretrained and trained GRUs achieved the same average F_1 score (0.28) with no significant differences. Regarding character bigrams (C2), trained GRU outperforms the pretrained version with 0.42 against 0.27 with significant differences for two authors. Furthermore, the pretrained version were able to identify only two authors significantly better than randomly while the trained version was able to identify all author significantly better than randomly.

For character trigrams (C3), ESN clearly outperforms GRU models with 0.71 against 0.27 and 0.46 for pretrained and trained GRUs respectively. We found also significant differences with the ESN model for three and two authors respectively for the pretrained and trained GRUs.

Document-level Bottom plot of Figure 6.5 presents the average F_1 score obtained by the different RNN architectures for the various features (WV, C1, C2, C3). For word-based models, ESNs outperform GRUs with 0.69 against 0.41 and 0.59 for the pretrained and trained GRUs. We found significant differences between GRUs and ESNs for one author (both GRU models) and between the two GRU models. Furthermore, ESN model identify two authors significantly better than randomly while the two GRU models were able to identify significantly zero and one author respectively.

For character unigrams (C1), pretrained and trained GRUs achieved respectively an average F_1 score of 0.47 and 0.49 with a significant difference in favor of the trained GRU model for one author. For character bigrams (C2), trained GRU outperforms the pretrained version with 0.51 against 0.49 with a significant difference for one author. Furthermore, the trained GRU was able to identify one author significantly while the pretrained version was unable to identify correctly any author.

For character trigrams (C3), ESNs outperform GRU models with an average F_1 score of 0.80 against 0.48 and 0.68 respectively for pretrained and trained GRU models. We found significant differences for two and three authors respectively between ESNs and these two models. Furthermore, the ESN model was able to significantly identify two authors while the pretrained GRU was unable to identify correctly any author.

Conclusions Results presented in this section show that ESN are able to outperform GRU models with word-based and character-based features at the token and document levels. We presented strong evidences that these differences are significant in favor of ESNs. One hypothesis that can explain these results is the very small number of parameters of ESNs which makes them suitable for small datasets with few positive samples and allows them to avoid overfitting effectively.

6.4.1 Overfitting and losses

To observe how different features and models behave during the training phase, Figure 6.6 plots training and validation losses per epoch for each trained and pretrained feature at the token level. All plots come from the same author (Asimov), with the same cell size (25).

For pretrained character unigram (GRU-C1-P) (orange / circle), both training and validation losses start around 0.25 and goes respectively to 0.23677 and 0.24969 (0.01292). The lowest validation loss (0.24695) is

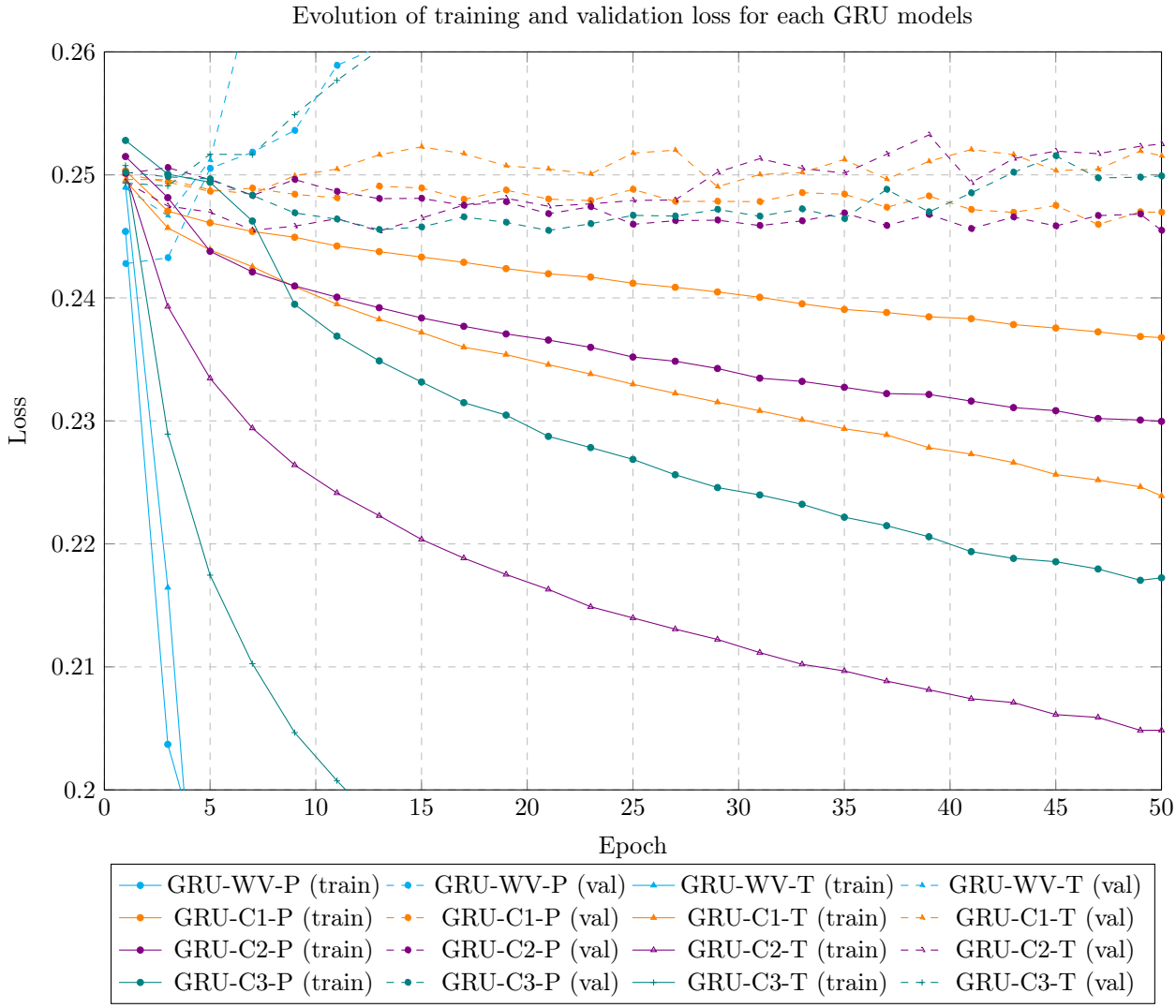


Figure 6.6 – Training and validation losses for each GRU models per authors.

achieved after 43 epoch. We can observe a slight overfitting with this feature and a very low improvement in the validation loss (-0.003) probably due to the number of parameters (few thousands) and the difficulty to estimate the gradient properly with so few positive examples. For trained character unigram (GRU-C1-T) (orange / triangle), both training and validation losses start round 0.249 and goes respectively to 0.224 and 0.251 (0.027) at the end of epoch 50. The lowest validation loss (0.24845) is achieved after only 7 epochs. The overfitting is much stronger here compared to the pretrained character unigram. This difference is probably related to the greater number of parameters of trained models.

For pretrained character bigrams (GRU-C2-P) (violet / circle), both training and validation losses start around 0.25 and goes respectively to 0.229 and 0.245 (0.016). The lowest validation loss (0.24563) is achieved after 41 epochs. We can observe a slight overfitting with this feature and a slight improvement in the validation loss (0.004). For trained character bigram (GRU-C2-T) (violet / triangles), both training and validation losses start at 0.25 and goes respectively to 0.205 and 0.252 at the end of epoch 50. This feature shows a better improvement in the training loss but a very low improvement in the validation loss (0.004) which is a sign of strong overfitting. The overfitting is also much strong here for the trained feature compare to pretrained character bigrams. As with character unigram, this difference is probably related to the much greater number of parameters of trained models.

For pretrained character trigrams (GRU-C3-P) (green / cicle), both training and validation start again around 0.25 and achieve respectively 0.217 and 0.249 (difference of 0.032). The lowest validation loss (0.24548) is achieved after 21 epochs. We can observe a slight overfitting with this feature and a slight improvement in

the validation loss (0.005). For trained character (GRU-C3-T) trigram (green / triangles), both training and validation losses start at 0.25 and goes respectively to 0.164 and 0.295. The lowest validation loss (0.2491) is achieved after only 3 epochs. This feature face a very strong overfitting as the training loss is going down fast and the validation loss shows no improvement and start rising after only 3 epochs. The overfitting is much stronger compared to the same pretrained feature. As with C1 and C2, this difference can be explained by the much greater number of parameters.

For pretrained word vectors (GRU-WV-P) (blue / circle), both training and validation losses start around 0.243 and go respectively to 0.131 and 0.283 (difference of 0.153). The lowest validation loss (0.24278) is achieved after the first epoch which shows a very strong overfitting with this features. After the first epoch, the validation loss greatly increase going up to 0.284. For trained word vectors (GRU-WP-T) (blue / triangle), both training and validation losses start around 0.249 and achieve respectively 0.048 and 0.383 after 50 epochs (difference of 0.335). The lowest validation loss (0.245) is achieved after 3 epochs which indicates also a very strong overfitting for this feature. The overfitting is much stronger compared to the same pretrained feature. As with character-based features, this difference can be explained by the much greater number of parameters.

With Figure 6.6, we can observe that a strong overfitting is at work with both trained and pretrained features. The difference greatly increases as we go from pretrained character unigrams (GRU-C1-P) to pretrained character trigrams (GRU-C3-P). For character unigrams (GRU-C1-P), the difference between training and validation loss at epoch 50 is 0.01019 while this difference for pretrained character trigrams (GRU-C3-P) is 0.03267. The gap between training and validation losses is even more important between trained and pretrained features. For pretrained character unigram (GRU-C1-P), the difference is 0.01019 while this difference for trained character unigrams (GRU-C1-T) is 0.02765. For pretrained character trigrams (GRU-C3-P), the difference is 0.03267 while this difference is 0.13103 for trained character trigrams (GRU-C3-T). For pretrained word vectors (GRU-WV-P), the difference is 0.153 while this difference for trained word vectors (GRU-WV-T) is 0.335.

As we increase character n-grams, we can observe more overfitting with trained features due to the increasing number of parameters. Overfitting is amplified with word-based features as the number of parameters increases even more. Indeed, the number of parameters with trained features is dependent on the vocabulary size which increase exponentially with increasing n-grams. For word-based features, vocabulary size is much larger compared to character-based features, the impact of this property is a stronger overfitting for both trained and pretrained versions.

6.5 Comparisons

To compare the various models we evaluated, we selected the best GRU model for each cell sizes at the token and document level. We kept the two ESN models, the word-based (ESN-WV) and character-based (ESN-C3), to which we added the random classifier. Selected models are presented in Table 6.1 (token-level) and 6.2 (document-level) which present the F_1 score for each author, the average F_1 score and the standard deviation. The last column shows the number of parameters to be learned by each model. We searched for significant differences between these models and symbols represent a significant difference with the model above in terms of F_1 score (†, ‡ and ΙΙ for respectively $\alpha = 5\%, \alpha = 1\%$ and $\alpha = 0.1\%$.

6.5.1 Token level

For Asimov, the best result is achieved by an ESN based on pretrained character trigrams (ESN-C3), 1,000 units, with an F_1 score of 0.71. The second best result is achieved also by an ESN but based on pretrained word vectors (ESN-WV) with an F_1 score of 0.58. Both are largely above the random classifier baseline (0.08). For GRU models, the best result is achieved by a model based on character bigrams (GRU-C2-T), 25 units, with an F_1 score of 0.46. A score significantly lower than the character-based ESN (ESN-C3).

The second best GRU model is based on trained word vectors (GRU-WV-T) also with 50 units, achieving an F_1 score of 0.38, not significantly different than the preceding model. All GRU models achieved results significantly higher than the random classifier except those based on character unigrams (GRU-C1). All GRU models are largely outperformed by the character-based ESN.

For Dick, the best result is also achieved by ESNs based on pretrained character trigrams (ESN-C3) with an F_1 score of 0.81 against 0.71 for the word-based ESN, the second best model. This result is largely and significantly higher than the random baseline (0.06). For GRU models, the best result is achieved by two

Classifier	**Asimov**	**Dick**	**Silverberg**	**Average F1**	**Parameters**
True	*0.08*	*0.06*	*0.20*	*0.11*	
ESN-WV 1,000 units	**0.58** ‡	**0.73** ΙΙ	**0.62** ΙΙ	**0.64** ±0.21	1,000
ESN-C3 1,000 units	**0.71** ΙΙ	**0.81** ΙΙ	**0.65** ΙΙ	**0.73** ±0.20	1,000
GRU-WV-P 25 units	0.35 †	0.42 ‡	**0.60** ΙΙ	0.46 ±0.24	120,701
GRU-WV-T 50 units	0.38 †	0.55 ΙΙ	0.53 ‡	**0.48** ±0.26	13,259,951
GRU-C1-P 100 units	0.25	0.25 †	0.35 ‡	0.28 ±0.19	9,351
GRU-C1-T 100 units	0.21	0.24 ‡	0.40 †	0.28 ±0.20	34,931
GRU-C2-P 25 units	0.21 †	0.29 †	0.29	0.27 ±0.18	12,351
GRU-C2-T 25 units	**0.46** ‡	0.36 ΙΙ	0.44 ‡	0.42 ±0.20	236,441
GRU-C3-P 25 units	0.23 ‡	0.21 ΙΙ	0.36 ‡	0.27 ±0.11	6,551
GRU-C3-T 100 units	0.33 †	**0.55** ΙΙ	0.50 ΙΙ	0.46 ±0.23	4,586,051
GRU best per author	0.46 ‡	0.55 ΙΙ	**0.63** ΙΙ	**0.55**	

Table 6.1 – Comparison of F_1 scores (5-fold CV) of RNN models on the task of Author Extraction (AE) with the SFGram dataset. Symbols represent a significant differences with the random classifier F_1 score (†, ‡ and ΙΙ for respectively $\alpha = 5\%, \alpha = 1\%$ and $\alpha = 0.1\%$.

GRU models based on trained character trigrams (GRU-C3-T) with 100 units and trained word embedding (GRU-WV-T) with 50 units with an F_1 score of 0.55.

All results obtained by GRU models are significantly higher than the random baseline but also significantly lower than the two ESN models. Compared with Asimov, Dick seems also easier to identify for ESN and slightly with GRU models despite the smaller dataset.

For Silverberg, the best result is achieved again by the ESN model based on character trigrams (ESN-C3) with an F_1 score of 0.65. The second and third best results are obtained respectively by the ESN based on pretrained word embedding (ESN-WV) and GRU based on pretrained word vectors (GRU-WV-P) with 0.62 and 0.60 respectively. However, scores between these three models are not significantly different.

Compared to Asimov and Dick, GRUs are able to achieve scores similar to ESN on Silverberg's novels. This can be explained by the fact that the dataset as much more positive examples for Silverberg and GRUs need larger datasets to be efficient. It is therefore not surprising that GRU have higher score with Silverberg. All models except GRUs based on pretrained character bigrams (GRU-C2-T) with 25 units achieved scores significantly higher than the random baseline.

On average, the best overal result is achieved by an ESN model based on pretrained character trigrams (ESN-C3), 1,000 units, with an average F_1 score of 0.73 against 0.64 for the second best result, the ESN based on pretrained word vectors (ESN-WV). These results are significantly higher than the random baseline (0.11). The best GRU model achieved an average F_1 score of 0.48 and is based on trained word vectors (GRU-WV-T) with a cell of 50 units. After these models, two GRU models based on pretrained word vectors (GRU-WV-P) and trained character trigrams (GRU-C3-T) with an average F_1 score of 0.46.

As different features seems to achieve different scores on different authors with GRUs, we can image using different GRU models for each author. The last line shows the best GRU results for each author and the corresponding average F_1. With this specific model, GRUs can achieve an average F_1 score of 0.55 against 0.73 and 0.64 respectively for ESN models. These ESN models achieved an average F_1 score significantly higher than score achieved by GRU models for two authors.

The last column shows the number of parameters to be learned by each model. The random baseline has no parameter to learn while ESN models has as many parameters to learn that there is units in the reservoir. The number of parameters of an ESN is the size of the output matrix $\boldsymbol{W}^{out}$ which is the number of units in the reservoir multiplied the number of outputs. We have here only one output ($\hat{y}$) which make ESN models very easy to train.

Classifier	Asimov	Dick	Silverberg	Avergage F1	Parameters
True	0.35	0.36	0.65	0.47	
Linear SVR word 3gram	0.35	0.36	0.60	0.44 ±0.24	∼5 millions
ESN-WV 1,000 units	0.41	**0.89** ‡‡	**0.78** ‡	**0.69** ±0.21	1,000
ESN-C3 1,000 units	**0.69** ‡	**0.95** ‡‡	**0.77**	**0.80** ±0.20	1,000
GRU-WV-P 100 units	0.31	0.41	0.52	0.41 ±0.37	39,871,901
GRU-WV-T 25 units	0.37	**0.66** ‡	**0.73**	0.59 ±0.27	13,259,951
GRU-C1-P 50 units	0.39	0.46 †	0.56	0.47 ±0.17	10,581
GRU-C1-T 100 units	0.43	0.42 †	0.61	0.49 ±0.18	34,931
GRU-C2-P 50 units	**0.47**	0.40	0.61	0.49 ±0.20	244,491
GRU-C2-T 25 units	0.35	0.45 †	0.71	0.51 ±0.21	236,441
GRU-C3-P 25 units	0.41	0.41	0.61	0.48 ±0.14	4,586,051
GRU-C3-T 25 units	**0.71** ‡‡	0.61 ‡‡	0.70	**0.68** ±0.18	4,586,051
GRU best per author	0.71	0.66	0.73	0.68	

Table 6.2 – Comparison of F_1 scores (10-fold CV) of RNN models on the task of Author Verification in Multi-Authored Documents (AVMA) using interest points. Symbols represent significant differences with the random classifier F_1 score (†, ‡ and ‡‡ for respectively $\alpha = 5\%, \alpha = 1\%$ and $\alpha = 0.1\%$.

6.5.2 Document level

Table 6.2 shows results obtained with different features and cell sizes for the three authors at the document level. Similarly to the first task, we added a random classifier as a baseline in addition to a linear Support Vector Machine Regression model (SVR) based on word trigrams. This model is implemented with Sklearn and trained with the same method as other models. This model is based on the BOW paradigm and trained to predict the probability of having a section of the document written by a given author.

For Asimov, the best result is achieved by a GRU based on trained character trigrams (GRU-C3-T), 25 units, with an F_1 score of 0.71. The second best results are obtained by an ESN model based on pretrained character trigrams (ESN-C3) with a F_1 score of 0.69. These two models achieved an F_1 score significantly higher than the random classifier but with no significant difference between them. A GRU model based on pretrained character bigrams with 50 units achieved the next score with 0.47 but with no significant difference with a random classifier. Other GRU models and the word-based obtained mixed results from 0.43 to 0.31.

ESN based on pretrained word vectors (ESN-WV) achieved a poor score with 0.41, not significantly higher than the random baseline. This result is surprising and would required more investigation in order to determine why such models are not able to identify Asimov at the document level while being efficient at the token level. SVM baseline got a F_1 score of 0.35 which is not significantly different than the random baseline. On this author, only character trigrams-based ESNs and GRUs were able to identify parts of text written by Asimov correctly in a document.

For Philip K. Dick, the best score is achieved by an ESN model based on pretrained character trigrams (ESN-C3) with 0.95. The second and the third best results are achieved respectively by an ESN model based on pretrained word vectors (ESN-WV) and a GRU model also based on trained word vectors (GRU-WV-T) with 0.89 and 0.66 respectively. For both ESNs and GRUs, these results are much higher compare to Asimov and Silverberg despite the lower number of positive examples available for training.

The other GRU models achieve mixed results going from 0.61 to 0.40. GRU models based on pretrained word embedding (GRU-WV-T), pretrained character bigrams (GRU-C2-P) and pretrained character trigrams (GRU-C3-P) where not able to do better than the random classifier. The SVM baseline got an F_1 of 0.36, not significantly higher than the random classifier. Also on this author, only ESN and GRU models were able to identify section of text written by Dick inside a document.

For Silverberg, best results are obtained by two ESNs based respectively on pretrained word vectors

Author	**Examples**
Isaac Asimov	The Hard One to whom Tritt had spoken was agreeing - the other still exuded concern. Dua was looking at Tritt. The first Hard One said, "Where is the food-ball now, Tritt?" Tritt showed them. It was hidden effectively and the connections were clumsy but serviceable. *The Gods Themselves*, March 1972.
Philip K. Dick	Miller straightened his collar and bright hand-painted necktie. He smoothed down his blue pinstripe coat, expertly lit a pipeful of two-century-old tobacco, and returned to his spools. *Exhibit Piece*, August 1954.
Robert Silverberg	All about me were the things of the gods, and 1 failed to detect the divine presence. Perhaps Schweiz had found the godhood through the souls of other men, but I, dabbling in selfbaring, somehow had lost that other faith and it did not matter to me. *A Time of Changes*, May 1971.

Table 6.3 – Example of a text classified as very probable for each author.

(ESN-WV) and pretrained character trigrams (ESN-C3) with an F_1 score of 0.78 and 0.77. The third model is a GRU based on trained word embedding (GRU-WV-T) with 25 units with an F_1 score of 0.73. The fourth and fifth best results are obtained by GRU models based respectively on trained character trigrams (GRU-C2-T) and trained character trigrams (GRU-C3-T) with F_1 scores of 0.71 and 0.70. We found no significant differences between these last four models. The word-based ESN is the only model able to perform significantly better than the random classifier on Silverberg's novels. The SVM baseline stay behind with an F_1 score of 0.60 which is not significantly different than the random baseline.

On average, the best average result is achieved by an ESN model based on pretrained character trigrams (ESN-C3), 1,000 units, with an average F_1 score of 0.80. This model also achieved the best result at the token level. The second best model is the ESN based on pretrained word embedding (ESN-WV) with an average F_1 of 0.69. A GRU model composed of specific models for each author achieved the third highest score with an average F_1 score of 0.68. On average, the ESN based on trained word vectors (ESN-WV) is largely outperformed by the same model based on character trigrams (ESN-C3). Between authors, Dick is again the easier to identify at the document level with a top F_1 score of 0.95. The second and third are respectively Silverberg and Asimov with a top score of 0.78 and 0.69. On average, the SVM baseline is not able to achieve an F_1 score significantly higher than a random classifier. This baseline is largely and significantly outperformed by neural models tested in this study.

The last column shows the number of parameters to be learned. The random baseline has no parameter to learn while ESN models have as many parameters to learn that there is units in the reservoir.

6.6 Output examples

This ease to identify Philip K. Dick is an interesting question for future research. Does this author have a particular single style or a specific vocabulary? In order to have a look at how the model deals with different authors, we extracted from a trained ESN a piece of the text of each author with a very high authorship score ($\hat{\boldsymbol{y}}$). The resulting is shown in Table 6.3. For Isaac Asimov, the text is extracted from the novel "*The Gods Themselves*" published in the issue of March 1972 of *Galaxy Magazine*. The novel depicts the story of a scientist finding out that a sample of tungsten has been transformed into plutonium 186. This leads to the development of a endless and clean source of energy. The novel is strewn with technological and scientific terms as is often the case in Asimov's novels.

For Philip K. Dick, the text is extracted from the novel *Exhibit Piece* published in the issue of August 1954 in *IF Magazine*. The story told in this novel addresses a common theme in Dick's subsequent works: the concept of shifting realities and time travel. For Robert Silverberg, the text is extracted from the novel *A Time of Changes* of the issue of May 1971 of *IF Magazine*. It won the Nebula Award that year, the equivalent of the Emmy Awards in science-fiction. It tells the story of a world where words such as *me* and *I* are forbidden and is written from an autobiographical point of view. The hero is telling his own story.

6.7 Conclusions

In this chapter, we investigated the effectiveness of different RNN models, such as ESNs and GRUs, to perform authorship verification in very noisy text streams at token and document level. The corpus used for these evaluations was extracted from science-fiction magazines published during 1952 and 1974 and is composed of 91 issues. We selected three well-known science-fiction authors, Isaac Asimov, Philip K. Dick and Robert Silverberg, and tested word and character-based features with F_1 score as the evaluation measure. Silverberg has the highest number of positive examples with 911 thousand words compared to 340 and 264 thousand for Asimov and Dick respectively for a total of 8 millions of words.

At the token level, we used a validation set to determine the best threshold to separate tokens written by the target authors from negative examples. We used the same methodology at the document level to compute a threshold to separate documents containing text written by the target author or not. We used 80%, 10% and 10% of the whole dataset as respectively a training, a validation and a test set.

We showed (Figure 6.1 and Table 6.3) that the output of RNN models can be used to extract evidences out of their predictions. The output of many-to-many RNNs is a timeserie of authorship score which can be used to extract representative texts and tokens for each author.

To evaluate the score of RNN models, we designed a main experiment in which we tested different RNN models such as ESNs and GRUs with different parameters (hidden sizes, trained/pretrained features). We did these tests with different features such as word embedding (WV), character unigrams (C1), character bigrams (C2) and character trigrams (C3). We then used statistical tests to find significant differences between models and with the random baseline classifier. The output of this experiment showed very high variability which make difficult to extract interesting conclusions and to draw clear patterns.

However, we selected (Figure 6.2) the best F_1 score achieved by GRU models for each features across the different hidden sizes. Our analysis showed that character trigrams-based ESN (ESN-C3) outperforms word-based ESN (ESN-WV) on two authors and on the F_1 score averaged over the three authors at the token level. However, we found no differences between these two models at the document level. This first approach also showed that Philip K. Dick is the easier author to identify with a maximum F_1 score of 0.81 (token) and 0.95 against 0.71/0.71 and 0.65/0.78 for respectively Asimov and Silverberg. From this data, we draw two hypothesis, the first claiming that ESN obtained significant higher F_1 scores than GRUs. The second claiming that for GRUs trained features perform better than pretrained features.

We evaluated the first hypothesis in Section 6.4 and Figure 6.5 where we selected the best GRU model for each features (WV, C1, C2 and C3) and type of training (pretrained/trained) across the different hidden sizes. We compared these models to F_1 scores obtained by ESNs at the token and document level. Our study showed that ESNs are able to perform significantly better than GRUs with word and character-based features with multiple authors. These results showed strong evidences in favor of ESNs on the two tasks.

The second hypothesis was evaluated in Section 6.3.2 and Figure 6.3 where we used data from our main experience and selected F_1 scores for various GRU models for each pretrained and trained version of each features. Our evaluations showed that at the token level the trained versions for character bigrams (C2) and trigrams (C3) performed significantly better than their pretrained counterparts. We were able to do the same observation at the document level.

In Section 6.3.3 and Figure 6.4, we performed an additional analysis seeking to determine whether the hidden layer size influence performances. We then used data from our main experience and separated the evaluation by hidden sizes for each features. As F_1 scores show high variability, we were unable to draw clear and very significant patterns. At the token level, our results showed that pretrained word embedding features seem more sensitive to overfitting while character-based features seem to benefit from increasing hidden size. At the document level, we were unable to draw any significant conclusions or interesting patterns. Our analysis was confirmed by the analysis of the training and validation losses in Section 6.4.1 and Figure 6.6 which showed a very fast increases of the validation loss and an equally rapid decline of the training loss for word-based features.

Various questions stay without a clear and definitive answer. First, an evaluation of RNNs on bigger sets of authors is still necessary. Especially those using pseudonyms to determine if models are able to undercover pseudonymous work and whether they are reliable with a higher number of possible authors. For example, the famous science-fiction author Randall Garrett is known to have used a lot of pseudonyms to publish his science-fiction novels. Using this kind of textual data, we would like to evaluate the possibility to identify interest points in novels published under a pseudonym. These kind of method could prove the collaboration of an author.

The SFGram dataset fits well the use of this kind of model based on deep learning as it contains thousands of documents and tens of millions of words. However, models based on the deep learning paradigm achieved mixed results with high variation between the different parameters (cell size, feature, pretraining/training). This makes difficult to draw any conclusions about which feature are most efficient or not. Furthermore, the number of parameters are much higher than ESN models which make the training time much longer, from tens of minutes for ESNs to several hours for GRU models. This long training time make very difficult to test different parameters, architectures and hyperparameters while ESNs are pretty simple and fast to implement. As a result, our result cannot show that GRUs or deep learning models cannot achieve better results than ESNs or classical methods as there is an infinite number of different models and parameters. However deep models we tested so far on this dataset where not able to achieve the same ratio of performance and training time.

In the future, it would be interesting to test deeper models such as stacked-ESN and bidirectional RNNs. We could then use the work presented in this study as a baseline for further investigation. Other investigations are possible on the use of other text representations such as POS tags, or sequence of POS tags. It would be also interesting to investigate features based on different noun or verb phrases (e.g., adverbial phrase of time, phase of manner, purpose phrase, etc.). These information could be used in combination to the lexical features as used in this study. We think that results presented in this chapter shows that RNNs are interesting models to handle streams of textual data. Consequently, it would be interesting to use these models to detect events in text streams coming from social networks.

Part III

Conclusions

Chapter 7

Conclusions and future work

In this study, we evaluated various kind of RNN models on three tasks related to the field of authorship analysis. As possible architectures, we selected one architecture referred as Echo State Networks (ESN) based on the Reservoir Computing (RC) paradigm, and two based on the deep learning paradigm, the Long Short-Term Memory (LSTM) networks and its simplified version, the Gated Recurrent Units (GRU). We kept our evaluation focused on many-to-many architectures that are able to provide some evidences for their predictions. Furthermore, for each model, we selected a set of important parameters in order to determine their respective impact on model's performances. Finally, we used statistical tests to find significant differences between models, baselines and random classifier.

The first task was evaluated in Chapter 4 where we investigated the performance of ESNs, LSTMs and GRUs on two authorship attribution problems using the Reuters C50 dataset. This dataset is commonly used in authorship attribution evaluations and is composed of 50 authors with 100 documents per author. Documents were selected in order to minimise the topic factor between texts. From this dataset, we selected 15 random authors to reduce the computational complexity.

For the first problem, models were asked to determine who is the author of a document in a set of candidate authors. The second problem was to determine who is the author of a sentence in two-authored documents in a set of candidate authors. All models must predict the correct author and their performance is the evaluated accuracy. In order to test various models, we designed a set of experiments to evaluate their performances and determine how various parameters influence their accuracy rates. We based our evaluations on 10-fold cross validation (10-CV) and statistical tests to detect significant differences. We started by presenting outputs of two experiments designed to evaluate the impact of respectively the leak rate and the spectral radius on ESN's accuracy.

We continued by analysing the impact of two other hyperparameters on ESN's accuracy, the reservoir size and the training set size. The training set size, specifically the number of documents, is an important hyperparameter as problems in authorship attribution face the lack training samples. Indeed, the amount of data that an author can generate is intrinsically limited. In order to evaluate each features as accurately as possible, we evaluated their average accuracy over 20 ESNs and we searched for significant differences. We also wanted to know which features and models where more impacted by a drop in training samples.

We then performed these experiments and two datasets, the first composed of the whole dataset (100 files per author) and the second composed of 10 files per author. We also evaluated which kind of feature, lexical or syntactic, outperformed the other. We focused the rest of our evaluation on the main question : *which RNN architecture achieve the highest accuracy?* As a consequence, we used output of the various experiments we performed before to compare three architectures, ESNs, LSTMs and GRUs, and a set of baseline methods composed of naive Bayes classifier, Random Forest and SVMs.

The second task was evaluated in Chapter 5 where we investigated the performance of RNN models to predict if a set of tweets have been written by a man or a woman. This task is referred as *Gender Profiling* (GP) and is part of a larger field known as *Author Profiling* (AP). To compare models, we used the PAN @ CLEF 2017 dataset composed of 360,000 tweets for a total of 3,600 profiles (100 tweets per profile). The dataset is well balanced with 1,800 profiles for each group (man/woman). All models must predict the correct label and its performance measure is the evaluated accuracy rate. We have evaluated our models only on the English subset.

Our main concern was to determine which kind of RNN architecture between the shallow one (ESN) and

the deeper ones (GRUs) obtains the highest accuracy rate. To this end, we evaluated these models with various textual representations based on character n-grams and word embedding. As a methodology, we used also 10-fold cross validation and statistical tests to extract significant differences. We also selected a set of classical methods in this field as baselines such as Random Forest, naive Bayes classifier and SVMs (SVMs). We also added a neural network baseline based on Convolutional Neural Network (CNN) and character bigrams. This baseline is referred as CNN-C2 and is based on a common architecture referred as Feed-Forward Neural Networks (FFNNs) with no recursive connections.

After evaluating this neural network baseline, we investigated the impact of various ESN parameters on their evaluated accuracy rates such as leak rate and state-gram. We continued our evaluation by focusing on GRUs and we evaluated these models with various features, trained and pretraining, and hidden layer sizes. We then compared these different RNN architectures and the various baseline methods we have selected.

The last task is split in two problems related to the field of author verification. Given an author and a set of documents of which he is the author, the goal is to determine whether a new document has been written by this target author or not. The first problem is to predict in a multi-authored document whether a token at time t has been written by this target author. We referred to this problem as Author Extraction (AE). The second problem is to determine whether a document contains parts written by a target author (Author Verification in Multi-Authored Documents) using interest points where the authorship probability reached a certain value. Compared to the first problem, we must give an answer for each document and not to the token level.

To evaluate models, we created an home-made dataset composed of science-fiction magazines publicly available through `archive.org`. An important property of these documents is that they were digitised with Optical Character Recognition (OCR). Consequently they contain a very high level of noise and errors. Among several thousand authors available, we selected three well-known science-fiction writers, namely Isaac Asimov, Philip K. Dick and Robert Silverberg. Various issues we used in our dataset were published between 1952 and 1974. These authors are known to not use any pseudonyms. Our dataset is composed of 89 issues containing a novel by at least one of these three authors. We added three issues with no text written by these authors for a total of 91 documents and 8 millions words. Each models are evaluated with the F_1 score as the dataset is very unbalanced and contains much more negative than positive samples.

We started by evaluating the performance of ESN based on pretrained word embedding (ESN-WV) and character trigrams (ESN-C3). We continued by evaluating the performances of GRUs with various features, hidden layer sizes and pretrained or trained features. Using the outputs of these experiments, we compared the various architectures with a baseline model known as SVM Regression. To determine how different features behave, we analyse the training and validation losses during the training phase to know how each feature is impacted by overfitting. Finally, we ended by showing how RNN models can be used to extract evidences from prediction and create a list of typical and important tokens used by a target author.

In this chapter we draw general conclusions on the results obtained on the three tasks and we try to identify different tracks for future research for deep learning models in authorship analysis. In Section 7.1, we compare results obtained on the three tasks. In Section 7.2 we draw general conclusions about results we obtained in this study and the importance of various parameters. Finally in Section 7.3, we introduce a general framework for future research which could exceed the limits of deep learning and neural network models observed in this study.

7.1 Comparisons

In Table 7.1, we show various models we selected and their corresponding evaluated performances. The second column present the accuracy obtained on the authorship attribution task separated in two, the accuracy at the document and sentence level. The third column present the accuracy obtained on the author profiling task. The fourth column present the F_1 scores obtained at the token and document level. Finally, the last column shows the range of parameters needed to be learned.

The top part of Table 7.1, shows results obtained by baseline models on the thee tasks. The middle and bottom parts show results obtained respectively by gated networks (LSTM, GRU) and ESNs. For authorship attribution and author profiling, symbols represent significant differences with the random classifier F_1 score (†, ‡ and ⁑ for respectively $\alpha = 5\%, \alpha = 1\%$ and $\alpha = 0.1\%$). For author verification, symbols represent the number of author predicted significantly (†, ‡ and ⁑ for respectively 1, 2 and 3 authors). Values in bold represent the front runners for each task (no significant differences).

Authorship Attribution On the authorship attribution task at the document level, the highest accuracy is obtained by the best of 50 ESNs ($n_h = 1500$) based on pretrained word embedding with 93.40%. The second highest accuracy is obtained by the best of 50 ESNs ($n_h = 1000$) with 92.80%. These third and fourth models are the linear SVM based on BOW and the ESN based on character encoder (CE) with respectively 92.27% and 92.07%. We found no significant differences between these four models and they then are the state-of-the-art of our evaluations on this task. Gated networks obtained an accuracy significantly lower than the ESN models based on pretrained word embedding (WV) and pretrained character embedding (C3, CE). Compared to baseline models, ESN models (ESN-WV, ESN-C3, ESN-CE) outperform all other models except the linear SVM with which we found no significant differences.

At the sentence level, the highest accuracy is obtained by the pretrained word-based GRUs with 86.57%. The second highest accuracy is achieved by the best of 50 ESNs ($n_h = 1000$) with 84.76%. The third and fourth models are the LSTM based on pretrained word embedding and the average word-based ESN with 83.45% and 83.27%. We found no significant differences between these three models and they then are the state-of-the-art of our evaluations on this task. Gated networks are more efficient at the sentence level as LSTMs and GRUs based on word embedding achieved the top one and three position. Compared to baseline models, ESN models and gated networks clearly outperform linear SVM, Random Forest and Naive Bayes.

Baseline models performed significantly lower than ESNs and gate network models with 74.86% and 72.09% for respectively the BOW-based Naive Bayes and the linear SVM also based on BOW. The other gated network models (GRU, LSTM) also performed significantly lower than ESNs and pretrained word embedding-based LSTM and GRU. For ESN models, those based on pretrained word and character embedding are all parts of the state-of-the-arts or just below. Best-of 50 ESN-WV ($n_h = 1000$), word-based ESN (ESN-WV), character encoder-based ESN (ESN-CE), best-of 50 ESN-WV ($n_h = 1500$) and character trigrams-based ESN (ESN-C3) obtained respectively 84.76%, 83.27%, 80%, 76.88% and 75.9%. These models achieved respectively the second, fourth, fifth, sixth and seventh positions in our evaluation.

On this task, all models were able to perform significantly better than randomly at the document and sentence level.

Author Profiling On the author profiling task, our experiments showed that the best accuracy is obtained by a linear SVM based on BOW with 81.30%. The second best model is an ESN based on pretrained word embedding (ESN-WV) with an accuracy of 80.80% against 79.92% for the third model, a linear SVM based on Bag-Of-Character (BOC). We found no significant differences between these three models and they then are the state-of-the-art of our evaluation of this task. These models are followed by character bigrams-based CNN model (CNN-C2) and two word-based GRU (GRU-WV-P and GRU-WV-T) with respectively 78.80%, 76.17% and 75.50%.

On this task, gated networks performed significantly worse than word-based ESN (ESN-WV) and baseline models such as SVMs. The best GRU model is the pretrained word-based GRU (GRU-WV-P) with an accuracy of 76.17% against 80.8%, 79.92% and 81.30% for the state-of-the-art models. Here, word-based ESNs are the only neural models able to compete with word and character-based SVMs. The other ESN models, including the character trigrams-based ESN (ESN-C3), obtained a lower accuracy than word-based ESN (ESN-WV) and all baseline models (except the random classifier). Furthermore, only four models obtained a p-value higher than 1%, the two GRUs based on pretrained character unigrams and bigrams, and the two ESNs based on POS (ESN-POS) and function words (ESN-FW).

Author Verification The output of our evaluation on the author verification task, both at the token and document level, were highly unstable and then the analysis was very difficult compared to the two other tasks. However, at the token-level, we found two state-of-the-art models, the pretrained word embedding-based ESN (ESN-WV) and the character trigrams-based ESN (ESN-C3), which obtained an average F_1 score of respectively 0.73 and 0.64. GRU models performed significantly worse than the ESN models as the best GRU model obtained an F_1 score of 0.46 compared to 0.64 and 0.73 respectively. For GRUs, best models are based on word embedding (GRU-WV) and trained character trigrams (GRU-C3-T) with respectively 0.48, 0.46 and 0.46. All models performed significantly better than the random classifier for three authors except GRUs based on pretrained character bigrams (C2) and unigrams (C1) which obtained F_1 scores significantly higher than the random classifier only for two authors.

Identifying authors at the document level turned out to be much more difficult than the first problem. Only three models were able to identify two authors significantly and only three were able to achieve this performance for one author. The state-of-the-art is composed of a single model, the character trigrams-based

Classifier	**Attribution**		**Profiling**	**Verification**		**Parameters**
	Document	Sentence		Token	Doc.	
Random baseline	6.66 %	6.66 %	50.00 %	0.11	0.47	
Rand. Forest BOW	80.67 % II	67.47 % II	74.56 % II			9m
Rand. Forest BOC			75.47 % II			
Linear SVM BOW	3) **92.27 %** II	72.09 % II	1) **81.30 %** II		0.44	300k
Linear SVM BOC			3) **79.92 %** II			300k
CNN-C2			78.80 % II			1 m
Pretrained BERT	91.52 % II					$>$ 100m
Naive Bayes BOW	89.20 % II	74.86 % II	75.47 % II			9 - 3 m
LSTM-WV-P	84.53 % II	3) **83.45 %** II				25k
LSTM-WV-T	77.47 % II	68.33 % II				$>$ 10m
LSTM-C3-P	53.75 % II	49.11 % II				430k
GRU-WV-P	83.13 % II	1) **86.57 %** II	76.17 % II	0.46 II	0.41	120k-303k
GRU-WV-T	75.78 II%		75.50 % II	0.48 II	0.59 †	43m
GRU-C1-P	76.88 % II	72.44 % II	53.00 % †	0.28 ‡	0.47 †	130k
GRU-C2-P	64.53 % II	65.07 % II	55.00 % †	0.27 ‡	0.49	150k
GRU-C2-T	65.47 % II		66.44 % II	0.42 II	0.51 †	300k
GRU-C3-P	61.56 % II	61.01 % II	61.28 % II	0.27 II	0.48	10k
GRU-C3-T	73.13 % II		68.83 % II	0.46 II	0.68 ‡	4.5m
ESN-POS	60.80 % II		69.64 % †			15k
ESN-FW	64.20 % II		71.91 % †			15k
ESN-C1	50.73 % II		65.45 % II			15k
ESN-C2	84.13 % II		68.53 % II			15k
ESN-C3	90.67 % II	75.90 % II	70.33 % II	1) **0.73** II	1) **0.80** ‡	15k
ESN-CE	4) **92.07 %** II	80.00 % II				325k
ESN-WV	91.67 % II	83.27 % II	2) **80.80** % II	2) **0.64** II	0.69 ‡	15k
Best-50 ESN-WV	2) **92.80 %** II	2) **84.76** % II				750k
Best-50 ESN-WV	1) **93.40 %** II	76.88 % II				1m

Table 7.1 – Comparison of different models evaluated in this study and various baseline models on three authorship analysis tasks. For authorship attribution and author profiling, symbols represent significant differences with the random classifier F_1 score (†, ‡ and II for respectively $\alpha = 5\%, \alpha = 1\%$ and $\alpha = 0.1\%$). For author verification, symbols represent the number of author predicted significantly (†, ‡ and II for respectively 1, 2 and 3 authors). Values in bold represent the front runners for each task (no significant differences between them).

ESN (ESN-C3) with 0.80 against 0.69 for the second best model, the ESN based on pre-trained word embedding (ESN-WV). The third model is a GRU based on trained character trigrams (GRU-C3-T) which obtained a score equivalent to the word-based ESN with 0.68. These three models were able to identify significantly two authors at the document level. GRUs based on trained word embedding (GRU-WV-T) and character bigrams (GRU-C2-T), and pretrained character unigrams (GRU-C1-P) achieved F_1 scores significant higher than randomly for only one author. Other models, GRU based on pretrained word embedding (GRU-WV-P), character bigrams (GRU-C2-P) and trigrams (GRU-C3-P) were not able to perform better than the random classifier. In comparison to the baseline model, a SVM based on BOW, obtained only a F_1 score of 0.44, also not significantly better than the random classifier.

7.2 Conclusions

Our empirical evaluations allow us to make multiple conclusions. The first concerns the surprisingly good results obtained by ESNs which were never applied to NLP tasks before this study. The second concerns gated networks which have very good performances on all NLP tasks. However, the results of our work have shown that they are not practically efficient on authorship attribution tasks. Our third conclusion is about the differences between various features we tested while our last conclusions will concern differences between trained and pretrained features.

Echo State Networks The key idea behind ESN is to create a reservoir of randomly connected neurons and to train only a simple linear layer. The scientific literature showed that this method is enough to get good results in practice and is a good method to get around the problem of vanishing gradient. ESNs are used for various problem such as timeseries classification and prediction. Here we applied ESNs to NLP tasks for the first time in the scientific literature and showed that it can achieve state-of-the-art performances on three tasks in author analysis.

To achieve these results, we evaluated ESN with various leak rates on two tasks, authorship attribution and author profiling. On both tasks our results showed that the dynamics of reservoirs on textual data is very low. On authorship attribution, the best accuracy is achieved with leak rates of 0.01, 0.001, 0.01, 0.01, 0.1, 0.1 and 0.3 for respectively character encoder (CE), character trigrams (C3), word embedding (WV), character bigrams (C2), function words (FW), Part-Of-Speech (POS) and character unigrams (C1). On author profiling, best accuracy is obtained with a leak rate value of 0.01, 0.05, 0.05, 0.4, 0.01 and 0.01 respectively for word embedding (WV), Part-Of-Speech (POS), function words (FW), character unigrams (C1), bigrams (C2) and trigrams (C3). The dynamics is very low in both case except for character unigrams (C1) on authorship attribution with 0.3, and for character bigrams (C2) on author profiling with 0.4. More evaluations should be done to confirm whether these features have faster dynamics and to understand this properties on a more theoretical ground.

We also investigated the accuracy obtained by ESN with different spectral radius. Spectral radius is the biggest absolute eigen value of the internal connection matrix $\boldsymbol{W}$. In the Reservoir Computing (RC) literature, it is a well-known fact that spectral radius should be set to be just under the border of chaos to achieve maximal memory and computing capability. Furthermore, reservoir with spectral radius above the border of chaos (1.0) can show chaotic behaviors where inputs are amplified. Here we evaluated the accuracy of ESNs with various spectral radius on the authorship attribution task and we found no improvement just before the border and chaos. Moreover, we found that all feature are effected by chaotic behaviours with spectral radius above 1.0, except for pretrained word embedding which stay stable up to a spectral radius of 2.0.

We also investigated the impact of training set size on the authorship attribution task. The main idea was to compare the evaluated accuracy of ESNs, baseline and LSTM models with different training set size. We also designed an experiment where we trained 20 ESN models and baseline models with only 10 files per author in order to have a more precise evaluation of their capacity to handle small datasets. Our results showed that word embedding-based and character encoder-based ESNs are part of the state-of-the-art methods on this benchmark with BOW-based SVMs. We think that this result showed that ESNs are very interesting models on authorship attribution tasks and should be investigated further.

Comparing the three tasks, our investigations showed that ESNs are systematically part of the front runners of our evaluations. On the authorship attribution problem, the word-based best-of 50 ESNs with 1,000 and 1,500 units, and the ESN based on character encoder (CE) achieved respectively the first, second and

fourth position on the document-level benchmark. The average ESN based on word embedding achieved fifth position just below state-of-the-art models. In comparison, the main baseline, a SVM based on BOW, achieved the third position. On the author profiling task, word-based ESNs (ESN-WV) achieved the second position, just below a linear SVM based on BOW, and is part of the state-of-the-art methods on this benchmark. On the last task, the author verification problem, two ESN models, based on word embedding (ESN-WV) and pretrained character trigrams (ESN-C3), achieved the top two position and outperformed all other models at the token level (Author Extraction). At the document level (Author Verification in Multi-Authored Documents), the character-based ESN (ESN-C3) clearly outperformed other models while keeping a low number of parameters to be learned.

We think our work gathered a convincing amount of evidence in favour of ESNs applied to authorship analysis problem and NLP tasks. We showed that dynamical properties should be slow and that word embedding textual representations seems to be immune to chaotic behaviours. The lack of transparency of neural models is a strong drawback in authorship analysis. As a result, we showed how the outputs of these models can be analysed in order to extract evidences and justifications out of their predictions. Our final argument concerns the number of training samples required and time necessary to train these models. Indeed, these two points are also strong drawbacks for neural models as deep learning models are hunger for data and need a large amount of computing power to be trained. We showed here that a shallower neural models can achieve state-of-the-art results while keeping its number of parameters very low. As a consequence, they are easy and fast to train even with a small amount of training samples. We think these results show that ESNs should be investigated more on various datasets and NLP tasks and should be considered more often in authorship analysis studies.

Gated Networks We investigated gated networks such as LSTMs or GRUs on the three authorship analysis tasks. We evaluated the accuracy of LSTMs on the authorship attribution task, and GRU's on the three benchmarks. We tested gated networks with various parameters and we compared these results with ESN and baseline models.

The difference between LSTMs and GRUs was covered in Chapter 4 and evaluated on the authorship attribution task. At document level, we showed that LSTM achieved higher accuracy than GRUs, while GRUs achieved highest accuracy at the sentence level. In both, we found no significant differences between these two models. Therefore, our results showed that GRUs are able to achieve similar accuracy than LSTMs while having fewer parameters. As the number of parameters impacts the number of samples necessary for training, GRUs are more suitable for authorship analysis.

Compared to ESNs, GRUs were outperformed at the document level but achieved higher accuracy at the sentence level. To explain this result, we hypothesised that GRUs can model temporal dependencies over few dozens time steps due to vanishing gradients. In contrary, ESNs being trained with linear methods, they are able to model longer dependencies but with less accuracy in short term. Regarding their number of parameters, the best LSTM at the document level has 725,715 parameters to be evaluated while the best ESN has only 15,000 parameters. Compared to baseline models, LSTMs are significantly outperformed by all baseline models except the Random Forest based on BOW.

On the author profiling task, ESNs based on pretrained word embedding significantly outperformed GRUs based on the same feature. However, GRUs based on trained and pretrained word embedding outperformed the other ESN models such as character-based ESN (ESN-C3, ESN-C2 and ESN-C1). Compared to baseline models, GRUs were significantly outperformed by three baseline models, the two SVMs based on BOW and Bag-Of-Character. Regarding the number of parameters, the best GRU model (GRU-WV-P) has 120,802 parameters while best SVMs and word-based ESN have respectively more than one million and 2,000 parameters to be learned.

On the author verification task at the token level (AE), GRUs are outperformed by the ESN model based on character trigrams (ESN-C3) on Asimov's novels. On Philip K. Dick, the best GRU model is outperformed by word and character trigrams-based ESNs (ESN-WV, ESN-C3). On Silverberg, we found no significant differences between the two ESN models and the top GRU model. At the document level (AVMA), we found no significant difference between the best GRU and ESN models (GRU-C3-T, ESN-C3). On Dick's novels, GRU models are again significantly outperformed by the two ESN models. On Silverberg's corpus, GRUs are outperformed by the word-based ESNs but achieved a score similar to the character trigrams-based ESN (ESN-C3). Compared to the baseline model, the linear SVM based on BOW, GRUs are able to predict two authors at the document level while SVMs do not perform better than the random classifier.

The main drawback of gated networks is the high number of parameters to be evaluated and, as a

consequence, the need for training samples. This drawback put a limit to the application of these kind of neural models in the field of authorship analysis. However, the use of SGD allows an infinite number of architectures and operations and we therefore think that these result should not be interpreted as proof that gated networks and deep learning models cannot perform well on authorship analysis tasks. But we think it shows that the paradigm should be changed and that deep neural network models should be used in the context of profile-based methods.

Pretrained vs trained features On the authorship attribution task at document level, we showed that pretrained features are more efficient than their trained counterparts except for character bigrams (C2) and trigrams (C3). At the sentence level, highest accuracy is also obtained with pretrained features.

On the author profiling task, the best GRU model is based on word embedding with no differences between pretraining and training. For character unigrams (C1), bigrams (C2) and trigrams (C3), trained versions of these features performed better than their pretrained versions.

On the author verification task at the token level, pretrained and trained word-based features obtained the same score. For character-based features, trained version of character bigrams (C2) and trigrams (C3) performed significantly better than their pretrained counterparts. At the document level, trained version of word embedding (WV) performed better. The same observation can be done for character-based features (C1, C2, C3).

In conclusion, pretrained features obtained higher accuracy on the authorship attribution task, while trained features performed better on the author profiling and author verification tasks. This observation could be explained by the differences in vocabulary used for the different tasks. The author profiling task use a lot of different words based on hashtags, smiles and references (@). Similarly, the dataset used for the author verification task is highly noisy. These differences could explain that trained features work better on the author profiling and verification task. More work should be done here with textual representations pretrained on Twitter data, or with data preprocessing in order to remove noise.

Word-based vs character-based features On the authorship attribution task at the document level, LSTMs and GRUs are more efficient with word-based feature, especially with pretrained word embedding (WV-P). Character-based features slightly increased with increasing n-grams for LSTMs but not for GRUs. For ESNs, best results where obtained with pretrained word embedding (ESN-WV), character trigrams (ESN-C3) and character encoder (ESN-CE). We also showed that increasing n-grams increased accuracy even more for ESNs. To explain this observation, we hypothesised that increasing n-grams can increase accuracy if the model used is able to model time dependencies of the required length. Regarding syntactic features, GRU models based on POS and function words (FW) where not able to learn this task. For ESN models, POS and function words (FW) performed worse than the character-based models but significantly higher than a random classifier. At the sentence level, LSTMs and GRUs were also more efficient with word-based features compared to the other features. For ESNs, we did the same observation than at the document level as they were also more efficient with pretrained word embedding (ESN-WV), character trigrams (ESN-C3) and character encoder (ESN-CE).

For the author profiling task, GRU models performed better with both pretrained and trained word-based features. As with the authorship attribution task, increasing n-grams increased the accuracy for GRU models with pretrained and trained character features. For ESNs, word-based models performed significantly better than character-based features. As with GRUs, increasing n-grams increased accuracy. Unlike the authorship attribution task, function words (FW) performed better than all character-based features and POS obtained higher accuracy than character trigrams (C2) and character unigrams (C1).

On the author verification task at the token level (AE), character trigrams performed better for ESN (ESN-C3) on Asimov's and Dick's novels, and with no significant difference with word-based (ESN-WV) model on Silverberg's. For GRUs, word and character trigrams-based models got similar results. At the document level (AVMA), word-based and character trigrams-based features obtained the best score with both GRUs and ESNs.

In conclusions, features based on character trigrams and word embedding obtained the best results on the three problems. On some task, word based performed better such as author profiling, while character trigrams (C3) performed better on author verification. Finally, we must take into account conclusions made above about trained and pretrained features. Effectively, more experiments should be done with word and character embedding pretrained on Twitter data and with noise removal data preprocessing in order to determine which feature can achieve higher accuracy.

Cell/hidden sizes For authorship attribution at the document level, we concluded that increasing the cell size of LSTM models for character trigrams and bigrams (C3, C2) can increase the accuracy. We also concluded that increasing the number of units of gated networks does not necessarily imply overfitting on this task, thanks to the regularisation power of dropout layers. Regarding pretrained features, we showed that some significant increases are possible for character-based representations but evidences were weaker. Our results also showed that some problem of overfitting are possible with GRUs based on word-embedding.

On the author profiling task, our results showed that increasing hidden sizes for GRUs can have a significant impact on accuracy rate. We showed that increasing hidden sizes allowed GRUs to achieve higher accuracy with pretrained word embedding but lowered accuracy for trained word embedding. As trained word embedding have much more parameters, it could be the sign of overfitting and more work could be done to work around this issues with more regularisation methods.

For the author verification task, we were unable to draw any conclusions on the impact of hidden layer sizes due to the high variability of model's performances. More work is then necessary on this point in the future with an improved version of the SFGram dataset with more samples and a bigger set of target authors.

7.3 Future Work

Beyond the results presented in this study, a large field of possible improvements and evaluations are feasible. First on the side of feature, our current evaluation of the POS feature could be improved by using sequences of pretrained embedding of POS tags instead of one hot vectors that will allow the ESN to more easily generalise to new samples. This could be used with ESNs as a new feature or in combination with pretrained word embedding. For GRUs and LSTMs, pretrained and trained POS could also be used in combination with word embedding. Result from the PAN @ CLEF author profiling task showed that state-of-the-art models in this lab use combination of features. An interesting path of research would be then to evaluate combination of features with recurrent neural models. On the side of character embedding, a large field of different parameters is open to investigation such as the number of dimension and the context size. Bigger character n-grams ($n > 3$) are equally open to examination. We could also pretrained character embedding on different kind of textual data coming from social networks in order to evaluate more accurately RNN models on author profiling tasks.

Furthermore, we want to evaluate in the future other recurrent models with deeper architecture. For example, the field of *Deep Reservoir Computing* with stacked architectures such as *Stacked-ESN* and *Deep-ESN* which are able to create hierarchical temporal representations of timeseries. In this thesis we only explored shallow ESNs with only one layer. However, these models have much more parameters to be optimised and they require deeper work to assess them properly. We could also combine ESN and gated networks to exploit both architectures. Regarding neural models, the state-of-the-art model in a lot of NLP tasks is not Transformer networks and BERT. We think then that these kind of neural models should be evaluated on different author analysis tasks such as those evaluated in this study. As author analysis required evidence-based methods, we would also like to evaluate and implement methods able to give even more information out of their prediction. For example with attention layers and deconvulational layers. In combination with many-to-many architectures this could open the way to more evidence-based neural models. Other neural architectures could also be evaluated based on Convolutional Neural Networks (CNN) in order to evaluate some of their properties that could match the needs of the field of author analysis.

In the future, we would like to evaluate neural models on other datasets with more authors and apply these models to collaborative work and social medias to detect points of interest in streams of text. As we have seen, performances showed high variability on the author verification task, we want then to continue our work on the SFGram dataset by increasing the number of issue and the number of authors. This dataset could then be used to evaluate different models on the task of author verification in stream of text. As gated networks require large datasets, we would also evaluate these models on larger datasets composed of Twitter profiles.

Finally, our work showed that the gated networks we evaluated in this study were not able to achieve state-of-the-art performances. We hypothesised that this is due to the large number of parameters and to the SGD algorithm which need large dataset to evaluate good parameter values. A solution to this issue and to improve the performance of deep learning methods on author analysis problems would be to switch from profile-based methods to instance-instance based methods. The main idea is to train deep learning models to learn distance measure between authors rather than multi-author classification systems. These models could

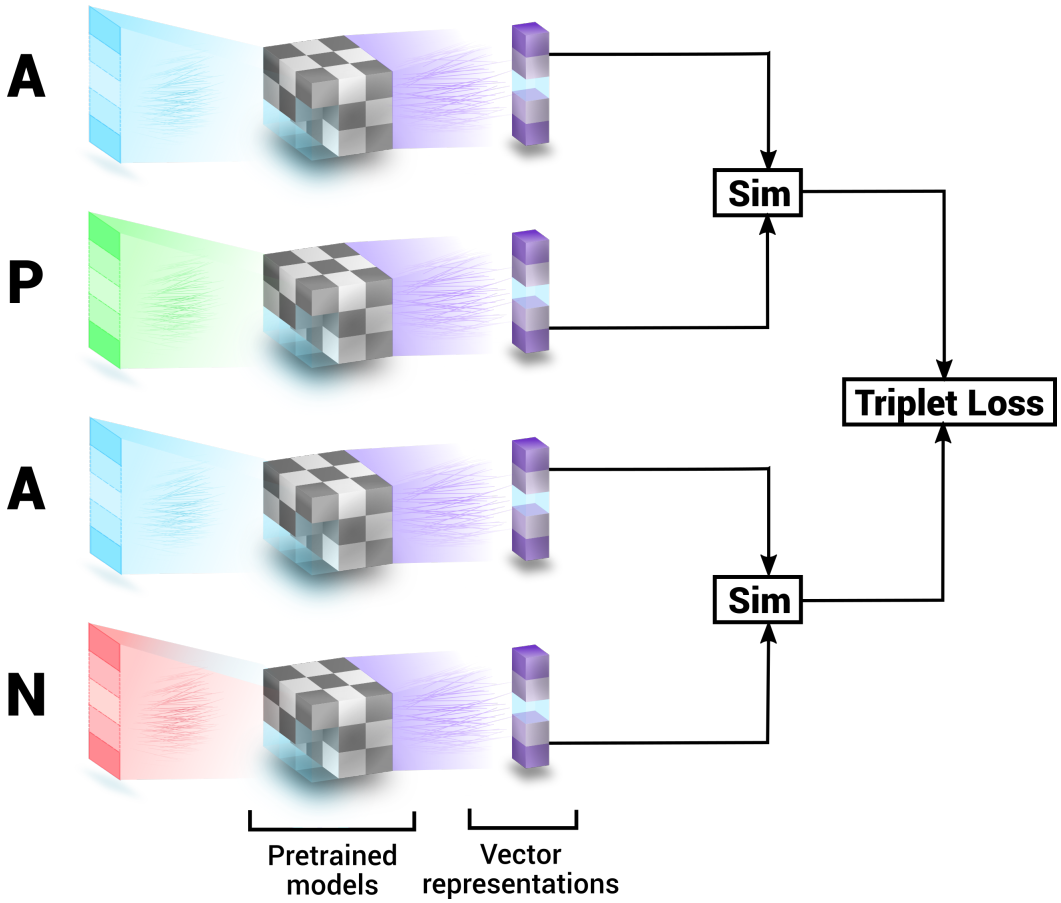


Figure 7.1 – Instance-based Author Analysis system with a triplet loss function.

be pretrained to learn a distance measure on very large datasets and used on a new unseen dataset. The next and final section will introduce this project in more detail.

7.3.1 Authorship Analysis, pretraining and few-shot learning

This thesis identified the main drawbacks with neural networks and deep learning in the field of authorship analysis. First, the small amount of data available to train models with a lot of parameters. Secondly, the training time of deep models that greatly slows down the development, testing and research loop. And third, the choice of the loss function and the significant number of parameters to tune and explore. One possible solution to the issue of small datasets would be to get inspiration from the problem of face recognition in computer vision. The task of face recognition consists to link an image of a face of a unknown person to face images contained in a database. This task face the same problems than authorship analysis. First we have a lot of different possible authors (or person) with a very small amount of samples per class. Second, face recognition systems only have a few examples available (less than ten face images per person), but must be able to compute similarities based only on these examples.

This approach is named *few-shot learning* in the deep learning framework to underline the problem of learning from just few examples. In document classification, this approach is similar to *instance-based learning* where the model has to learn similarities between instances, in opposition to *profile-based learning* where the models use a set of training document to compose a profile of the author. In face recognition, the system has to link a single unknown instance to other instances contained in a dataset by computing a similarity measure and defining a significance threshold.

An appropriate loss function to learn similarities in face recognition is the *Triplet Loss* function [178]. This loss function is used in face recognition to learn similarities between a pair of faces [179]. This loss function use a baseline (named anchor) input which is compared with a negative (from a different class)

and a positive (from the same class) sample. The key idea is to minimise the distance between the baseline and the positive sample while maximising the distance between the baseline and the negative example. The triplet loss function is defined as,

$$\mathscr{L}(A, P, N) = \max(sim^{\alpha}_{A,N} - sim^{\alpha}_{A,P} + \alpha, 0) \tag{7.1}$$

where A is the baseline sample extracted randomly from the dataset. P is the positive example belonging to the same class than A, and N is the negative sample belonging to a different class. $sim^{\alpha}_{A,N}$ and $sim^{\alpha}_{A,N}$ are respectively the similarity between the baseline and the negative sample and between the baseline and the positive sample. α is a margin between the positive and the negative distances which prevents the two similarities from deviating towards zero. As distance measure, one can use for example the Euclidian distance,

$$\mathscr{L}(A, P, N) = \max(||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + \alpha, 0) \tag{7.2}$$

The *loss function* is then the sum of the *Triplet Loss* over batch samples.

$$J(\alpha) = \sum_{i=1}^{n} \mathscr{L}(A^{(i)}, P^{(i)}, N^{(i)}, \alpha) \tag{7.3}$$

The loss function could be integrated in a model to learn similarities between documents. The key idea would be to use heavily pretrained models to learn a similarity measure with pairs of documents from the sample author (P) and from different authors (N). Pretrained models could be RNNs, FFNN or BERT-based models with a sliding window that would encode documents into a single vector representation, and pretrained on common task in NLP such as next word prediction. As shown in Figure 7.1, the whole system would be trained on a dataset containing a huge number of authors. If the dataset has enough author, it could learn the distance measure effectively even with a small number of documents per author. The goal would be for the model to learn a set of feature heavily related to authorship which would allow to overcome the small dataset issue identified in this study. The extracted feature set could combine word-based information and character-based inputs. Once the similarity learned, this model could be used on new dataset to compare documents and determine if a pair of document is from the same author by defining a threshold that surrounds positive samples. Part of the new dataset could be used to compute that threshold. The need for training data would be greatly reduced as the number of parameters to train would be small, and the training time would be greatly reduced also as we only have to run the model on pairs of example. This line of research would require four steps.

First, we would have to create of huge authorship analysis dataset with multiple labelled data to train instance-based deep learning models. As deep learning models need a lot of data, this dataset should be composed of thousands of authors with dozens of documents per authors. Each document should be single authored and the authorship of each document should be certain and no pen name should be present in the dataset. Very efficient deep learning models in language modelling are trained on several hundreds of millions of examples. In the social network and blog era, constructing such a authorship attribution dataset for deep models should be achievable.

The second step would be to pretrain instance-based deep models on this big dataset to learn pretrained features related to authorship. This step would require a lot of computing power which would be achievable through GPUs and dedicated devices. The model would use the triplet loss function with baseline, positive and negative samples chosen randomly in the dataset. The possible issue here is the possible overfitting with models with to many parameters. The open question would be to know which kind of models (CNN, FFNN, RNN) is the most effective.

In the third step, we would explore the performance of different instance-based deep models pretrained at step 2 on classical and new authorship analysis datasets to determine if it is possible to transfer learned competences on new datasets. The open question here is to determine if common features can be found and transfer between author analysis tasks and datasets.

Finally, at step 4, we would explore the possible integration of attention layers and models to extract information and justifications about the decision made by models. Attention layers could be used to determine on which part of the two documents tested the model is focusing on to compute the similarity measure.

Bibliography

[1] T. Solorio, R. Hasan, and M. Mizan, "A case study of sockpuppet detection in wikipedia," in *Proceedings of the Workshop on Language Analysis in Social Media*, pp. 59–68, 2013.

[2] C. Machinery, "Computing machinery and intelligence-am turing," *Mind*, vol. 59, no. 236, p. 433, 1950.

[3] N. Chomsky, *Syntactic structure*. Mouton, 1957.

[4] J. Pierce, J. Carroll, E. Hamp, D. Hays, C. Hockett, A. Oettinger, and A. Perlis, "Computers in translation and linguistics (alpac report). report 1416," *National Academy of Sciences/National Research Council*, 1966.

[5] J. Deng, A. Berg, S. Satheesh, H. Su, A. Khosla, and F. Li, "Large scale visual recognition challenge 2012," in *ILSVRC 2012 Workshop*, 2012.

[6] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE Access*, vol. 7, pp. 19143–19165, 2019.

[7] J. Karhunen, T. Raiko, and K. Cho, "Unsupervised deep learning: A short review," in *Advances in Independent Component Analysis and Learning Machines*, pp. 125–142, Elsevier, 2015.

[8] D. Zhang, M. M. Islam, and G. Lu, "A review on automatic image annotation techniques," *Pattern Recognition*, vol. 45, no. 1, pp. 346–362, 2012.

[9] D. Benni, F. Chauvet, and A. Guillot, "Human-machine dialog system," Aug. 9 2016. US Patent 9,411,370.

[10] W. J. Hutchins and H. L. Somers, *An introduction to machine translation*, vol. 362. Academic Press London, 1992.

[11] C. D. Manning, H. Schütze, and P. Raghavan, *Introduction to information retrieval*. Cambridge university press, 2008.

[12] M. Iyyer, V. Manjunatha, J. Boyd-Graber, and H. Daumé III, "Deep unordered composition rivals syntactic methods for text classification," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, pp. 1681–1691, 2015.

[13] R. S. Cooper, J. F. McElroy, W. Rolandi, D. Sanders, R. M. Ulmer, and E. Peebles, "Personal virtual assistant," June 29 2004. US Patent 6,757,362.

[14] T. C. Mendenhall, "The characteristic curves of composition," *Science*, vol. 9, no. 214, pp. 237–249, 1887.

[15] G. K. Zipf, *The psychobiology of language*. Houghton, Mifflin, 1935.

[16] A. Bissell, "Cusum techniques for quality control," *Applied Statistics*, pp. 1–30, 1969.

[17] D. I. Holmes and F. J. Tweedie, "Forensic stylometry: A review of the cusum controversy," *Revue Informatique et Statistique dans les Sciences Humaines*, vol. 31, no. 1, pp. 19–47, 1995.

[18] F. Iqbal, R. Hadjidj, B. C. Fung, and M. Debbabi, "A novel approach of mining write-prints for authorship attribution in e-mail forensics," *digital investigation*, vol. 5, pp. S42–S51, 2008.

[19] O. De Vel, A. M. Anderson, M. W. Corney, and G. M. Mohay, *Multi-topic e-mail authorship attribution forensics*. ACM, 2001.

[20] M. Cristani, G. Roffo, C. Segalin, L. Bazzani, A. Vinciarelli, and V. Murino, "Conversationally-inspired stylometric features for authorship attribution in instant messaging," in *Proceedings of the 20th ACM international conference on Multimedia*, pp. 1121–1124, ACM, 2012.

[21] A. Orebaugh and J. Allnutt, "Classification of instant messaging communications for forensics analysis," *The International Journal of Forensic Computer Science*, vol. 1, pp. 22–28, 2009.

[22] R. Layton, P. Watters, and R. Dazeley, "Authorship attribution for twitter in 140 characters or less," in *Cybercrime and Trustworthy Computing Workshop (CTC), 2010 Second*, pp. 1–8, IEEE, 2010.

[23] C. Labbé and D. Labbé, "Inter-textual distance and authorship attribution corneille and molière," *Journal of Quantitative Linguistics*, vol. 8, no. 3, pp. 213–231, 2001.

[24] A. Tuzzi and M. A. Cortelazzo, "It takes many hands to draw elena ferrante's profile," *UPPADO*, p. 9.

[25] J. Savoy, "Is starnone really the author behind ferrante?," *Digital Scholarship in the Humanities*, 2018.

[26] G. K. Mikros, "Blended authorship attribution: Unmasking elena ferrante combining different author profiling methods," *UPPADO*, p. 85.

[27] A. Abbasi and H. Chen, "Applying authorship analysis to extremist-group web forum messages," *IEEE Intelligent Systems*, vol. 20, no. 5, pp. 67–75, 2005.

[28] G. Frantzeskou, E. Stamatatos, S. Gritzalis, and S. Katsikas, "Effective identification of source code authors using byte-level information," in *Proceedings of the 28th international conference on Software engineering*, pp. 893–896, ACM, 2006.

[29] S. Burrows and S. M. Tahaghoghi, "Source code authorship attribution using n-grams," in *Proceedings of the Twelth Australasian Document Computing Symposium, Melbourne, Australia, RMIT University*, pp. 32–39, Citeseer, 2007.

[30] S. Burrows, A. L. Uitdenbogerd, and A. Turpin, "Application of information retrieval techniques for source code authorship attribution," in *International Conference on Database Systems for Advanced Applications*, pp. 699–713, Springer, 2009.

[31] G. Frantzeskou, S. G. MacDonell, and E. Stamatatos, "Source code authorship analysis for supporting the cybercrime investigation process," in *Handbook of Research on Computational Forensics, Digital Crime, and Investigation: Methods and Solutions*, pp. 470–495, IGI Global, 2010.

[32] G. U. Yule, "On sentence-length as a statistical characteristic of style in prose: With application to two cases of disputed authorship," *Biometrika*, vol. 30, no. 3/4, pp. 363–390, 1939.

[33] A. Honoré, "Some simple measures of richness of vocabulary," *Association for literary and linguistic computing bulletin*, vol. 7, no. 2, pp. 172–177, 1979.

[34] C. C. Fries, *The structure of English*. Harcourt Brace, 1952.

[35] P. Mcnamee and J. Mayfield, "Character n-gram tokenization for european language text retrieval," *Information retrieval*, vol. 7, no. 1-2, pp. 73–97, 2004.

[36] E. Stamatatos, "Intrinsic plagiarism detection using character n-gram profiles," *threshold*, vol. 2, no. 1,500, 2009.

[37] V. Kešelj, F. Peng, N. Cercone, and C. Thomas, "N-gram-based author profiles for authorship attribution," in *Proceedings of the conference pacific association for computational linguistics, PACLING*, vol. 3, pp. 255–264, sn, 2003.

[38] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[39] A. Globerson, G. Chechik, F. Pereira, and N. Tishby, "Euclidean embedding of co-occurrence data," *Journal of Machine Learning Research*, vol. 8, no. Oct, pp. 2265–2295, 2007.

[40] R. Lebret and R. Collobert, "Word emdeddings through hellinger pca," *arXiv preprint arXiv:1312.5542*, 2013.

[41] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Advances in neural information processing systems*, pp. 2177–2185, 2014.

[42] Y. Li, L. Xu, F. Tian, L. Jiang, X. Zhong, and E. Chen, "Word embedding revisited: A new representation learning and explicit matrix factorization perspective.," in *IJCAI*, pp. 3650–3656, 2015.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[44] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[45] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[46] G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[47] M. Koppel, N. Akiva, and I. Dagan, "Feature instability as a criterion for selecting potential style markers," *Journal of the American Society for Information Science and Technology*, vol. 57, no. 11, pp. 1519–1525, 2006.

[48] P. Comon and C. Jutten, *Handbook of Blind Source Separation: Independent component analysis and applications*. Academic press, 2010.

[49] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[50] R. Collobert, "Word embeddings through hellinger pca," in *in Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, Citeseer, 2014.

[51] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84–96, 2014.

[52] A. Krivine, *Çomprendre sans prévoir, prévoir sans comprendre*. Cassini, 2018.

[53] C. Anderson, "The end of theory: The data deluge makes the scientific method obsolete," *Wired magazine*, vol. 16, no. 7, pp. 16–07, 2008.

[54] D. C. Dennett, *From bacteria to Bach and back: The evolution of minds*. WW Norton & Company, 2017.

[55] W. Samek, G. Montavon, A. Vedaldi, L. K. Hansen, and K.-R. Müller, *Explainable AI: interpreting, explaining and visualizing deep learning*, vol. 11700. Springer Nature, 2019.

[56] M. Zechner, M. Muhr, R. Kern, and M. Granitzer, "External and intrinsic plagiarism detection using vector space models," in *Proc. SEPLN*, vol. 32, pp. 47–55, 2009.

[57] S. M. Zu Eissen and B. Stein, "Intrinsic plagiarism detection," in *European Conference on Information Retrieval*, pp. 565–569, Springer, 2006.

[58] I. Bensalem, P. Rosso, and S. Chikhi, "Intrinsic plagiarism detection using n-gram classes," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1459–1464, 2014.

[59] G. Oberreuter, G. L'Huillier, S. A. Rıos, and J. D. Velásquez, "Approaches for intrinsic and external plagiarism detection," *Proceedings of the PAN*, 2011.

[60] E. Stamatatos, M. Tschnuggnall, B. Verhoeven, W. Daelemans, G. Specht, B. Stein, and M. Potthast, "Clustering by authorship within and across documents," in *Working Notes Papers of the CLEF 2016 Evaluation Labs. CEUR Workshop Proceedings/Balog, Krisztian [edit.]; et al.*, pp. 691–715, 2016.

[61] W. R. Ashby, *An introduction to cybernetics*. Chapman & Hall Ltd, 1961.

[62] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[63] Y. Kim, C. Denton, L. Hoang, and A. M. Rush, "Structured attention networks," *arXiv preprint arXiv:1702.00887*, 2017.

[64] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[65] J. Ba, V. Mnih, and K. Kavukcuoglu, "Multiple object recognition with visual attention," *arXiv preprint arXiv:1412.7755*, 2014.

[66] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 1480–1489, 2016.

[67] N. Pappas and A. Popescu-Belis, "Multilingual hierarchical attention networks for document classification," *arXiv preprint arXiv:1707.00896*, 2017.

[68] X. Zhou, X. Wan, and J. Xiao, "Attention-based lstm network for cross-lingual sentiment classification," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 247–256, 2016.

[69] J. Lu, C. Xiong, D. Parikh, and R. Socher, "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 375–383, 2017.

[70] C. Liu, J. Mao, F. Sha, and A. Yuille, "Attention correctness in neural image captioning," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[71] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[72] L. Vanni, M. Ducoffe, D. Mayaffre, F. Precioso, D. Longrée, V. Elango, N. S. Buitrago, J. G. Huesca, L. Galdo, and C. Aguilar, "Text deconvolution saliency (tds): a deep tool box for linguistic analysis," 2018.

[73] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital Signal Processing*, vol. 73, pp. 1–15, 2018.

[74] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[75] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649, IEEE, 2013.

[76] R. Collobert and J. Weston, "Fast semantic extraction using a novel neural network architecture," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pp. 560–567, 2007.

[77] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.

[78] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[79] J. T. Goodman, "A bit of progress in language modeling," *Computer Speech & Language*, vol. 15, no. 4, pp. 403–434, 2001.

[80] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur, "Recurrent neural network based language model," in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[81] T. Mikolov, S. Kombrink, L. Burget, J. Černockỳ, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 5528–5531, IEEE, 2011.

[82] P. Koehn, *Statistical machine translation*. Cambridge University Press, 2009.

[83] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[84] H. Zhou, Y. Zhang, S. Huang, and J. Chen, "A neural probabilistic structured-prediction model for transition-based dependency parsing," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, pp. 1213–1222, 2015.

[85] R. Johnson and T. Zhang, "Semi-supervised convolutional neural networks for text categorization via region embedding," in *Advances in neural information processing systems*, pp. 919–927, 2015.

[86] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.

[87] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150, Association for Computational Linguistics, 2011.

[88] T. H. Nguyen and R. Grishman, "Relation extraction: Perspective from convolutional neural networks," in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing*, pp. 39–48, 2015.

[89] W. Yin, H. Schütze, B. Xiang, and B. Zhou, "Abcnn: Attention-based convolutional neural network for modeling sentence pairs," *arXiv preprint arXiv:1512.05193*, 2015.

[90] C. D. Santos and B. Zadrozny, "Learning character-level representations for part-of-speech tagging," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1818–1826, 2014.

[91] M. Auli and J. Gao, "Decoder integration and expected bleu training for recurrent neural network language models," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, pp. 136–142, 2014.

[92] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

[93] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks," 2013.

[94] O. Irsoy and C. Cardie, "Deep recursive neural networks for compositionality in language," in *Advances in neural information processing systems*, pp. 2096–2104, 2014.

[95] M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney, "Translation modeling with bidirectional recurrent neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 14–25, 2014.

[96] E. Kiperwasser and Y. Goldberg, "Simple and accurate dependency parsing using bidirectional lstm feature representations," *arXiv preprint arXiv:1603.04351*, 2016.

[97] T. Watanabe and E. Sumita, "Transition-based neural constituent parsing," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, pp. 1169–1179, 2015.

[98] G. Chrupała, "Normalizing tweets with edit scripts and recurrent neural embeddings," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, pp. 680–686, 2014.

[99] A. Sordoni, M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan, "A neural network approach to context-sensitive generation of conversational responses," *arXiv preprint arXiv:1506.06714*, 2015.

[100] W. Ling, T. Luís, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso, "Finding function in form: Compositional character models for open vocabulary word representation," *arXiv preprint arXiv:1508.02096*, 2015.

[101] R. A. Matthews and T. V. Merriam, "Neural computation in stylometry i: An application to the works of shakespeare and fletcher," *Literary and Linguistic Computing*, vol. 8, no. 4, pp. 203–209, 1993.

[102] T. V. Merriam and R. A. Matthews, "Neural computation in stylometry ii: An application to the works of shakespeare and marlowe," *Literary and Linguistic Computing*, vol. 9, no. 1, pp. 1–6, 1994.

[103] B. Kjell, "Authorship attribution of text samples using neural networks and bayesian classifiers," in *IEEE INTERNATIONAL CONFERENCE ON SYSTEMS MAN AND CYBERNETICS*, vol. 2, pp. 1660–1660, INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1994.

[104] F. J. Tweedie, S. Singh, and D. I. Holmes, "Neural network applications in stylometry: The federalist papers," *Computers and the Humanities*, vol. 30, no. 1, pp. 1–10, 1996.

[105] J. F. Hoorn, S. L. Frank, W. Kowalczyk, and F. van Der Ham, "Neural network identification of poets using letter sequences," *Literary and Linguistic Computing*, vol. 14, no. 3, pp. 311–338, 1999.

[106] U. Stańczyk and K. A. Cyran, "Machine learning approach to authorship attribution of literary texts," *International journal of applied mathematics and informatics*, vol. 1, no. 4, pp. 151–158, 2007.

[107] M. Tearle, K. Taylor, and H. Demuth, "An algorithm for automated authorship attribution using neural networks," *Literary and linguistic computing*, vol. 23, no. 4, pp. 425–442, 2008.

[108] M. Bagavandas, A. Hameed, and G. Manimannan, "Neural computation in authorship attribution: The case of selected tamil articles," *Journal of Quantitative Linguistics*, vol. 16, no. 2, pp. 115–131, 2009.

[109] N. Tsimboukakis and G. Tambouratzis, "A comparative study on authorship attribution classification tasks using both neural network and statistical methods," *Neural Computing and Applications*, vol. 19, no. 4, pp. 573–582, 2010.

[110] A. Neme, B. Lugo, and A. Cervera, "Authorship attribution as a case of anomaly detection: A neural network model," *International Journal of Hybrid Intelligent Systems*, vol. 8, no. 4, pp. 225–235, 2011.

[111] U. Bandara and G. Wijayarathna, "Deep neural networks for source code author identification," in *International Conference on Neural Information Processing*, pp. 368–375, Springer, 2013.

[112] R. Chandrasekaran and G. Manimannan, "Use of generalized regression neural network in authorship attribution," *International Journal of Computer Applications*, vol. 62, no. 4, 2013.

[113] D. Rhodes, "Author attribution with cnns," *Avaiable online: https://www. semanticscholar. org/paper/Author-Attribution-with-Cnn-s-Rhodes/0a904f9d6b47dfc574f681f4d3b41bd840871b6f/pdf (accessed on 22 August 2016)*, 2015.

[114] D. Bagnall, "Author identification using multi-headed recurrent neural networks," *arXiv preprint arXiv:1506.04891*, 2015.

[115] S. Macke and J. Hirshman, "Deep sentence-level authorship attribution," 2015.

[116] A. Neme, J. Pulido, A. Muñoz, S. Hernández, and T. Dey, "Stylistics analysis and authorship attribution algorithms based on self-organizing maps," *Neurocomputing*, vol. 147, pp. 147–159, 2015.

[117] Z. Ge, Y. Sun, and M. J. Smith, "Authorship attribution using a neural network language model.," in *AAAI*, pp. 4212–4213, 2016.

[118] S. Ruder, P. Ghaffari, and J. G. Breslin, "Character-level and multi-channel convolutional neural networks for large-scale authorship attribution," *arXiv preprint arXiv:1609.06686*, 2016.

[119] X. Yang, G. Xu, Q. Li, Y. Guo, and M. Zhang, "Authorship attribution of source code by using back propagation neural network based on particle swarm optimization," *PloS one*, vol. 12, no. 11, p. e0187204, 2017.

[120] M. L. Brocardo, I. Traore, I. Woungang, and M. S. Obaidat, "Authorship verification using deep belief network systems," *International Journal of Communication Systems*, vol. 30, no. 12, p. e3259, 2017.

[121] P. Shrestha, S. Sierra, F. Gonzalez, M. Montes, P. Rosso, and T. Solorio, "Convolutional neural networks for authorship attribution of short texts," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, vol. 2, pp. 669–674, 2017.

[122] L. Wang, "News authorship identification with deep learning," 2017.

[123] G. Luppescu and F. Romero, "Comparing deep learning and conventional machine learning for authorship attribution and text generation,"

[124] C. Qian, T. He, and R. Zhang, "Deep learning based authorship identification."

[125] K. Luyckx and W. Daelemans, "The effect of author set size and data size in authorship attribution," *Literary and linguistic Computing*, vol. 26, no. 1, pp. 35–55, 2011.

[126] F. Rangel, P. Rosso, B. Verhoeven, W. Daelemans, M. Potthast, and B. Stein, "Overview of the 4th author profiling task at pan 2016: cross-genre evaluations," *Working Notes Papers of the CLEF*, 2016.

[127] E. Stamatatos, N. Fakotakis, and G. Kokkinakis, "Automatic text categorization in terms of genre and author," *Computational linguistics*, vol. 26, no. 4, pp. 471–495, 2000.

[128] H. Van Halteren, "Linguistic profiling for author recognition and verification," in *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, p. 199, Association for Computational Linguistics, 2004.

[129] M. Koppel, J. Schler, and E. Bonchek-Dokow, "Measuring differentiability: Unmasking pseudonymous authors," *Journal of Machine Learning Research*, vol. 8, no. Jun, pp. 1261–1276, 2007.

[130] H. J. Escalante, M. Montes, and L. Villaseñor, "Particle swarm model selection for authorship verification," in *Iberoamerican Congress on Pattern Recognition*, pp. 563–570, Springer, 2009.

[131] M. Koppel and Y. Winter, "Determining if two documents are written by the same author," *Journal of the Association for Information Science and Technology*, vol. 65, no. 1, pp. 178–187, 2014.

[132] S. Argamon and P. Juola, "Overview of the international authorship identification competition at pan-2011.," in *CLEF (Notebook Papers/Labs/Workshop)*, 2011.

[133] P. Juola and E. Stamatatos, "Overview of the author identification task at pan 2013.," in *CLEF (Working Notes)*, 2013.

[134] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[135] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[136] P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[137] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[138] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," *IEEE transactions on neural networks*, vol. 11, no. 3, pp. 697–709, 2000.

[139] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[140] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, p. 34, 2001.

[141] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CoRR*, vol. abs/1202.2745, 2012.

[142] T.-H. Wen, M. Gasic, N. Mrksic, P.-H. Su, D. Vandyke, and S. Young, "Semantically conditioned lstm-based natural language generation for spoken dialogue systems," *arXiv preprint arXiv:1508.01745*, 2015.

[143] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[144] T. Natschläger, W. Maass, and H. Markram, "The" liquid computer": A novel strategy for real-time computing on time series," *Special issue on Foundations of Information Processing of TELEMATIK*, vol. 8, no. LNMC-ARTICLE-2002-005, pp. 39–43, 2002.

[145] J. Kuwabara, K. Nakajima, R. Kang, D. T. Branson, E. Guglielmino, D. G. Caldwell, and R. Pfeifer, "Timing-based control via echo state network for soft robotic arm," in *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pp. 1–8, IEEE, 2012.

[146] P. G. Plöger, A. Arghir, T. Günther, and R. Hosseiny, "Echo state networks for mobile robot modeling and control," in *RoboCup 2003: Robot Soccer World Cup VII*, pp. 157–168, Springer, 2004.

[147] E. A. Antonelo, B. Schrauwen, X. Dutoit, D. Stroobandt, and M. Nuttin, "Event detection and localization in mobile robot navigation using reservoir computing," in *Artificial Neural Networks–ICANN 2007*, pp. 660–669, Springer, 2007.

[148] E. Antonelo, B. Schrauwen, and D. Stroobandt, "Mobile robot control in the road sign problem using reservoir computing networks," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 911–916, IEEE, 2008.

[149] F. Wyffels and B. Schrauwen, "A comparative study of reservoir computing strategies for monthly time series prediction," *Neurocomputing*, vol. 73, no. 10, pp. 1958–1964, 2010.

[150] P. Coulibaly, "Reservoir computing approach to great lakes water level forecasting," *Journal of hydrology*, vol. 381, no. 1, pp. 76–88, 2010.

[151] A. Ferreira, T. Ludermir, R. de Aquino, M. Lira, and O. Neto, "Investigating the use of reservoir computing for forecasting the hourly wind speed in short -term," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 1649–1656, June 2008.

[152] X. Lin, Z. Yang, and Y. Song, "Short-term stock price prediction based on echo state networks," *Expert Systems with Applications*, vol. 36, no. 3, Part 2, pp. 7313 – 7317, 2009.

[153] N. Schaetti, R. Couturier, and M. Salomon, "Reservoir computing: Étude théorique et pratique en reconnaissance de chiffres manuscrits, mémoire de master," 2015.

[154] N. Schaetti, M. Salomon, and R. Couturier, "Echo state networks-based reservoir computing for mnist handwritten digits recognition," *19th IEEE International Conference on Computational Science and Engineering (CSE 2016)*, 2016.

[155] A. Jalalvand, G. Van Wallendael, and R. Van de Walle, "Real-time reservoir computing network-based systems for detection tasks on visual contents," in *Computational Intelligence, Communication Systems and Networks (CICSyN), 2015 7th International Conference on*, pp. 146–151, IEEE, 2015.

[156] I. B. Yildiz, H. Jaeger, and S. J. Kiebel, "Re-visiting the echo state property," *Neural networks*, vol. 35, pp. 1–9, 2012.

[157] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke, "The microsoft 2017 conversational speech recognition system," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5934–5938, IEEE, 2018.

[158] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[159] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[160] J. Schmidhuber, F. Gers, and D. Eck, "Learning nonregular languages: A comparison of simple recurrent networks and lstm," *Neural computation*, vol. 14, no. 9, pp. 2039–2041, 2002.

[161] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in neural information processing systems*, pp. 545–552, 2009.

[162] J. Schmidhuber, D. Wierstra, and F. J. Gomez, "Evolino: Hybrid neuroevolution/optimal linear search for sequence prediction," in *Proceedings of the 19th International Joint Conferenceon Artificial Intelligence (IJCAI)*, 2005.

[163] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*, p. 89, Presses universitaires de Louvain, 2015.

[164] R. Jia and P. Liang, "Data recombination for neural semantic parsing," *arXiv preprint arXiv:1606.03622*, 2016.

[165] G. Weiss, Y. Goldberg, and E. Yahav, "On the practical computational power of finite precision rnns for language recognition," *arXiv preprint arXiv:1805.04908*, 2018.

[166] D. Britz, A. Goldie, M.-T. Luong, and Q. Le, "Massive exploration of neural machine translation architectures," *arXiv preprint arXiv:1703.03906*, 2017.

[167] R. S. Ginzburg, S. S. Khidekel, G. Y. Knyazeva, and A. A. Sankin, *A course in modern English lexicology*. Higher School Publishing House, 1966.

[168] H. Jaeger, *Short term memory in echo state networks*, vol. 5. GMD-Forschungszentrum Informationstechnik, 2001.

[169] J. Boedecker, O. Obst, J. T. Lizier, N. M. Mayer, and M. Asada, "Information processing in echo state networks at the edge of chaos," *Theory in Biosciences*, vol. 131, no. 3, pp. 205–213, 2012.

[170] A. Basile, G. Dwyer, M. Medvedeva, J. Rawee, H. Haagsma, and M. Nissim, "Is there life beyond n-grams? a simple svm-based author profiling system," in *Working Notes of CLEF 2017-Conference and Labs of the Evaluation Forum*, 2017.

[171] F. Rangel, P. Rosso, M. Potthast, and B. Stein, "Overview of the 5th author profiling task at pan 2017: Gender and language variety identification in twitter," *Working Notes Papers of the CLEF*, pp. 1613–0073, 2017.

[172] M. Martinc, I. Skrjanec, K. Zupan, and S. Pollak, "Pan 2017: Author profiling-gender and language variety prediction.," in *CLEF (Working Notes)*, 2017.

[173] E. S. Tellez, S. Miranda-Jiménez, M. Graff, and D. Moctezuma, "Gender and language-variety identification with microtc.," in *CLEF (Working Notes)*, 2017.

[174] B. Ganesh and A. Kumar, "Participation at the author profiling shared task at pan at clef.," 2017.

[175] N. Schaetti, "Unine at clef 2017: Tf-idf and deep-learning for author profiling.," in *CLEF (Working Notes)*, 2017.

[176] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[177] N. Schaetti, M. Salomon, and R. Couturier, "Echo state networks-based reservoir computing for mnist handwritten digits recognition," in *Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC), 2016 IEEE Intl Conference on*, pp. 484–491, IEEE, 2016.

[178] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *Journal of Machine Learning Research*, vol. 11, no. Mar, pp. 1109–1135, 2010.

[179] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Index