People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Ghardaia
Faculty of Science and Technology
Laboratory of Mathematics and Applied Sciences
Department of Mathematics and Computer Science

Master Thesis
Presented to obtain the Master diploma in Computer Science
Specialty: Intelligent Systems for Knowledge Extraction

Theme

Next Word Prediction Based On Deep Learning

Presented By
Ferialle LAHRACHE Sana DJEBRIT

Jury members:

Mr. Slimane BELLAOUAR	MCB	Univ. Ghardaia	President
Mr. KERRACHE Chaker Abdelaziz	MCB	Univ. Ghardaia	Examiner
Mr. MAHDJOUB Youcef	MAA	Univ. Ghardaia	Examiner
Mr. Slimane OULAD NAOUI	MCB	Univ. Ghardaia	Supervisor

University year: 2019 - 2020

ملخص

مع غزو الاجهزة الالكترونية جميع مجالات الحياة، يعتبر تسريع عملية ادخال المعلومات و تسهيلها امرا حتميا الان، برمجيا يمكننا المساهمة في ذلك عن طريق التنبؤ بالكلمة الموالية.

نناقش في هذا العمل الطرق الفعالة للتنبؤ بالكلمة الموالية، خصوصا مع ضخامة البيانات الحالية. يهدف هذا العمل الى تطبيق التعلم العميق على هاته الاشكالية، تحديدا الشبكات العصبية المتكررة و الشبكات التلافيفية المؤقتة، و مقارنة بسيطة بين نتائجهما.

نستعمل في هذا المشروع ثلاث قواعد بيانات: الاولى هي *CourseraSwiftkey* ، الثانية هي كتاب *WritingsofNietzsche : Volume*1 بقلم *Nietzsche Friedrich* و الثالثة هي فئة *news* من مجموعة *Brown* في مكتبة *nltk* . قمنا بمعالجتها، بعد ذلك ادخالها الى نماذج الشبكات العصبية المتكررة و الشبكات التلافيفية المؤقتة.

النتائج كانت مرضية بناءً على أحدث النتائج و بيئة العمل وقاعدة البيانات المستخدمة، حيث وصلنا الى دقة مقدارها 71.51% بالنسبة لنموذج الشبكة العصبية المتكررة و 65.20% بالنسبة لنموذج الشبكة التلافيفية المؤقتة باستخدام قاعدة البيانات الثالثة عند الاخذ بعين الاعتبار الثلاث كلمات السابقة للتنبؤ بالكلمة اللاحقة. نظرا للنتائج نستطيع ان نقول ان الشبكة التلافيفية المؤقتة تنافس الشبكة العصبية المتكررة في مجال نمذجة اللغة.

رغم اننا تحصلنا على نتائج مرضية، الا ان النتائج كانت لتكون افضل لولا عدم كفاءة الاجهزة و كذا محدودية بيئة العمل بسبب القيود المفروضة من قبل *GoogleColab* و *Kaggle* و كذا الظروف التي واجهناها و نحن في صدد انجاز هذا العمل بسبب جائحة فيروس كوفيد 19 . سنطور البحث في المستقبل باستعمال ارضيات توافق اكثر متطلبات التعلم العميق.

الكلمات الفتاحية: التنبؤ بالكلمة الموالية، الشبكة العصبية المتكررة، الشبكة التلافيفية المؤقتة، التعلم العميق، نمذجة اللغة، انظمة التنبؤ بالكلمات، معالجة اللغات الطبيعية.

Abstract

With the invasion of electronic devices all areas of our life, speeding up the process of entering information and facilitating it is now an imperative. We can contribute to this by predicting the next word.

In this work we discuss the effective methods of predicting the next word, especially given the magnitude of current data. This work aims to apply deep learning to this problem, specifically recurrent neural networks and temporal convolutional networks, with a primitive comparison between their results.

In this project we use three databases: the first one is Coursera Swiftkey, the second is the book: Nietzsche Writings: Volume1 by Friedrich Nietzsche and the third is the News category from the Brown corpus in the nltk library. We prepare and insert them into RNN and TCN models.

The results were satisfactory according to the state of the art results and the platform and data set used, as we reached an accuracy of 71.51% for the RNN model and 65.20% for TCN model using the third database when taking into account the three previous words to predict the next word. Given the results, we can say that the temporary convolutional network competes with the recurrent neural network in the field of language modeling.

Although we obtain satisfactory results, they would have been better had it not been for the inefficiency of the devices and the limited work environment due to the restrictions imposed by Google Colab and Kaggle, as well as the circumstances that we faced while we were in the process of completing this work due to the pandemic Covid19. We will develop the research in the future by using platforms that meet the requirements of most deep learning.

Key words: Next word prediction, Recurrent neural networks, Temporal convolutional networks, Deep learning, Language modeling, Word Prediction systems, Natural Language Processing.

Résumé

Avec l'invasion des appareils électroniques dans tous les domaines de notre vie, accélérer le processus de saisie des informations et le rendre plus facile est désormais un impératif. Par programmation, nous pouvons y contribuer en prédisant les mots suivants.

Dans ce travail, nous discutons des méthodes efficaces de prédiction du mot suivant, en particulier compte tenu de l'ampleur des données actuelles. Ce travail vise à appliquer l'apprentissage profond à ce problème, en particulier les réseaux de neurones récurrents et les réseaux convolutifs temporels, avec une comparaison très primitive entre leurs résultats.

Dans ce projet, nous utilisons trois bases de données: la première est Coursera Swiftkey, la seconde est le livre: Nietzsche Writings: Volume1 de Friedrich Nietzsche et la troisième est la catégorie News du corpus Brown de la bibliothèque nltk. Nous les préparons, puis les insérons dans des modèles RNN et TCN.

Les résultats ont été satisfaisants selon les résultats de l'état de l'art et la plate-forme et la base de données utilisés, car nous avons atteint une précision de 71,51% pour le modèle RNN et de 65,20% pour le modèle TCN en utilisant la troisième base de données en tenant compte des trois mots précédents pour prédire le mot suivant. Au vu des résultats, on peut dire que le TCN est en concurrence avec le RNN dans le domaine de la modélisation du langage.

Bien que nous ayons obtenu des résultats satisfaisants, ils auraient été meilleurs à cause de l'inefficacité des appareils, ainsi que l'environnement de travail limité en raison des restrictions imposées par Google Colab et Kaggle, ainsi que les circonstances auxquelles nous avons été confrontés alors que nous étions en train de terminer ce travail en raison de la pandémie de virus Covid 19. Nous développerons la recherche à l'avenir en utilisant des plateforms qui répondent aux exigences de la plupart des deep learning.

Mots clés: Prédiction du mot suivant, Réseaux de neurones récurrents, Réseaux convolutifs temporels, Apprentissage profond, Modélisation du langage, Systèmes de prédiction du mots, Traitement du langage naturel.

Contents

List of Figures

Acknowledgement

We would like to extend our thanks to the supervisor of this thesis, Mr. OULAD NAOUI Slimane, for his suggestion of this topic and his valuable guidance, and also for his efforts on teaching us in previous years with his wonderful way, thank you very much.

We warmly thank the jury members for accepting the evaluation of our work.

Also, we would like to extend our sincere thanks to our professor Mr. BELLAOUAR Slimane, who has always been a source of inspiration for us, and who we have much appreciation and gratitude for his teaching, guidance and encouragement, thank you really for your incredible efforts.

Finally, we address our big thanks to everyone who helped and encouraged us during the realization of this work.

Dedication

To my dear father, who raised me and supported me throughout my academic career and my life,"**Houcine**".
To my dear mother, who gave me all the support and love, and without her, I would not have become what I am today, "**Meriem**" and her familly "**lamour**".
To my sister "**Sara**", who stood beside me and supported me, and to my brothers "**Readouane**" and "**Mouhammed**".
To my partner and companion "**Sana**", whom I admire and cherish her friendship and I am proud to have done this work with her.
Of course, I do not forget my friends and colleagues throughout the various stages of my studies and those who are still with me and support me. I hope that our friendship will last in the future as well.
To all my neighbors and loved ones, whom I cherish so much, especially "**Sendess**".
Of course, I do not forget to mention the teachers who supported me, encouraged me and believed in me since the beginning of my studies until now, and whom I thank very much for being part of my success.
Let me also thank the "**MjnAch**" group and the "**Little Coder**" group who I benefited from them and will continue to benefit from them in the future and whom I am proud to be a part of, and I wish success to all members of both groups.
And last but not least, I thank God Almighty for my life and this journey that I am proud of.

LAHRACHE FERIALLE

Dedication

First of all, I would like to thank Allah who made us complete this work, then

I want to dedicate this work
To my refuge in this life, my role model and my inspirational source, my father who no words can fulfill my gratitude for him 'Abdelhamid'
To my soulmate, my idol, the source of my joy and hopes, my mother who no words can describe my love for her 'Zineb'
To parts of my soul whose belive in me and encourages me always, my brothers: my lovely 'Wafa', my precious 'Abderahmane', my little 'bassem' and my princess 'Alaa'
To my grandmothers whose giving me a push when i lose hope by their prayers; may Allah prolong their life
To my grandpa who instilled in us the love of seeking knowledge 'Mohamed', may Allah prolong his life, and to the soul of my other grandpa 'Ahmed' may Allah have mercy on him
To my dear aunts and uncles for their support
To my lovely partner in this work 'Feryel' who i appreciate her kindness and encouragement, I am grateful to have had this experience with you, may Allah open the doors of success for you and making your dreams reality
To all teachers, in all different levels, whose teached me, believed in my capacities and encouraged me, all my gratitude
To all my dear cousins, friends and all those i love
To our special class 2015/2020
and because we keeping the best for last, I want to wrap up with them,
To my besties whose i was with this recent years in the same class with the same circumstances, i spent with you the most amazing time together, members of MjnAch and InstMjnAch groups, may Allah achieve what you want.

DJEBRIT Sana

Introduction

With the big data revolution, old problems demanded new ways to solve them, and most especially, we find Deep Learning (DL). One of the domains affected by this revolution is Natural Language Processing (NLP), where with the rapid development and proliferation of social media and websites we have a massive amount of data that can help us to achieve good results in this field. One of NLP sub-domains is Language Modeling (LM) which includes the task of word prediction.

With the technological development and the great use of electronic devices, the search for ways to speed up the process of entering information became an imperative, as the beginnings were since the emergence of augmentative and alternative communication systems, as well as word prediction systems since 1980. To face the next word prediction problem, the beginnings were with using statistical and probabilistic methods, then combined with knowledge based models, then heuristic modeling. After the immediate surge in data, machine learning techniques stepped in to suggest good solutions and finally, with the strong return of neural networks because of its revolutionary results with big data, deep learning models are sweeping the field and satisfy the objectives in this task, that's why we chose them in our work to solve the problem.

In this thesis we present DL models that predict the next word using Temporal Convolutional Network (TCN) and Recurrent Neural Network (RNN) and compare their results, as we train them using three different data sets as we reached an accuracy of 71.51% for the RNN model and 65.20% for TCN model using the third database when taking into account the three previous words to predict the next word.

Our thesis consists of three chapters:

In the first one, we present the concept of deep learning in which we state its history, contents, and basic principles and architectures.

At the second chapter we represent the state of the art of next word prediction, covering several concepts such as NLP, Language Modeling (LM), word prediction systems, factors affecting prediction, and the different prediction methods from classical to machine learning methods, and finally coming to DL models.

The last chapter represents the steps and architectures proposed to solve the problem of Next Word Prediction (NWP) using TCN/RNN.

And finally, we end with a conclusion that summarizes all of the above.

Chapter 1

From Artificial Intelligence to Deep Learning

1.1 Introduction

Artificial Intelligence (AI) is considered one of the most important fields of science and engineering that many researchers have been interested in. This field includes many branches the most important is Machine Learning (ML). With the rapid development of technology and the increase in data volume, the focus has increased on the field of ML which has become an important part of many applications such as image classification, natural language processing, video recommendation, text extraction and many other applications.

Recently, a sub field of ML called Deep Learning (DL) has appeared and became very popular because it deals with algorithms that mimic the way the human brain works.

In this chapter, we highlight the effectiveness of DL starting from its origins. We present a brief reminding about AI and ML, fundamentals of neural networks (NN), then we present some incentives that led to heading into the field, the fields where DL application make a huge positive results. After that, we present the most popular architectures and learning algorithms of the field. Finally, we give an overview about generalization, regularization and optimization techniques.

1.2 Artificial Intelligence

The humans have always wanted to make a machine equivalent to their intelligence. The beginning was with Turing Machine in 1936 by Alan Turing. In 1943, the computer appeared and evolved, at the same time humans become more and more eager to acquire a machine closer to them in intelligence. So they tried to make the machine solve problems automatically by adding techniques that use logic, if-then rules and so on.

Artificial intelligence can be defined as a set of techniques that allow the machine to simulate a human, in other word, to perform operations and solve problems that humans or some animals usually solve (Lecun [2016]).

Artificial intelligence, in its definition, is divided into four parts (Russell and Norvig [2010]):

Thinking Humanly

"The exciting new effort to make computers think ... machines with minds, in the full and literal sense." (Haugeland [1989])

Thinking Rationally

"The study of mental faculties through the use of computational models." (Charniak and McDermott [1985])

Acting Humanly

"The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil et al. [1990])

Acting Rationally

"Computational Intelligence is the study of the design of intelligent agents." (Poole et al. [1998])

With the huge increase in the amount of data, the problems have become more complex and the solutions provided by artificial intelligence became unhelpful in making the machine provide the appropriate decisions. This led to the emergence of machine learning.

1.3 Machine Learning

Machine Learning, according to Arthur Samuel, is the scientific specialty that focuses on making computers able to learn without programming them properly. Sometimes the correct form of doing things is costly and inaccurate (Samuel [1959]).

Tom Mitchell gives a modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." (Mitchell [1997])

Machine learning algorithms are classified according to the desired result into several types, here we explain some common types mentioned in Figure 1.1:

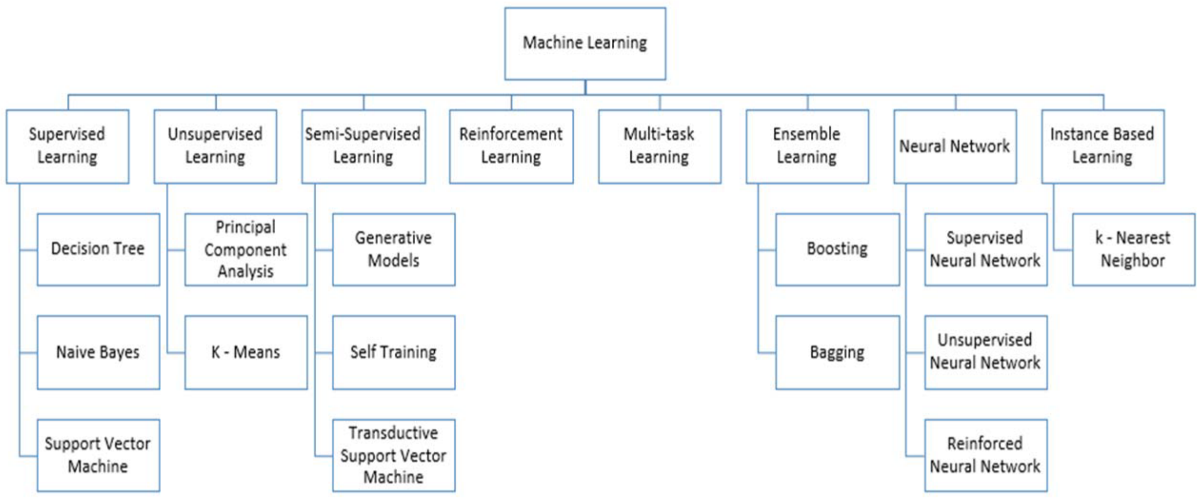


Figure 1.1: Types of Learning (Dey [2016])

1.3.1 Supervised Learning

Supervised machine learning algorithms are algorithms that require a human supervisor. We have an input data set divided into training set and test set where the data set contains labeled data, i.e both input X and the desired output Y, and the algorithm try to reach Y using X without any idea of the relation between them and the learned pattern from the training set is applied on test set for regression which means predicting the output which has a continuous value and classification which means classifying the input into specific class or group where the output has a discrete value (Dey [2016]).

1.3.2 Unsupervised Learning

Unlike supervised machine learning algorithms, Unsupervised Machine Learning algorithms do not require a human supervisor. The input data set contain unlabeled data, i.e only the input X, and the output Y is not known, we have no idea about what the result should be and we try to learn features from the data and find relations between the inputs to build clusters in order to classify data and it is also used in dimensionality reduction (Dey [2016]).

1.3.3 Semi-Supervised Learning

In semi-supervised learning we use both labeled and unlabeled examples or data to generate a function or classifier (Ayodele [2010]). This type helps to improved accuracy and also facilitates the classification of data because it allows to classify unlabeled data with the use of a knowledge from some labeled data.

1.3.4 Reinforcement Learning

In this type of learning, we do not know the correct output of a specific input where the algorithm performs actions that have an impact on the environment. In return, the environment provides notes that direct the learning algorithm and measure the quality of the output, where whenever the environment is given a reward, the higher the quality of the result presented by the algorithm (Lison [2015]).

1.4 Neural Networks

Artificial Neural Network (NN) is one of the most important machine learning techniques. It allows the grateful training of models by extracting features by experiences and not by defined rules. Artificial NN were developed by observing how the human neural network work and trying to build a mechanism that simulates this process. In the following, we provide a brief review the most important NN notions.

1.4.1 Biological Origin

The human brain contains millions of neurons that are linked together, forming neural networks which are also called connectionist models. Its function is to transmit information that is processed in the cerebral cortex, where the cognitive functions are language, thinking, learning and memory, which are among the most complex brain operations to define in terms of neural mechanisms (Du and Swamy [2013]). Neurons are the main structure of the nervous system where they connect to each other using so-called axons and the areas between the axons and dendrites are linked with synapses. Synapses are structures that acts like a path way connection to transmit signals to other cells. The strength of this cross-linking between neurons changes in response to external stimuli, this change is what leads people to what is called learning which is the function that artificial neural networks try to emulate (Aggarwal [2018a]).The Figure 1.2 shows the structure of neurons and how two neurons connect to each other, where the Figure 1.3 shows a formal neuron.

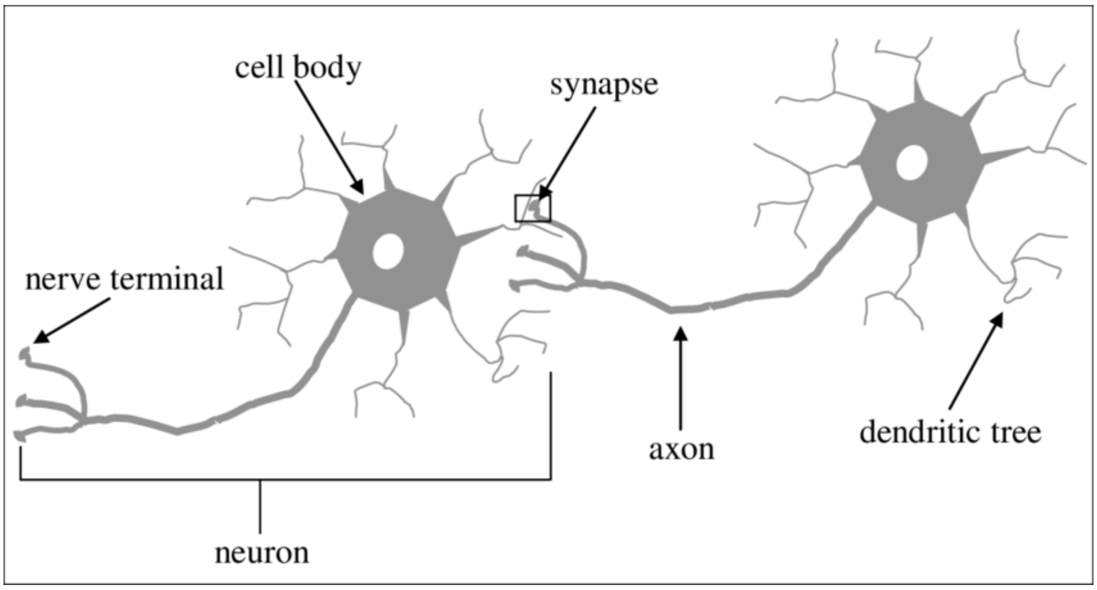


Figure 1.2: Biological Neuron (Smith et al. [2007]).

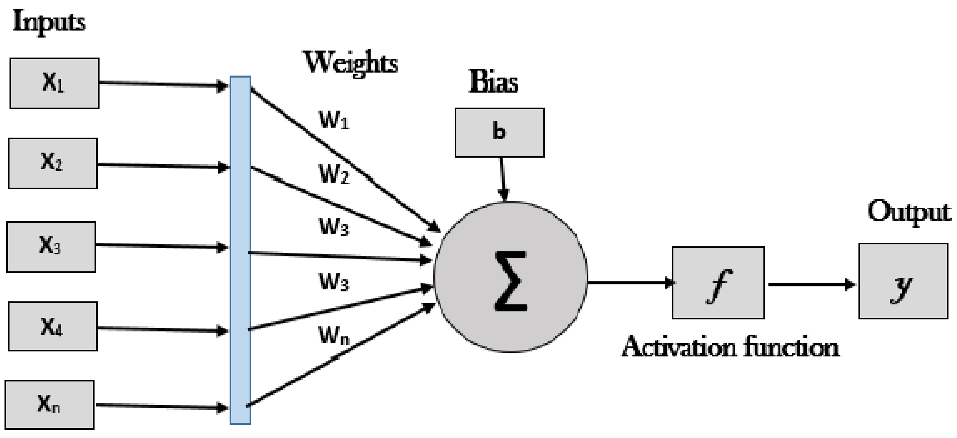


Figure 1.3: formal Neuron (Abdessamad et al. [2015]).

1.4.2 Principal Architectures

The neuron is a basic processing unit in a neural network where it was achieved using a simple amplifier and the synapse is achieved using a resistor. The neurons processes the information entered into them and produce an output by applying a function usually called an activation function which we will explain later while the synapses that connect the neurons to each other are modeled with weights (Du and Swamy [2013]).

The perceptron, also known as single layer perceptron or single layer NN, is the simplest architecture of neural networks that was used initially then the multi layer perceptron or multi layer NN emerged after that in recent years deep neural networks appeared (Du and Swamy [2013]). We will provide in the following sections a brief explanation of these architectures

The Perceptron

The Perceptron or single computational layer is the simplest NN invented in 1957 by Frank Rosenblatt Rosenblatt [1958] based on the neuron model proposed by McCulloch and Pitts [1943] (Hayman [1999]). It is a linear classifier for binary predictions that consists of input layer which is a $\mathcal{K}$ nodes that receive the $\mathcal{K}$ input information (features) and each node have a proper weight that presents its effectiveness in the process; so it transmits the $\mathcal{K}$ features X = $[x_1 \dots x_k]$ with the $\mathcal{K}$ weights W = $[w_1 \dots w_k]$ to the output node in order to compute the linear function in 1.1

$$W \cdot X = \sum_{i=1}^{k} w_i \cdot x_i \tag{1.1}$$

Then the predicted value $\hat{Y}$ is computed using an activation function which takes the value $W \cdot X$ as a parameter:

$$\hat{Y} = f(\sum_{i=1}^{k} w_i \cdot x_i) \tag{1.2}$$

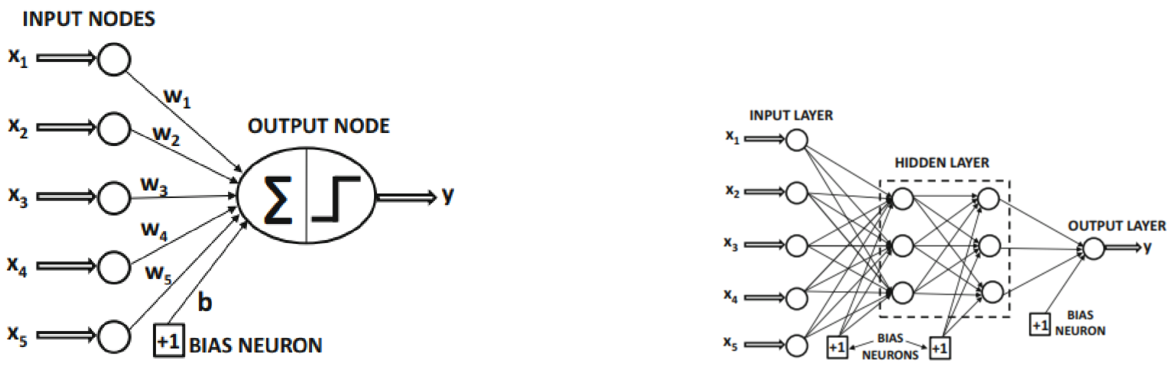
Sometimes we need to add a node to the input layer that represents the invariant part in the prediction, named “bias”. We need it when the binary class distribution is highly imbalanced. The value of the bias node is always equal to 1, the weight of this neuron provides the bias variable, as if it represents the threshold of latency in the biological neural synapses. So the function will be (Aggarwal [2018a]):

$$\hat{Y} = (W \cdot X + b) = f(\sum_{i=1}^{k} w_i \cdot x_i + b) \tag{1.3}$$

The preceprton architecture is shown in Figure 1.4a.

Multi-layer Neural Networks

Multi-layer NN consist of multiple computational layers where each layer feed the next one. That is what leads to name this architecture as “feed-forward networks”. Each input node pass the value x_i and the weight w_i to the first hidden layer, then each node make computation using an activation function and pass the result to the next layer and so on until we reach the output layer that makes the expected result (Aggarwal [2018a]). The multi-layer NN architecture is shown in Figure 1.4b.



(a) Perceptron (b) multi layer NN

Figure 1.4: The basic architectures: Perceptron and multi layer NN (Aggarwal [2018a]).

Deep Neural Networks

A deep NN is a more complex architecture than its predecessor as it was called deep because it contains many hidden layers. We will present some deep neural networks in the coming sections.

Activation Function

The activation function converts the signal coming from the neurons to an expected result, where the bias can adjust up or down the function, which leads to greater learning opportunities for the network (Aggarwal [2018a]).

There are many activation functions as shown in Figure 1.5, each one is appropriate in a given type of problem.

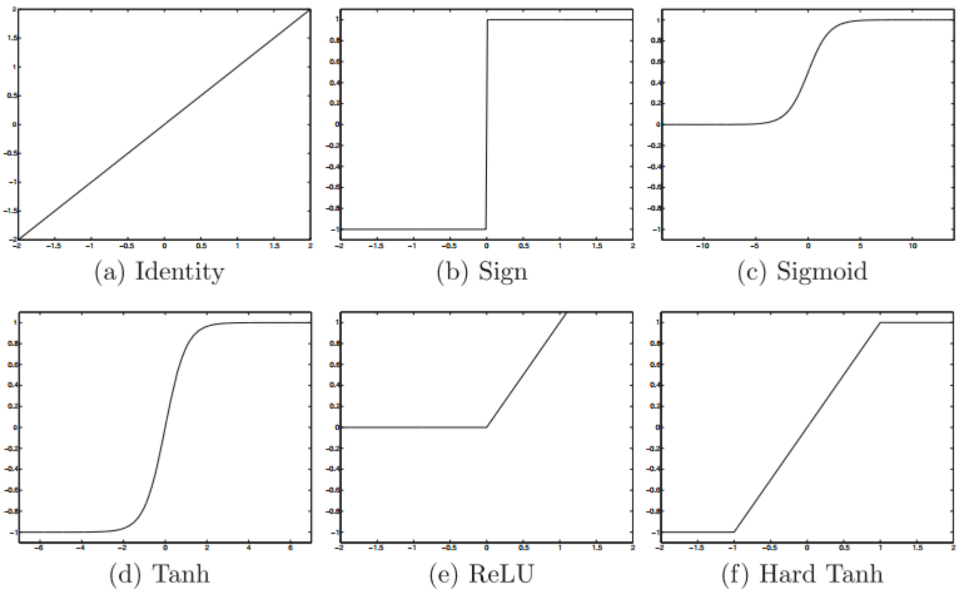


Figure 1.5: Various activation functions (Aggarwal [2018a]).

- The sign function can be used in case of binary outputs

$$f(v) = sign(v) \tag{1.4}$$

- Whereas the sigmoid function is helpful in performing computations that should be interpreted as probabilities

$$f(v) = segmoid(v) = \frac{1}{1 + e^{-v}} \tag{1.5}$$

- Besides tanh function is helpful when the calculation outputs are required to be positive and negative

$$f(v) = \tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \tag{1.6}$$

- ReLU and hard tanh activation functions have replaced sigmoid and tanh activation functions in modern neural networks because they make the multi-layered neural networks easy to train (Aggarwal [2018a]).

- There is another activation function called softmax which is used when we want our output vector to be a probability distribution over a set of mutually exclusive labels. The output of a neuron in a softmax layer depends on the outputs of all the other neurons in its layer (Buduma and Locascio [2017]), as converges the equation 1.7

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{1.7}$$

  A strong prediction would have a single entry in the vector close to 1, while the remaining entries were close to 0 (Buduma and Locascio [2017]).

1.4.3 Learning algorithms

The important phase is the training of the neural network to make it capable to learn. The neuron can make sense from the incoming parameters and produce a result. That's what the learning algorithms do.

Learning algorithms are algorithms that work to find appropriate weights and/or other network parameters in order to minimize a cost function. Usually, neural networks are trained by epoch. An epoch means presenting training data to the network and processing them with the training algorithm only once (Du and Swamy [2013]).

As mentioned earlier, learning methods are traditionally divided into supervised, unsupervised and reinforcement learning.

In supervised learning, the output is known so we change the network parameters according to the result of the comparison between the true and the obtained output.

The most popular and effective supervised training algorithm is the back propagation gradient developed later.

In the case of unsupervised learning the output is not known, we try to link information from the inputs and find links between them to find common patterns or features. In this type of learning, we need a criterion to end the learning process. If there is no criterion, the process continues even when a result is presented. The three well known approaches in this type of learning are: Hebbian learning, competitive learning, and the Self Organizing Maps (SOM) (Du and Swamy [2013]).

Reinforcement learning is a special case of supervised learning, where the output is unknown and the agent learn what kind of actions he had to make in order to maximize the total expected reward. It rewards the NN for a good output result and punishes it for a bad one (Du and Swamy [2013]).

Back Propagation Gradient

The principle of this algorithm is to calculate the gradient error between the supposed result and the result at the actual output where the gradient is a vector calculated using partial derivations of a function dependent on more than one variable, then to adjust the synaptic weights of the previous layers by the back-propagation of the error value. The derivative is useful for minimizing the error function because it help us to know how we can change X in order to make an improvement in Y (Goodfellow et al. [2016]). The operation is repeated until we reach the defined number of iteration or meet the validation rate as shown in Figure 1.6 (Khacef et al. [2018]).

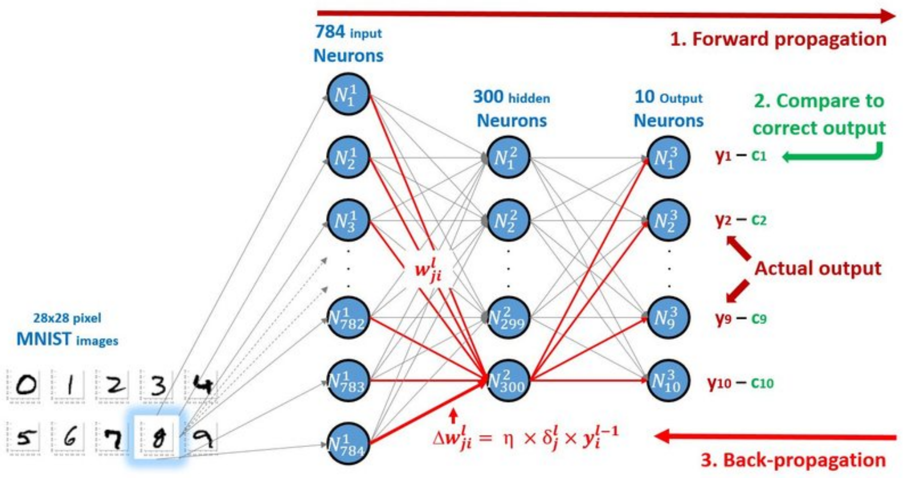


Figure 1.6: Gradient back propagation learning algorithm steps (Khacef et al. [2018])

The algorithm can be resumed in the next steps.

Forward phase

- Initialize the value of the input layer from the data set.
- Initialize the weights of neurons randomly.
- Get the result of the output layer.
- Calculate the gradient error E with the equation.

$$E = \frac{1}{2}\sum_c \sum_j (y_{j,c} - d_{j,c})^2 \tag{1.8}$$

Where c is the index over cases (input-output pairs), j is an index over output units, y the actual output and d is the desired output (Rumelhart et al. [1986]).

Backward phase

- Compute the partial derivation of E in order to minimize it by the gradient decent.

$$\frac{\partial E}{\partial y_j} = y_j - d_j \tag{1.9}$$

$$\frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j}.\frac{dy_j}{dx_j} = \frac{\partial E}{\partial y_j}.y_j(1 - y_j) \tag{1.10}$$

- Change weights according to error:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial x_j}.\frac{\partial x_j}{\partial w_{ji}} = \frac{\partial E}{\partial x_j}.y_j \tag{1.11}$$

And for the output of the i^{th} unit:

$$\frac{\partial E}{\partial x_j}.\frac{\partial x_j}{\partial y_i} = \frac{\partial E}{\partial x_j}.w_{ji} \tag{1.12}$$

Taking into account all the connections we got:

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial E}{\partial x_j}.w_{ji} \tag{1.13}$$

- Finally we collect the partial derivatives on all training examples in our data set:

$$\Delta w = -\epsilon \frac{\partial E}{\partial w}(1.14)$$

This method does not converge as quickly as methods that use the second derivatives, but it is much simpler (Rumelhart et al. [1986]).

1.4.4 Advantages and Disadvantages of NNs

Neural networks have many advantages and disadvantages we mention among them (Zou et al. [2009], Walczak and Cerpa [2003], Mijwel [2018])

Advantages

- Model non-linear and complex problems.
- Can be generalize and predict invisible data.
- Able to perform more than one function simultaneously.
- Information storing in the entire network and not in a database so also if few data disappear, the network stay functioning. NN are fault-tolerant systems.

Disadvantages

- The requirement of powerful processing devices.
- When the network produces a result, it doesn't give any idea about how or why, which reduce the trust (black boxes).
- The choice of the right network structure is hard, because it is achieved through experience and trial and error.

1.4.5 Typical applications

Deep learning is currently one of the most common and widespread fields of research, which has led to its use in many applications. We mention in the following sections some of the most prominent.

Natural Language Processing

NLP is a special field that contains several algorithms and techniques which focus on how to train the machine to understand the human language, which is a challenge due to the complexity, ambiguity, and diversity of the human languages.

The solutions performed by NLP have achieved great development and have proven an effectiveness in achieving high accuracy in their tasks. Here are some of these leading learning solutions (Pouyanfar et al. [2018]):

- Sentiment Analysis: Is a branch of NLP which deals with text analysis and classification of the writer's feeling or opinion. Most data sets for sentiment analysis are classified as either positive or negative, and neutral expressions are removed by personality classification methods.

- Machine Translation: Deep learning plays an important role in improving the methods and approach of machine translation. One of the most famous and used structures in this field is RNN (see later), where the structure of RNN-based encoding and decoding was introduced in neural machine translation.

Visual data processing

We can say that deep learning techniques have become the main parts of media systems and computer vision, as the famous convolutional neural network architecture showed important results in simulations of real-world tasks, including image and video processing and object detection.

1.5 Deep Neural Networks

The great effectiveness of deep neural networks comes from their ability to extract high-level features from the data presented to them after using statistical learning on a large group to obtain an effective representation. This differs from the previous result that uses handmade features and as the name shows, there are many layers that increase their effectiveness (Sze et al. [2017]). An example of a simple deep neural network is shown in Figure 1.7.

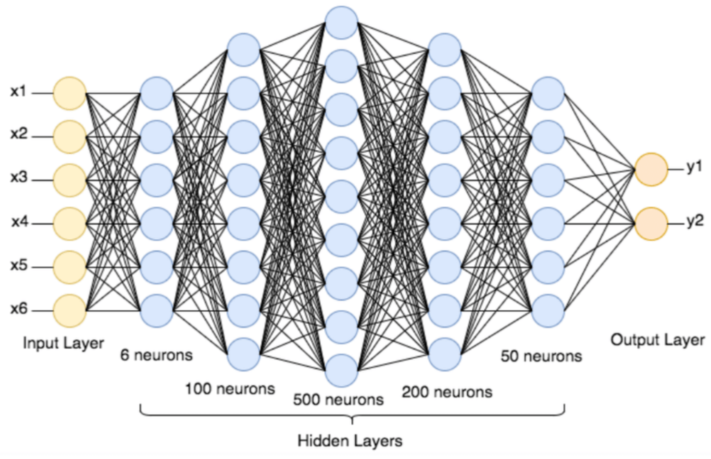


Figure 1.7: Deep neural network (Bahi and Batouche [2018]).

1.5.1 Emergence of deep learning

The modern history of deep learning started in 1943 when the McCulloch-Pitts (MCP) model was introduced and became known as the prototype of artificial neural model.

After the MCP model, the Hebbian theory, originally used for the biological systems in the natural environment, was implemented (Pouyanfar et al. [2018]).

After that, the first electronic device called "perceptron" within the context of the cognition system was introduced in 1958, though it is different from typical perceptrons nowadays (Pouyanfar et al. [2018]).

In 1980, the "neocogitron" was introduced, it inspired the convolutional neural network (Pouyanfar et al. [2018]).

In 1986 the Recurrent Neural Networks (RNNs) were proposed (Pouyanfar et al. [2018]).

Next, in the 1990s LeNet made the Deep Neural Networks (DNNs) work practically (Pouyanfar et al. [2018]).

Around 2006, Deep Belief Networks (DBNs) with a layer-wise pretraining framework were developed (Pouyanfar et al. [2018]).

Google AlphaGo is an inspirational application of deep learning, which shocked the world at the start of 2017. It won 60 online games Under the pseudonym name "master" in a row against human professional Go players, including three victories over Ke Jie, from December 29, 2016, to January 4, 2017 (Pouyanfar et al. [2018]).

Deep learning has imposed its effectiveness on solving problems that have been unable to solve the best attempts of artificial intelligence algorithms for years. This became evident from its effectiveness with the complex dilemmas of high-dimensional data and thus its effectiveness in many fields: such as the field of images, sound, chemistry, bio-informatics and especially the understanding of natural languages, where it achieved surprising and very promising results (LeCun et al. [2015]).

There are many deep networks, we will present next an overview about the most famous one.

1.5.2 Convolutional Neural Networks

Convolutional neural networks (CNN) is one of the most famous deep neural networks and the most widely used, especially in the field of computer vision, where its structure simulates the way the visual cortex works in the cat's brain (Pouyanfar et al. [2018]).

It is designed to work with grid-structured inputs, which have strong spatial dependencies in local regions of the grid (Aggarwal [2018a]). In other word the input is multiple arrays, for example a color image composed of three 2D arrays containing pixel intensities in the three color channels (LeCun et al. [2015]). Despite the focus of CNNs on image data processing, we can also consider sequential data such as texts as a special case of grid structured data and use CNN to process it (Aggarwal [2018a]). This has already been done and has shown high efficiency, especially in the field of natural language processing.

There are four key ideas behind CNNs: local connections, shared weights, pooling and the use of many layers which makes the network more effective (LeCun et al. [2015], Pouyanfar et al. [2018]).

The CNNs consist of several consecutive layers where the output of a layer is the input to the next layer, these layers are known to be repeatable where each of them has a specific function and when repeated it gives better results.

Figure 1.8 shows the structure of a CNN :

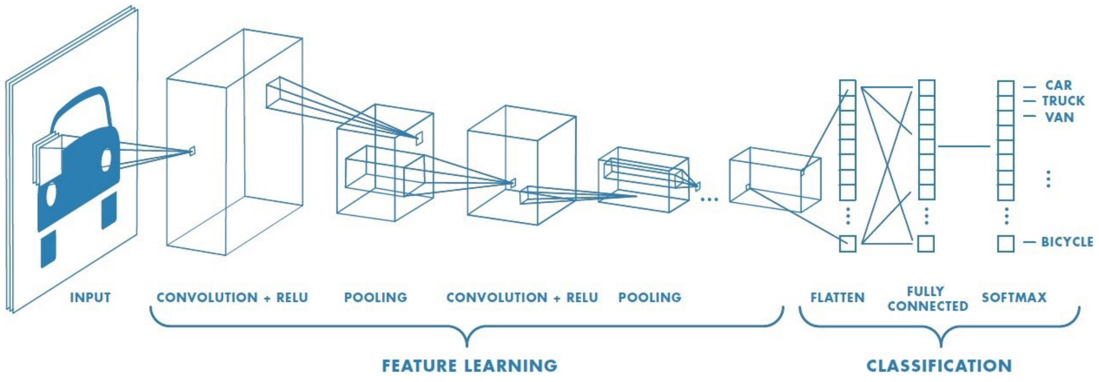


Figure 1.8: Convolutional Neural Network (Kulkarni and Shivananda [2019]).

The previous figure shows that CNN contains a convolutional layer followed by an activation function then the pooling layer comes usually max pooling layer. Convolutional and pooling layers can be done many times after that comes the flattening that gives us vector presented as input to the fully connected layer that works like a traditional NN and produces the final output. Here is an explanation about the different layers.

Convolutional layer

The convolutional layer takes inputs and applies to each position a filter, the result is called a feature map. A specific number of filters are applied resulting many feature maps.

The units in this layer are organized in feature maps, where each unit communicates with the previous layer via a set of weights called the filter bank (LeCun et al. [2015]). After getting the result we apply ReLU activation function which does not change the dimension. The result is a 3-dimensional grid structure which has a height, width and depth (Aggarwal [2018a]). The depth refer to the number of future maps. The Figure 1.9 shows the input and output future maps in the convolutional layer.

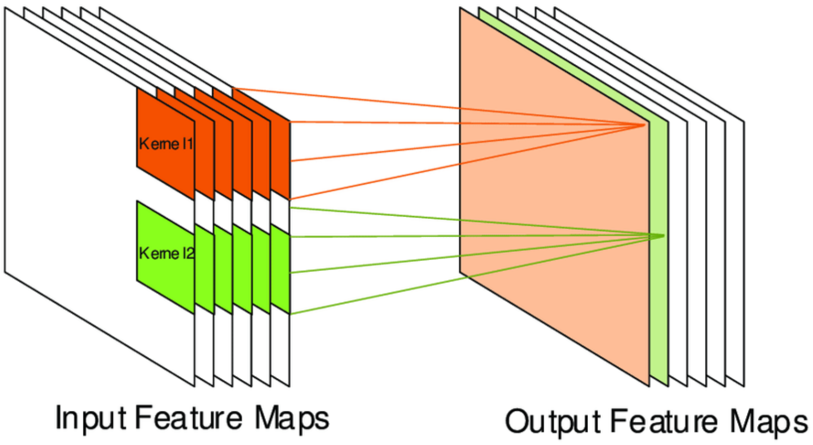


Figure 1.9: Convolutional layer (Véstias [2019]).

The function of this layer can be summarized in extracting features and applying filters.

Pooling layer

Pooling means doing a variety of computations to reduce the dimension of feature maps. To do so, we include another type of layer called pooling (Buduma and Locascio [2017], Sze et al. [2017]). There are three types of pooling:

- Max pooling: We take the maximum value of the pixels in a specific range.
- Min pooling: We take the minimum value of the pixels in a specific range.
- Average pooling: We take the average value of the pixels in a specific range.

To preserve the important features we use the Max-pooling, where each feature map is divided into squares of equal size, then an intense feature map is created. We take the maximum value in every square and produce a condensed feature map (Buduma and Locascio [2017]).

This process is illustrated in Figure 1.10

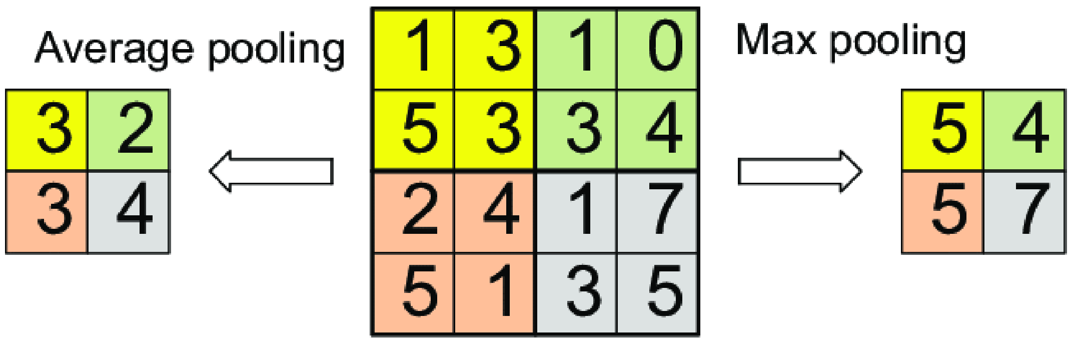


Figure 1.10: Average and Max-Pooling (Véstias [2019]).

Flattening

We need to convert the output of previous layers into one dimension feature vector to be used by next layer, here where comes the turn of flattening which do this as shown in Figure 1.11

Fully Connected layer

In this layer we connect each feature in the final spatial layer with each hidden state in the first fully connected layer. We can say that its structure is similar to the traditional feeding network. To increase the strength of calculations we can use more than one continuous layer. The majority of parameters lie in these fully connected layers due to the big number of connections (Aggarwal [2018a]).

Figure 1.11 shows the flattening operation followed by the fully connected layers and output layer that gives the output of the CNN.

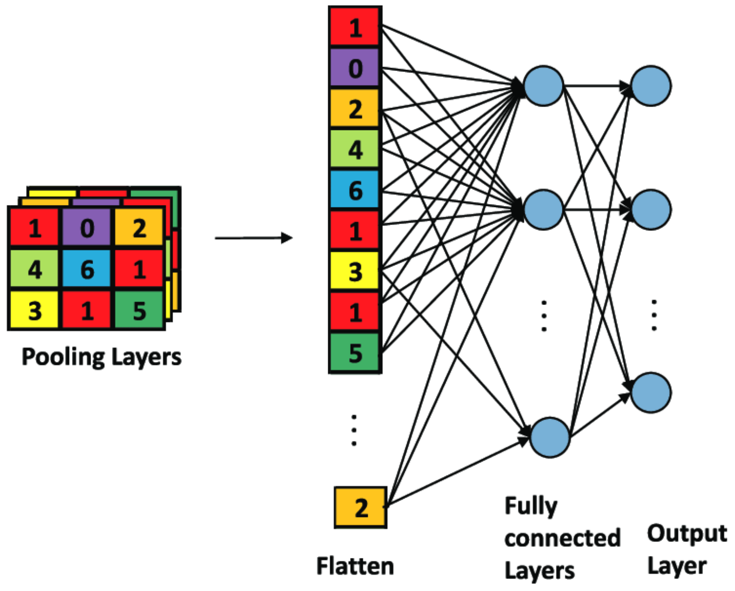


Figure 1.11: Flatten and fully connected layers (Ng et al. [2019]).

1.5.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are among the most famous deep neural networks that simulate the repetitive brain structure. They have been proposed in 80's and specialize in the sequential domain. RNN is characterized by its unique structure which is different from the rest of the networks where it has feedback connections (Du and Swamy [2013]), which makes it one of the best ways to solve problems that need previous knowledge or experience such as prediction and production (Bullinaria [2013]), it is also used in cases where the input has a variable of unfixed length.

RNNs process the entry sequence each time while keeping the state vector in its hidden units since the latter contains information about the history of all previous elements of the sequence (LeCun et al. [2015]).

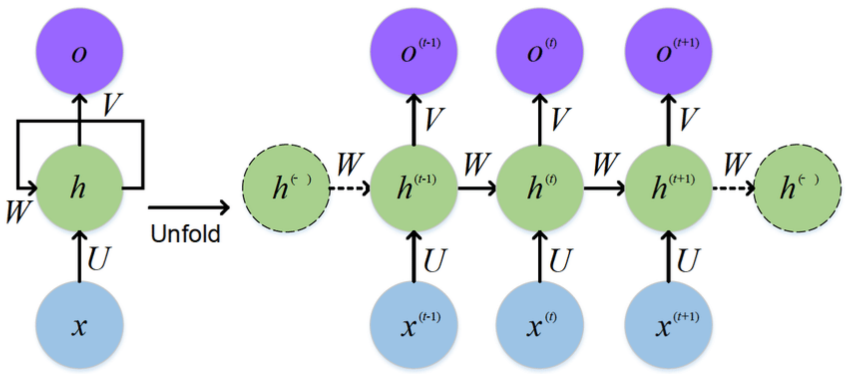


Figure 1.12: Recurrent Neural Network (Feng et al. [2017]).

As shown in Figure 1.12 the output of the hidden layer is its input and this operation is repeated for a specific times in different time steps. The weights are shared at each time step.

The hidden state at time t is given by a function of the input state at time t and the hidden state at time t-1 (Aggarwal [2018a]):

$$h_t = f(h_{t-1}, x_t) \tag{1.15}$$

Note that we have input hidden weight W_{xh} and hidden layer hidden weight W_{hh} and the resulted output hidden weight W_{yh} and by using tanh as a function the equation became:

$$h_t = tanh(W_{hh}.h_{t-1}, W_{xh}.x_t) \tag{1.16}$$

and the output is:

$$y_t = W_{yh}.h_t \tag{1.17}$$

As RNN can process inputs of different lengths and generate outputs of different lengths it has different representations as shown in Figure 1.13 (Subramanian [2018]):

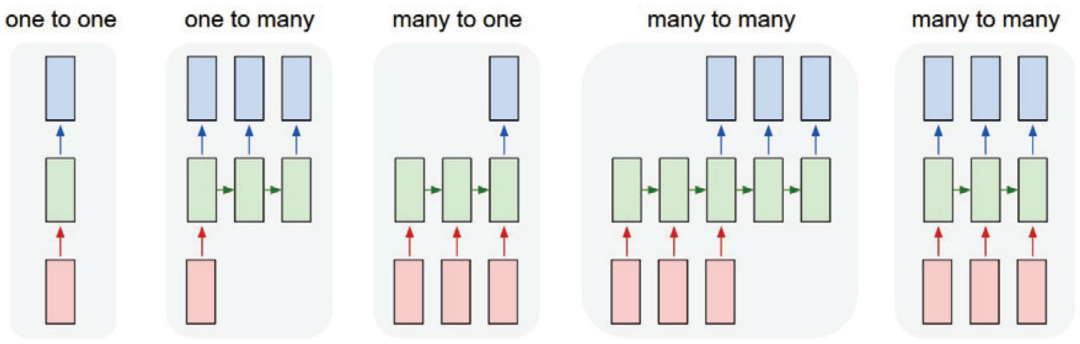


Figure 1.13: RNN representations (Subramanian [2018]).

- One to one: Used for fixed size input to fixed size output and it is appropriate for image classification.
- One to many: The output is a sequence and it is used for image captioning or music generation.
- Many to one: The input is a sequence and it is used for sentiment analysis or classification.

- Many to many: There is two types, the first has a sequence input and used in machine translation where the second has a synchronized sequence input and output and it is used for video classification.

RNN Problems

We can look at RNN as very deep feedforward network, where all layers share the same weights and as we mentioned earlier, they learn long-term dependencies and this produces a learning problem where it is difficult to store information for a long time (LeCun et al. [2015]).

While learning the RNN the gradients keep changing at each time step, they may grow or shrink which cause an exploding gradients problem or vanishing gradients problem.

Exploding gradients problem: It is a problem that happens when training the RNN with the backpropagation. The shared weights are the same, so the gradient is multiplied with the same quantity resulting an exploding in the gradient when the weight w is bigger then one (Aggarwal [2018a]).

Vanishing gradients problem: Is a problem that happens when the shared weight w is less then one which makes the gradient keep shrinking throw time (Aggarwal [2018a]).

Solutions: There is different solutions (Aggarwal [2018a], Grosse [2017]):

- We can change the used activation function because its derivation is included in the product while calculating the gradient, an activation function such as ReLU may be more helpful then tanh and segmoid.
- For the exploding gradient problem we can use the Gradient Clipping which prevent gradients from exploding by rescaling them. It means capping the maximum value for the gradient.
- We can use also a technique called Input Reversal which means reversing the input words order when the gaps between the input and the output are very long.
- Another solution proposed in Hochreiter and Schmidhuber [1997], Graves et al. [2008], where the structure of RNN have been improved with an architecture called short-term long-term memory, denoted by LSTM.

Backpropagation Throw Time

Backpropagation Throw Time (BPTT) is an updated version of Back Propagation for RNNs which update weights throw time. BPTT unfold the RNN in time to create an equivalent feedforward every time a sequence is processed which make the derivatives calculable via standard BP (Du and Swamy [2013]). BPTT performs gradient descent on a complete unfolded network.

Suppose that the training starts at time t_0 and ends at time t_1, the total cost function is the sum over time of the standard error function $E_{sse/ce}(t)$ at each time-step (Bullinaria [2013]), as follows in 1.18 :

$$E_{total}(t_0, t_1) = \sum_{t=t0}^{t1} E_{sse/ce}(t) \tag{1.18}$$

and the gradient descent weight updates each time-step 1.19 :

$$\Delta W_{ij} = -\eta \frac{\partial E_{total}(t_0, t_1)}{\partial W_{ij}} = -\eta \sum_{t=t0}^{t1} \frac{\partial E_{sse/ce}(t)}{\partial W_{ij}} \tag{1.19}$$

The partial derivatives $\partial E_{sse/ce}/\partial W_{ij}$ have contributions from each weight $W_{ij} \in \{W_{IH}, W_{HH}\}$ and depend on the inputs and hidden unit activation at previous time steps, after that the error back-propagate throw time and network (Bullinaria [2013]).

Long Short Term Memory

Long Short Term Memory (LSTM) is an update of RNN introduced by Hochreiter and Jürgen Schmidhuber and Bengio et al. in Hochreiter and Schmidhuber [1997], Graves et al. [2008] in order to solve the exploding/vanishing gradient problems, where the basic idea was to modify RNN to allow a better flow to error derivations.

LSTM was designed to transmit important information many time steps into the future in order to learn and remember long term dependencies. The key component of LSTM is the memory cell which hold the information that it has learned over time until it's needed. The activations correspond to short-term memory while the weights correspond to long-term memory (Buduma and Locascio [2017], Grosse [2017]).

LSTM maintains the state of the cell as well as carrying weights to ensure that the signal and information are not lost during sequence processing. Also, it can learn how to make a multi-step prediction from one-step and it may be useful for predicting a time series.

In RNN there is a single tanh layer, while in LSTM there is four layers as in Figure 1.14

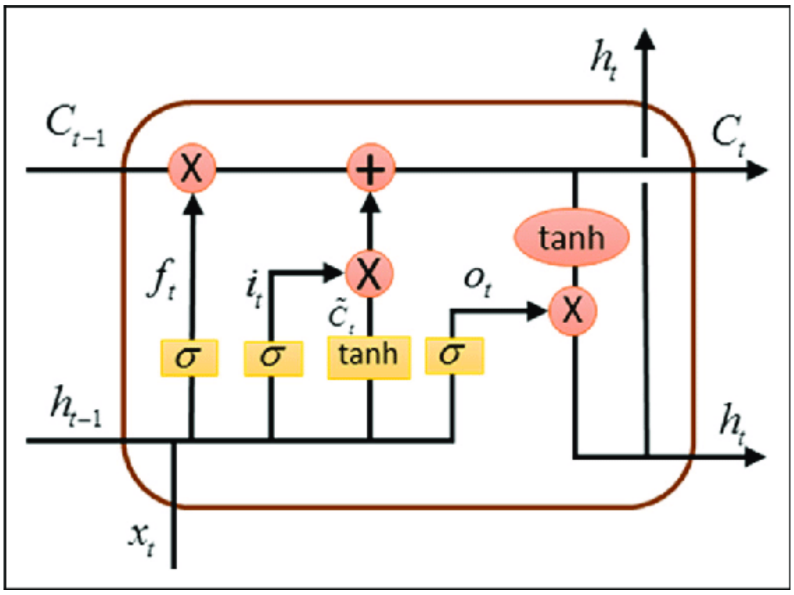


Figure 1.14: LSTM structure (Tao and Zhou [2017]).

As shown in Figure 1.14 LSTM have 3 main gates: input gate, forget gate and output gate.

Forget gate f_t : The gate's job is to discover if the past memory is useful or not. The past memory is rich with information that we have to figure out if it is useful or not. This gate multiplies the past memory cell with a vector that takes the value of one or zero (memory kept or forget). This vector results from applying a segmoid function on the result of the multiplication of the weight by the past hidden layer and the current input added to them the bias .

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f) \tag{1.20}$$

Input gate i_t : Also use the input and the past hidden state to preserve the worthless input and use it to gate the new memory.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i) \tag{1.21}$$

New memory generation $\tilde{C}_t$: Use the input and the past hidden state to generate a new memory.

$$\tilde{C}_t = \tanh(W_c.[h_{t-1}, x_t] + b_c) \tag{1.22}$$

Final memory generation C_t : It takes the advice of the forget gate f_t and input gate i_t to produce the final memory.

It adds the forget gate result to the result of the multiplication of the input gate with the generated memory to produce the final memory.

$$C_t = f_t.C_{t-1} + i_t.\tilde{C}_t \tag{1.23}$$

Output gate o_t : It separates the final memory from the hidden state and makes the assessment regarding what parts of the memory needs to be present in the hidden state.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) \tag{1.24}$$

$$h_t = o_t.\tanh(C_t) \tag{1.25}$$

Gated Recurrent Units

Gated Recurrent Units (GRU) is an updated version of LSTM, where the forget and input gates are replaced with an update gate z_t, and a reset gate r_t is introduced to modify the h_{t-1}, and the internal memory C_t is eliminated (Aggarwal [2018a]), as shown in Figure 1.15.

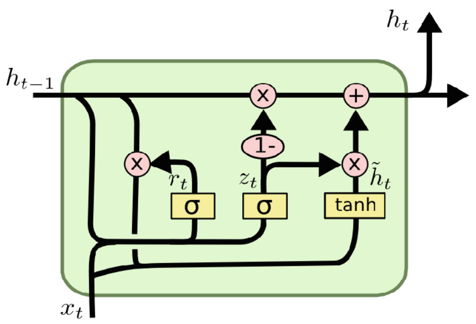


Figure 1.15: GRU structure (Jabreel and Moreno [2019]).

The main difference with the LSTM is that a single gating unit simultaneously controls the forgetting factor and the decision to update the state unit. The updated equations are the following:

The update gate z_t:

$$z_t = \sigma(W_z.[h_{t-1}, x_t]) \tag{1.26}$$

The reset gate r_t:

$$r_t = \sigma(W_r.[h_{t-1}, x_t]) \tag{1.27}$$

The new memory $\tilde{h}_t$:

$$\tilde{h}_t = \tanh(W.[r_t * h_{t-1}, x_t]) \tag{1.28}$$

The hidden state h_t:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h} \tag{1.29}$$

Bidirectional RNN

As mentioned earlier, RNN have knowledge of previous entries, but this knowledge is up to a certain point, and so does not have information about future cases.

In the bidirectional recurrent network, we have separate hidden states $h^{(t)}$ and $g^{(t)}$ for the forward and backward directions as shown in Figure 1.16. The forward states interact in the forwards direction, while the backwards states interact in the backwards direction. Both $h^{(t)}$ and $g^{(t)}$ receive input from the same vector $x^{(t)}$ and they interact with the same output vector $o^{(t)}$ (Aggarwal [2018a]).

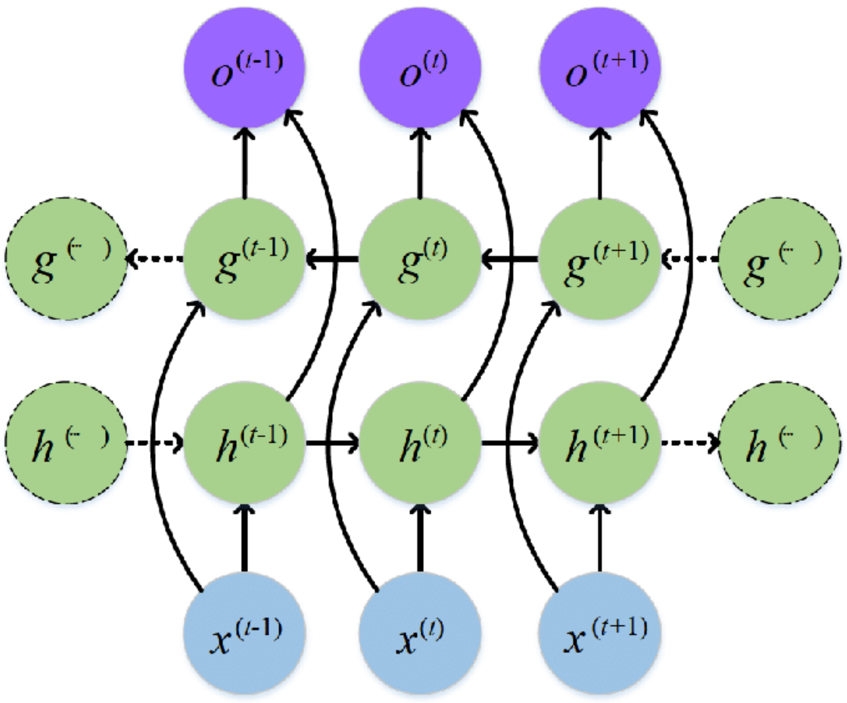


Figure 1.16: Bidirectional RNN (Feng et al. [2017]).

Multi-layer RNN

The multi-layer structure is used in large and complex models and can be used with RNN as it is considered highly effective, especially with the LSTM or GRU structure. The nodes in higher-level layers receive input from those in lower-level layers. The relationships among the hidden states can be generalized directly from the single-layer network (Aggarwal [2018a]).

Example on multi-layer RNN is described by Figure 1.17:

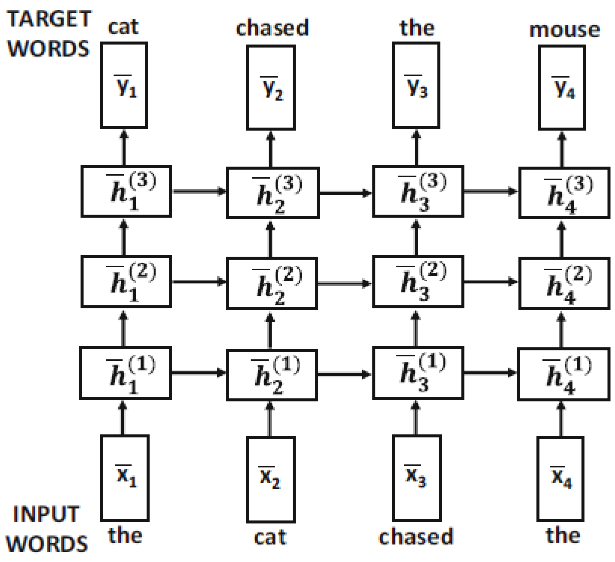


Figure 1.17: multi-layer RNN (Chaudhuri [2019]).

1.6 Generalization, Regularization and Optimization techniques

This section overviews some generalization techniques, strategies of regularization, and optimization techniques for deep NN.

1.6.1 Generalization

The goal of building and training machine learning and deep learning models is to give them the ability to generate results when providing them with data they have not trained on which make them general models.

Generalization is the ability to perform well on unseen inputs where their value is estimated or in other words, the input data is interpolated and extrapolated (Du and Swamy [2013], Goodfellow et al. [2016]).

To represent our problems well, the training set should be sufficiently large and for a good generalization the size of the training set should be several times larger than the network's capacity (Du and Swamy [2013]). To know the generalization degree of our model we compute the generalization error. The generalization error is divided into two parts, which are an approximation error, which is due to the presence of a limited number of used parameters and the ignorance of the noise level in the training data, and the training error (estimation error), which is due to the presence of a limited number of data. When using a larger model, our approximation error decreases, but the estimation error increases, and therefore we cannot reduce the error components at one time, but we try to determine an optimal size for the trade-off between approximation and estimation errors. We can also view the generalization error as the sum of bias squared plus the variance (Du and Swamy [2013]).

The bias and the variance are useful to characterize the concept of generalization (Goodfellow et al. [2016], Aggarwal [2018a]). The bias is caused by the simplifying assumptions and the inappropriate choice of the size of a class of model when the training set is infinite, which causes errors. Whereas the variance is caused by finite number of training samples which makes the model not able to learn all the parameters of the model. The limited data make the model have a large number of parameters.

To get a good generalization we need to balance the bias and the variance values.

Generalization by early stopping

Early stopping is one of the best methods used to improve generalization as we stop training before reaches the absolute minimum. We should choose an optimum stopping point to avoid over-training and high-frequency noise learning (Du and Swamy [2013]).

Generalization by regularization

Another method to improve generalization is the regularization. Its concept is summarized in choosing a good regularizer to decrease the variance by affecting the bias as little as possible (Du and Swamy [2013]).

Over-fitting and Under-fitting

The over-fitting and under-fitting problems are among the most common generalization problems that can be defined as follows (Du and Swamy [2013], Goodfellow et al. [2016]). When training the model too much it gives us perfect results for training data and the training error decreases until it is almost nonexistent, but on the other hand the generalization error increases too much and the model became not generalizable this is over-fitting. When the model is not adequately trained, both training error and generalization error increases, this is called under-fitting. Over-fitting gives a good training performance and a poor generalization while under-fitting gives a poor training and generalization performances.

When the bias is high it cause an under-fitting and when the variance is high it cause an over-fitting.

The under-fitting problem does not occur much and can be solved by acquiring more data for the algorithm to train on or increasing the model's complexity by adding more layers or weights or parameters or also increasing the number of times a model is trained, while the over-fitting problem is solved by regularization.

1.6.2 Regularization

Regularization is defined by: "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error." (Goodfellow et al. [2016]).

The regularization technique enables us to convert incorrectly presented problems into well-posed problems in order to make the solution stable. Generally the regularization is performed by adding a penalty called regularizer to the cost function. There is no best machine learning algorithm and no best form of regularization, we have to choose the best suitable regularization form to the task that we want to solve (Goodfellow et al. [2016]).

L1 and L2 Regularization

Here we present two forms of regularization which can be used to prevent over-fitting (Buduma and Locascio [2017], Subramanian [2018]):

L1 regularization

L1 regularization also refereed to as Lasso Regression is the most used type of regularization in ML/DL which adds absolute value of magnitude of coefficient as penalty term to the loss function. In NNs a coefficient is added to all weights in the network or with another word a sum of absolute values of weight coefficients are added to loss function. L1 has a property that leads the weight vector to became very close to zero during optimization. It makes the neurons resistant to noise in the input by making them use a small subset of their most important inputs.

L2 regularization

L2 regularization also refereed to as weight decay or Ridge Regression is another common type of regularization. It augments the error function by adding squared magnitude coefficient which is the weight in NNs to the loss function or with another word a sum of absolute values of weight coefficients are added to loss function. It encourage the network or the model to use all the inputs a little, using L2 during the gradient descent update means that all the weights decayed linearly to zero.

L1 regularization is useful in understanding which features are contributing to a decision, if feature analysis is not necessary it is preferable to use L2 regularization because it presents a better empirically performance.

Dropout

Dropout is one of the most used and powerful regularization technique developed by Hinton and his students at the University of Toronto in (Hinton et al. [2012]). It is widespread used in DL and applied to intermediate layers of the model during the training time.

It can be considered as a different kind of method to prevent over-fitting in deep NNs. A probability is given to neurons where they remain active only if the probability value is different from zero which make the network forced to accurate even when some information are missing. With this way the network is prevented from relying heavily on any neuron.

This naïve implementation of dropout requires scaling of neuron outputs at test time which is undesirable so we use the inverted dropout where the scaling is done at training time. The output of the neuron that his activation is not silence is divided by the probability p (a hyperparameter) before propagating his value to the next layer (Buduma and Locascio [2017]).

To sum up, regularization is one of the central techniques in ML/DL, rivaled in its importance by the optimization (Goodfellow et al. [2016]).

1.6.3 Optimization

Optimization is known as the task of minimizing or maximizing a function usually called objective function. Usually most optimization problems are minimization problems and the minimized function is called cost or loss function. There are two kinds of optimization algorithms (Goodfellow et al. [2016]): First order and second order.

First-order Optimization Algorithms

Is the algorithms that use only the gradient such as:

Gradient Descent

Gradient Descent (GD) is a method that calculate the gradient and minimize the loss function $f(x)$ by moving x in small steps according to the result of the derivation provided by the gradient vector which tell us which direction to move. The gradient descent is used in NNs learning. The gradient descent have many versions, we mention of them (Buduma and Locascio [2017]):

Stochastic Gradient descent (SGD): We estimate the error at each iteration with respect to a single example which helps us to navigate flat regions.

Batch Gradient Descent (BGD): The hole data set is used to compute the error and the gradient take the path of steepest descent.

Mini-Batch Gradient Descent: We compute the error at each iteration with respect to a subset of the data which is refer to as Mini-batch.

Momentum: The momentum method is used to accelerate learning when we face a high curvature, small but consistent gradients. It move in the direction of past gradient after accumulating an exponentially decaying moving average of them. Some times SGD is slow in learning so it is added with the momentum algorithm.

Second-Order Optimization Algorithms

Sometimes, we are interested in the second derivative which tells us how the first derivative changes when changing the inputs, this is important because it tells us whether the gradient step will cause a lot of improvement as we expect based on the gradient alone. This led to the emergence of second-order optimization algorithms that not only uses the gradient but also the Hessian Matrix that collects all second order derivatives.

The most widely used second-order method is Newton's method (LeCun et al. [1998], Goodfellow et al. [2016]).

1.7 Conclusion

This chapter prove the choose of deep learning to solve the problem of the next word prediction. To be more confident, we will see how this problem can be solved without using DL and with it. The next chapter present the different methods used in next word prediction.

Chapter 2

Next Word Prediction

2.1 Introduction

Nowadays people rely heavily on new technologies in their daily lives, such as smartphones and social media, which makes them search for aids that reduce effort in writing. Smart keyboards provide suggestions to the user by predicting the next word, which is one of the major problems currently being addressed and which we will present in this chapter.

Word prediction dates back to early last century before the advent of computers with Andrei Markov, who developed the mathematical basis to predict upcoming symbols from symbol chains of fixed length (Markov [1913]).

Next, Claude Shannon devised a game where a player guesses a random text snippet letter by letter (Shannon [1951]). Then a family of models based on Markov's work emerged which assume that the appearance of a symbol depends on a number of other symbols, n-gram language model is an example.

In 1956, Noam Chomsky showed in his seminal paper Chomsky [1956] that a natural language expression can be described by other means than by a Markov model. He gave a proposition of a phrase structure grammar and creates a hierarchy of language types. Those grammatical, syntactic and semantic information have been incorporated and used later in word prediction.

The two labs IBM Thomas J. Watson Research Center and CMU used n-grams in their speech recognition systems (Baker 1975b, Jelinek 1976, Baker 1975a, Bahl et al. 1983, Jelinek 1990).

Many prediction systems based on different prediction methods have been in use since the early 1980s.

In this chapter we will present the various methods mentioned previously to solve the problem of next word prediction as well as the new ones based on machine learning and deep learning.

2.2 Natural Language Processing

Natural Language Processing (NLP) is a very active research field since its appearance four decades ago. The origins back to Alan Turing test where the exigency to imitate the human lead to thinking about processes which can manipulate the natural language (Gendron and Gendron [2015]), where it collected a set of various models, tools, statistical and linguistic techniques to solve problems. The main goal of NLP is to study how humans understand and use languages and develop tools that enable computers to simulate humans (Doszkocs [1986], Chowdhury [2003]). "It encompasses all computer-based approaches to the handling of unrestricted written or spoken language, from purely “mechanistic” procedures, as employed in many text editors, word processors, and in automatic-indexing approaches in information retrieval (IR), to “intelligent” analysis, understanding and expression of “meaning” as exemplified in natural-language understanding, question answering, speech recognition, machine translation and expert systems in artificial intelligence (AI)." (Doszkocs [1986])

NLP can be divided into two sub-fields: Natural Language Understanding (NLU) and Natural Language Generation (NLG), as shown in Figure 2.1. NLU deals with an understanding of the structure of language, whether it is words, phrases or speeches. It focuses on syntax and semantics and tries to solve different types of ambiguities related to them. Where, NLG try to teach machines how to create Natural Language in a reasonable way (Thanaki [2017]).

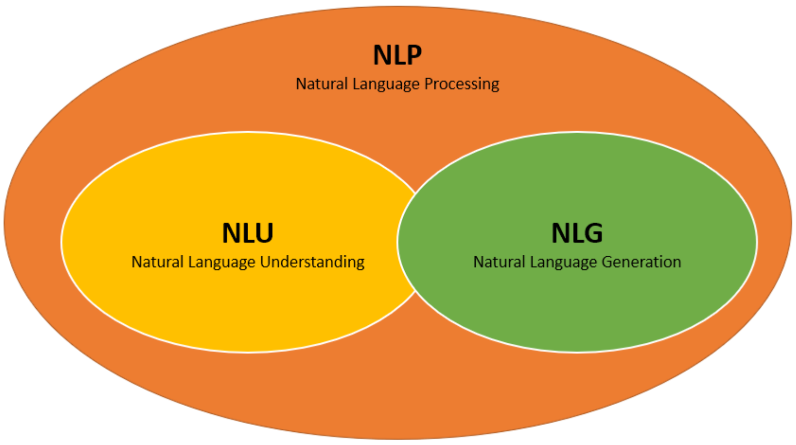

NLP
Natural Language Processing
NLU
Natural Language Understanding
NLG
Natural Language Generation

Figure 2.1: The relation between NLP, NLU and NLG.

The problem of next word prediction that we cover in this work fall under NLG problems.

2.3 Language Modeling

The idea that some word sequences are more frequent than others has led to the concept of language modeling which is now considered a central task in NLP (Jozefowicz et al. [2016], Pereira and Paraboni [2007]).

A language model can be considered as a probability distribution $P(W)$ over W which is a sequences of symbols called strings, $P(W)$ tel us how often W occurs in a given text or sentence in a specific language. Language Modeling is widely used in many tasks such as: Machine Translation, Speech Recognition, Spelling Correction and in predicting the next word which is it’s main goal. The probability of a word in a sequence of words is based on the previous words in the sequence (Chen and Goodman [1999], Zitouni [2014], Kłosowski [2018]):

$$P(W) = \prod_{i=1}^{N} P(w_i|w_1, ..., w_{i-1}) \tag{2.1}$$

Where $P(w_i|w_1, ..., w_{i-1})$ is a conditional probability that w_i will occur given the previous word sequence $w_1, ..., w_{i-1}$.

A basic and widely used language model is N-gram language model which gives the probability of a word w_i based on N previous words.

$$P(w_i|w_{i-N+1}, ..., w_{i-1}) \tag{2.2}$$

The first and simplest way to approximate this probability is called Maximum likelihood estimation (MLE) (Wandmacher [2009]).

2.3.1 Maximum likelihood Estimation

MLE calculate the probability distribution based on frequencies where it is formalized as follow:

$$P_{MLE}(w_i|w_1, ..., w_{i-1}) = \frac{C(w_1, ..., w_i)}{C(w_1, ..., w_{i-1})} \tag{2.3}$$

Where $C(w_1, ..., w_i)$ is the frequency of the sequence $w_1, ..., w_i$ in the corpus.

The maximum likelihood estimation can lead to poor performance when we estimate the probability based on a long history. It sets a zero probability for a word that has not occurred before in the training set. The phenomenon of not seen enough data in training data is refer to as data sparsity (or data sparseness) problem, and this is what we do not want it to happen because we will work on different data from the training set so a technique called Smoothing or Discounting is used to address this problem (Allison et al. [2006], Wandmacher [2009]).

2.3.2 Smoothing

Smoothing or discounting refers to those techniques that adjust MLE to give more appropriate probabilities. It makes the probability distributions more uniform by upwarding low probabilities and down-warding high probabilities, they also improve the accuracy of the model (Chen and Goodman [1999]). Here we present some of the famous and widely used smoothing techniques.

Additive Discounting

Also known as additive smoothing is the oldest and simplest used solution. We add counts by 1 according to Laplace work in 19th century to avoid zero (Chen and Goodman [1999], Wandmacher [2009]), as seen:

$$P_{Laplace}(w_i|w_1, ..., w_{i-1}) = \frac{C(w_1, ..., w_i) + 1}{C(w_1, ..., w_{i-1}) + |V|} \tag{2.4}$$

Where $|V|$ is the size of the vocabulary.

This technique does not give good result because it can change the probability distribution strongly (Chen and Goodman [1999], Wandmacher [2009]). Lidstone (1920) generalized Laplace's law by adding a factor δ to every count which improves the probability estimates significantly where $0 < \delta \leq 1$:

$$P_{Lidstone}(w_i|w_1, ..., w_{i-1}) = \frac{C(w_1, ..., w_i) + \delta}{C(w_1, ..., w_{i-1}) + \delta|V|} \tag{2.5}$$

Good–Turing Estimate

Good–Turing Estimate (GT) estimate is central to many different smoothing techniques. GT estimate assume that the probability of not yet seen events equal to the number of events seen once n_1 divided on the total number of existing events N: $\frac{n_1}{N}$ Pereira and Paraboni [2007]. In GT estimate we assume that for any n-gram that occur r times, it occur r^* times where (Chen and Goodman [1999]):

$$r^* = (r + 1).\frac{n_{r+1}}{n_r} \tag{2.6}$$

Where n_r is the number of n-grams that occur r times.

With this technique the resulted probability contains non-zero estimates to unseen n-grams (Pereira and Paraboni [2007]). GT is used as a tool in various smoothing techniques and is not in itself used for smoothing because it does not include the combination of higher-order models with lower-order models which are considered necessary as described later (Chen and Goodman [1999]).

Kneser-Ney Discounting (KN)

Kneser-Ney discounting (Kneser & Ney, 1995) is an extension of absolute discounting. Absolute discounting is based on subtracting a small fixed amount D from all non-zero counts where in KN discounting the combination of lower-order distribution with a higher-order distribution is built in a new way where in previous the lower-order distribution was taken to be as a smoothed version of the lower-order maximum likelihood distribution and in KN the calculation on the lower-order n-gram probabilities is optimized in case the higher order n-gram was unseen where the lower-order distribution is selected in a way that the marginals of the higher-order smoothed distribution match the marginals of the training data (Wandmacher [2009], Chen and Goodman [1999], Körner and Staab [2013]).

In case of uni-gram which is the lowest level there is no interpolation, so:

$$P_{KN}(w_i) = \frac{N_1 + (\bullet w_i)}{N_1 + (\bullet\bullet)} \tag{2.7}$$

Where:

$$N_{1+}(\bullet w_i) = |\{w_{i-1} : C(w_{i-1}, ..., w_i) > 0\}| \tag{2.8}$$

$$N_{1+}(\bullet\bullet) = |\{(w_{i-1}, w_i) = C(w_{i-1}, ..., w_i) > 0\}| = \sum_{w_i} N_{1+}(\bullet w_i) \tag{2.9}$$

$N_{1+}(\bullet w_i)$ refer to the number of words that precede w_i at least once in the data set.

In case of higher levels:

$$\begin{aligned} P_{KN}(w_i|w_{i-n+1}, ...w_{i-1}) &= \frac{\max\{N_{1+}(\bullet w_{i-n+1}, ...w_i) - D, 0\}}{N_{1+}(\bullet w_{i-n+1}, ...w_{i-1}\bullet)} \\ + \frac{D}{N_{1+}(\bullet w_{i-n+1}, ...w_{i-1}\bullet)}&.N_{1+}(w_{i-n+1}, ..., w_{i-1}\bullet).P_{KN}(w_i|w_{i-n+2}, ...w_{i-1}) \end{aligned} \tag{2.10}$$

Where:

$$N_{1+}(\bullet w_{i-n+1},....w_i) = |\{w_{i-n} : C(w_{i-n},....,w_i) > 0\}| \tag{2.11}$$

$$N_{1+}(\bullet w_{i-n+1},....w_{i-1}\bullet) = |\{(w_{i-n},w_i) : C(w_{i-n},....,w_i) > 0\}| = \sum_{w_i} N_{1+}(\bullet w_{i-n+1},....w_i) \tag{2.12}$$

As in absolute discounting the D is a small fixed account where $0 \leq D \leq 1$ and $\frac{D}{N_{1+}(\bullet w_{i-n+1},....w_{i-1}\bullet)}.N_{1+}(w_{i-n+1},....,w_{i-1}\bullet)$ is the weight that determines the impact of the lower order value on the result.

After compensation we get:

$$P_{KN}(w_i|w_{i-n+1},....w_{i-1}) = \frac{\max\{C(w_{i-n+1},....,w_i) - D, 0\}}{\sum_{w_i} C(w_{i-n+1},....,w_i)} + \frac{D}{\sum_{w_i} C(w_{i-n+1},....,w_i)}.N_{1+}(w_{i-n+1},....,w_i\bullet).P_{KN}(w_i|w_{i-n+2},....w_{i-1}) \tag{2.13}$$

Modified Kneser-Ney Discounting

Chen and Goodman in Chen and Goodman [1999] introduced a variation of KN smoothing by converting it into an interpolation smoothing method where:

$$P_{MKN}(w_i|w_{i-n+1},....w_{i-1}) = \frac{C(w_{i-n+1},....,w_i) - D(C(w_{i-n+1},....,w_i))}{\sum_{w_i} C(w_{i-n+1},....,w_i)} + \gamma(w_{i-n+1},....,w_i).P_{MKN}(w_i|w_{i-n+2},....w_{i-1}) \tag{2.14}$$

Where $\gamma(w_{i-n+1},....,w_i)$ is the scaling or interpolation factor to mix in lower-order distribution and it value depends on D.

The single discount D is replaced with three different parameters, D_1, D_2, and D_{3+}, where:

$$D(C) = \begin{cases} 0 & \text{if } C = 0 \\ D_1 & \text{if } C = 1 \\ D_2 & \text{if } C = 2 \\ D_{3+} & \text{if } C \geq 1 \end{cases} \tag{2.15}$$

The modified KN discounting performs better than the KN discounting and it is considered as the best way to solve zero probability problem (Chen and Goodman [1999], Wandmacher [2009]).

2.3.3 Model combination

Backing-off

Backing off presented in Katz [1987], also called katz smoothing, is a combining model which combine information from several models. The idea is to consult the model of order $n-1$ if the model of order n cannot help and so on or in other words combining higher-order models with lower-order models (Wandmacher [2009], Chen and Goodman [1999]):

$$P_{BO}(w_i|w_{i-n+1},.....,w_{i-1}) = \begin{cases} \frac{C^*(w_{i-n+1},....,w_i)}{C(w_{i-n+1},....,w_{i-1})}, & \text{if } C(w_{i-n+1},....,w_i) > 0 \\ \alpha_{w_{i-n+1},....,w_{i-1}} P_{BO}(w_i|w_{i-n+2},.....,w_{i-1}), & \text{otherwise} \end{cases} \tag{2.16}$$

The algorithm recursively applies models of a shorter history until $n = 1$. C^* represents an already smoothed count which is calculated according to GT estimate, because such a backoff scheme is normally applied in combination with a standard smoothing scheme and the coefficient α is a normalizing factor which is also called backoff weight which assures that the sum of all probabilities remains unchanged.

Interpolation

Interpolation is another combining model that combines information from several models. Unlike backing-off, interpolation mixes the estimates from all models. In linear or simple interpolation we multiply each model by a coefficient λ_i which represent the weight then we add to it the single estimate (Wandmacher [2009]):

$$P_{LI}(w_i|w_{i-2}w_{i-1}) = \lambda_1.P^*(w_i|w_{i-2}w_{i-1}) + \lambda_2.P^*(w_i|w_{i-1}) + \lambda_3.P^*(w_i) \tag{2.17}$$

Where $0 \leq \lambda_i \leq 1$ and $\sum_i \lambda_i = 1$. p^* is meant to be any kind of estimate. We mention also that there is other types of interpolation like geometric interpolation.

2.3.4 Expectation Maximization algorithm

One of the most important tasks is to find the optimal parameters for a particular algorithm. Often this is done by experimenting several times until finding the optimal value, but experimentation takes time and does not guarantee reaching a result. This is why many algorithms that perform this task have been developed. In probabilistic, the most notable one is Expectation Maximization (EM) algorithm.

EM is a greedy algorithm proposed by Dempster et al. in 1977. The algorithm always converges but it also may stuck within a local minimum, moreover it is fast and very easy to apply. The algorithm has two main steps: expectation step which initialize the parameter or expect his value and maximization step which maximize the parameter value expected on expectation step. It is used to find optimal weights for combined language models and had different versions (Wandmacher [2009]).

2.4 Word Prediction Systems

Prediction in NLP means guessing the missing letter, word or sentence (Ghayoomi and Momtazi [2009]). Currently there are many prediction programs that aim to reduce the number of keystrokes needed for writing by trying to predict the letter, word or phrase that the user is writing or intending to write later (Väyrynen et al. [2007]).

The prediction program monitors the words and letters that the user is typing and applies one of the prediction methods or many of them to produce a list of predictions that are updated with each new letter the user writes then the user selects the appropriate prediction from this list.

We are going to concentrate on word prediction systems where only the next word is offered. The biggest challenge is that the left context is the only piece of information available for predicting and often not sufficient (Väyrynen et al. [2007]).

Word prediction systems use both word prediction and word completion, which first predicts the next possible word and gives a list of suggestions, then when the user does not choose one of the suggestions presented he continues writing letters here comes the word completion role where the prediction system gives the following possible words based on the written letters. This process is illustrated in Figure 2.2

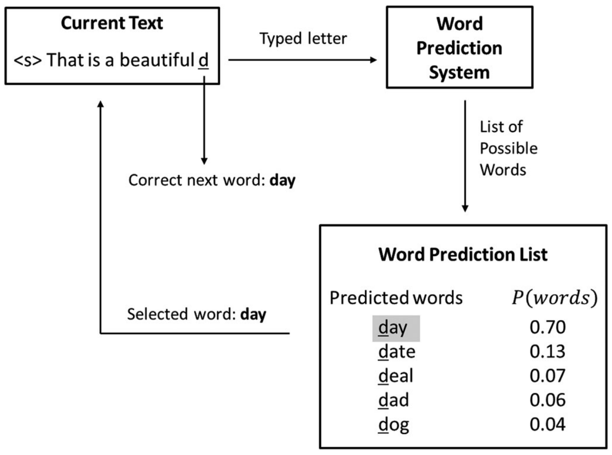


Figure 2.2: Example of the prediction system to English with five words in the prediction list (Cavalieri et al. [2016]).

2.4.1 Evaluation Metrics

The quality of prediction systems and methods can be evaluated using several measures, the most famous are discussed:

Savings

It means the potential savings obtainable by using a predictor (Väyrynen et al. [2007]). There is three involved parameters which can be defined as follow:

Savings is the cost difference between using predictions and entering standard events.

Where the cost of using predictions includes the visual scan time for searching for predictions and cognitive time for deciding on the correct prediction with physical movement time required to select a prediction.

Whereas the cost of entering standard events comprise the cognitive time required to formulate the next event and the physical movement required to enter the next event.

Hit Rate

Hit Rate (HR) is the percentage of times that the intended word appears in the prediction list where a high hit rate means a good prediction performance (Wandmacher [2009]).

Hit Ratio

Hit ratio expresses the possibility of guessing the word before writing it and is calculated by dividing the number of times that words are guessed by the number of written words (Garay-Vitoria and Abascal [2006]).

$$Hit\ Ratio = \frac{number\ of\ times\ that\ words\ are\ guessed}{number\ of\ written\ words} \tag{2.18}$$

Keystrokes Until Completion

Keystrokes Until Completion (KUC) is the average number of keystrokes that have to be entered before the intended word appears in the prediction list where a high performance means a low value of KUC (Wandmacher [2009], Aliprandi et al. [2008]).

$$KUC = \frac{(c_1 + c_2 + .. + c_n)}{n} \tag{2.19}$$

Where $c_1...c_n$ is the number of keystrokes for each of the n words before the desired word appears.

Accuracy

Accuracy (ACC) is the percentage of words that could be predicted and completed before the user reached the end of the word (Wandmacher [2009]).

Keystroke Saving Rate

Keystroke Saving Rate (KSR) or Keystroke Saving (KS or KSS) is the percentage of saved keystrokes reached by using the predictor (Wandmacher [2009]). It does not tell us how to reduce the keystrokes but tell us how much we can save by using the predictor.

$$KSR = (1 - \frac{K_{red}}{K_{all}}) \times 100 = (\frac{K_{all} - K_{red}}{K_{all}}) \times 100 \tag{2.20}$$

Where K_{red} is the number of keystrokes needed on the input device when typing with reduction or prediction method and K_{all} is the number of keystrokes needed on the input device when typing without reduction or prediction method.

Perplexity

Perplexity (PP) is the average number of possible choices or words that may occur after a string of words and it can be measured with cross entropy calculated on test set with N words (Cavalieri et al. [2016]), where the cross entropy equation is:

$$H(W) = \frac{1}{N}\sum_{i=1}^{N} \log(\frac{1}{P(w_i|w_{i-(n-1)}, t_{i-(m-1)})}) \tag{2.21}$$

where $W = (w_1, w_2, ..., w_N)$ represents the words in the test set and the PP is:

$$PP(W) = 2^{H(W)} \tag{2.22}$$

2.4.2 Applications

Word Prediction is useful in many domains and used in many applications (Ghayoomi and Momtazi [2009], Aliprandi et al. [2008], Väyrynen et al. [2007]).

- Text production proper where we generate texts by predicting the next words.
- Writing assistance systems and assistive communication system such as Augmentative and Alternative Communication (AAC) devices, where those systems predict the next word or character that the users wants to write to help them and reduce the effort needed to write.

- Speech recognition, where in the case of different pronunciation of words from one person to another, we can predict those words based on what the user previously said, and we can improve the results of speech recognition by correcting the resulting words by predicting them.
- Spelling correction and error detection, where we predict correct words based on typed characters and words.
- Word-sense disambiguation, where we can know the exact meaning of it based on its predecessors or predicting a synonym for that word, which makes the meaning more clear.
- Statistical machine translation, where when translating from a language to another we may make mistakes due to the difference between languages so we use word prediction to minimize and correct those errors.
- Handwriting recognition and optical character recognition where many wrong words can be obtained due to the difference in the writing method from person to another, here we can use word prediction to reduce these errors and letter prediction to make Optical Character Recognition more accurate.
- Also it can be used in text-entry interfaces for messaging on mobile phones and typing on handheld and ubiquitous devices or in combination with assistive devices such as keyboards, virtual keyboards, touchpads and pointing devices.

2.5 Factors affecting prediction

There are some factors that influence the prediction process such as text block size, dictionary structure and prediction method. We give a survey here on those techniques (Garay-Vitoria and Abascal [2006]).

1. Predictable Block Size: The choice of the size of predicted block is an important factor due to its contribution in the value of savings and hit ratio, where it can take the value of 1 character (at this case the model save no keystrokes), n characters (n-grams), word or more than one where the most block used is the word.
2. Dictionary Structure: Word prediction systems store the required information using dictionaries (lexicons), it can use more than one; the organization of them makes difference in the system, where there's two type of organization, sequential: simple but slow, or structured: faster but more complex. Note that the adaptation of dictionaries can augment the keystroke savings in the case where the user's vocabularies are added.
3. Prediction Methods: There are many approaches, we present five of them in what follows:

    (a) Frequencies: The system suggests the most probably words that have the same prefix with the word written by the user. It is possible to store the recency also in order to improve the results but also increase the computational complexity and the storage.

(b) Word probabilities: A two-dimensional array stores the word and the conditional probability of the next word likely. This method can be improved if recency is taken on consideration, but it is usually restricted because of the difficulty of updating the table with fixed dimension with the user vocabulary.

(c) Syntactic prediction with probabilities: For this approach, two information are used: the probability of appearance for each word, and the probability of appearance for each semantic category after the appearance of another. This approach achieves usually better results than the precedent discussed.

(d) Syntactic prediction using grammars: Grammatical rules are taken in to consideration which constructed by analyzing sentences using grammars and NLP techniques to obtain the category with the highest probability of appearance. The same dictionary of the precedent approach is used with some addition of morphological information. It is complicated in computation than the last approach but makes appropriate predictions.

(e) Semantic prediction: This approach is similar to the precedent, where words are associated with a semantic category, which is difficult and augment the computational complexity, as results are similar to those of the syntactic approaches.

4. User Interface Characteristics: The characteristics of the user interface are also one of the most important factors that affect the quality of the prediction program, since if they are not well prepared the user may not benefit from them. The number of suggestions must be taken into consideration, many suggestions may distract the user and take time from him to read them up to the time required to write the desired word while submitting one or two suggestions represents a risk as there is a high possibility that no word will be chosen. The manner in which choices are presented is also important as showing proposals as vertical word lists gives better acceptance because the effort required to see and manipulate them is less.

5. Proposals Arrangement: The arrangement of proposals is also important, as they are often arranged according to the likelihood of their occurrence, they can also be arranged alphabetically to facilitate their reading, or provide suggestions in a specific location for the user to save their position and access them easily.

6. It is also needed to balance algorithm complexity, dictionary size and response time to get more efficient results. The user's physical and mental ability should also be taken into account as it greatly affects the predictor's effectiveness. When building a prediction system, smart decisions must be taken, taking into account all the factors mentioned above and trying to balance them.

2.6 Classical Prediction Methods

Classical techniques are those that use probabilities in the estimation of the occurrence of a word. Other techniques enhance these technique by syntactic or semantic construction (Ghayoomi and Momtazi [2009], Makkar et al. [2015]).

2.6.1 Statistical Modeling

Statistical modeling, also known as probabilistic modeling, use the probability of the word appearing in the text to anticipate letters and words and even phrases and place them in the list of predictions. In case of words, the prediction is made with n-gram model which is based on Markov assumption where the precedent $n-1$ word affect the next word. In what follows, we describe the most common statistical methods for prediction:

Word Frequency

The first method in word prediction use the word frequency to complete what the user is writing without taking into consideration what was previously written. The systems based on this method used the unigram model with a fixed lexicon. This method makes the system gives the same prediction suggestions for a specific letter sequences and often gives inappropriate suggestions (Ghayoomi and Momtazi [2009], Makkar et al. [2015]).

Swiffin et al. [1987] used unigram model but with adaptive lexicon in PAL[1]. PAL generates prediction based on word frequency and recency of use information stored in its lexicon. The predictions are generated and updated according to the typed prefix. The lexicon adapts to the user by adding the user's vocabulary to the dictionary and update the frequency and recency to give more appropriate predictions. PAL reduces to over 50% the number of characters necessary to enter a text.

N-Gram Language Model

This method has been developed to overcome the limitation of the previous method. It takes into account the previous context where the previous words are used to predict the next word. When using only the previous word, the model is called bigram, and when using the previous two words it is called trigram and so on (in general when using the previous $n-1$ word to predict the n word is called the n-gram model). This method provides smart suggestions and saves time by moving away from grammar rules.

N-gram language model is a probabilistic language model based on Markov assumption, the beginnings was in 1913 with Markov in Markov [1913], who propose this technique, which called later markov chains, to predict from a roman if the next letter will be a vowel or a constant, for more histories check Jurafsky and Martin [2009].

There are tow approaches of them: Character based and word based, where the concept is to extract n sequence of successive elements from a text (Majumder et al. [2002]).

The idea of n-grams is making a realizable solution to keep the history of almost all known words (preceding) by just the $n-1$ preceding words, according to Markov assumption:

$$P(w|w_1, ..., w_{L-1}) \approx P(w|w_{L-n+1}, ..., w_{L-1}) \tag{2.23}$$

Where L is the position of the current word.

For example with unigram the probability of the sentence " I read a paper " became: $P(I - read - a - paper) = P(I).P(read).P(a).P(paper)$

Many works used N-grams for prediction, we mention some of them in the following.

[1]PAL is an abbreviation for predictive adaptive lexicon which is a communication aid and keyboard emulator developed at Dundee university by Arnott et al. in 1984

Ghayoomi and Assi [2005] implemented a word predictor for the Persian language that uses unigram, bigram and trigram word model. Their system achieves 57.57 % of keystroke savings and 24.42 % of HR where the number of suggestions is 9.

Haque et al. [2016] applied N-gram language model specifically unigram, bigram and trigram, back-off and deleted interpolation techniques to predict Bangla words in a sentence and used large data set of text word collected from different news papers. The results showed that trigram, backoff and deleted interpolation performed almost in the same trend-line where their accuracy was 63.04, 63.50, 62.86. While Bigram perform modestly with an accuracy of 45.84 and unigram performance was very poor with an accuracy of 21.24.

Bhuyan and Sarma [2018] used N-gram language model to predict the next word that the user wants to write in their Automatic Formation, Termination and Correction system of Assamese words but it can also be used for other languages. After comparing different n-gram models, the results showed that trigram and quadrigram are almost similar, but corpus size for them has a big difference. They concluded that for faster computers it is better to use a quadrigram, but for economical computers the trigram or bigram is better.

Let us mention other researches that used n-grams (as each had a point of view) like Lesher et al. [1999] who studied the effects of n-gram order and training text size on word prediction, others focus on complex languages like Gao et al. [2002] who applied trigram language model to Chinese. Others concentrate on corpus like Trnka and McCoy [2007] how analyzed the effect of in-domain and out-of-domain data on word prediction with trigram model.

Skip-Gram Language Model

Despite the effectiveness of smoothing techniques in solving the data sparsity problem, it has some limitations. For this reason, other techniques were sought to overcome this problem, among them we find the skip-gram language model.

Skip-grams come from the concept of introducing gaps in n-grams. Skip-grams incorporate long distance relations between words beyond the level of n consecutive words without an exponential increase in the parameter space. It allows to skip k tokens while predicting the context of a word (Pickhardt et al. [2014], Aggarwal [2018b]).

Pickhardt et al. [2014] provided a generalized language model (GLM) based on skip n-gram models interpolated using modified Kneser-Ney smoothing. Experiment over English text corpora led to a substantial reduction of perplexity between 3.1% and 12.7% in comparison to traditional language models using modified kneser-Ney (MKN) where for higher orders the GLM outperforms MKN for all test cases and the investigation over three other languages and a domain specific corpus gave a consistent improvements moreover the experiments on a small training data set of only 736 KB text gave an improvements of even 25.7% reduction of perplexity.

Hidden Markov Models

Hidden Markov Models (HMM) presented in Baum and Petrie [1966] are considered as statistical modeling techniques often used for sequences or time series processing (Eddy [1996]). They are defined in Rabiner and Juang [1986] as following:

"An HMM is a doubly stochastic process with an underlying stochastic process that is not observable (it is hidden), but can only be observed through another set of stochastic processes that produce the sequence of observed symbols."

Jordan et al. [2020] presented an HMM-based word prediction method to predict words with virtual keyboard. The results showed a reduction in the total amount of clicks with 26.3% and 61% user typed the text in less time moreover 90.7% of the desired words were displayed.

2.6.2 Knowledge-based Modeling

One of the disadvantages of systems based on statistical modeling is that it predicts words syntactically, semantically, or pragmatically inappropriate which makes it difficult for the user to choose the intended word. As a solution, knowledge-based modeling omits some inappropriate words, making the results more specific and relevant to the user. The used linguistic knowledge in prediction systems is syntactic, semantic, and pragmatic (Ghayoomi and Momtazi [2009], Makkar et al. [2015]).

Syntactic Prediction

The syntactic structure of the language is used to present syntactically appropriate words where part of speech (POS) tags are determined and the syntactic knowledge is used by the prediction system which makes it more accurate and knowing the type of the following words (Ghayoomi and Momtazi [2009], Makkar et al. [2015]). Syntactic prediction methods include Statistical syntax and rule-based grammars.

1. **Statistical Syntax:** It uses both syntactic categories sequence and POS tags to make predictions where the appearance of word is estimated using n-gram word tags. POS tags is sufficient in the simplest method where the probability of each predicted word is estimated by its probability of existing in a specific position along with its tag with respect to the past word tag. In another approach the probability of each predicted word is estimated according to the past word and its POS tag and the POS tag of the word before the past word, this means that we use word bigram and POS trigram model.

   Part-Of-Speech (POS) Tagging: Tagging sets a description to a token, this description can be a semantic information or part of speech and is known as tag (Indurkhya and Damerau [2010]). Part Of Speech Tagging means labeling words with their appropriate Part of speech. This part of speeches: noun, verb, adjective, etc.

   Fazly [2002] implemented word unigram and bigram and introduced new prediction algorithms which exploit POS tag information and compaired them with wordQ [2]. Results showed that all the predictors outperform wordQ where the syntactic

[2]WordQ is a commercial word prediction program.

predictors gives the best results, and using only POS info gives results better than unigram but worse than bigram. When n=5 and taking into account that predictions may be repeated, the linear combination with $\alpha = 0.8$ gives the best result with a HR of 37.78 % and KSR of 53.14 % noting that the tags-and-words model gives a results close to it.

After a year Fazly and Hirst [2003] investigate the effect of incorporating syntactic information into a word completion algorithm. They introduced the two algorithms that were explained in Fazly [2002]. Tag-and-words algorithm which combined tag trigrams and word bigrams and linear combination algorithm that combines tag trigram and word bigram models. The results showed that the linear combination algorithm outperforms all other with 36.23% HR and 51.98% in KS.

Aliprandi et al. [2008] enhanced Fastype which is a word prediction system designed to predict words for inflected languages with word and Part-of-Speech n-gram language models and a new interface called DonKey. The added POS n-grams and Tagged Word (TW) n-grams where the tagged word n-gram model extends word n-gram model by adding POS information. They use a linear combination algorithm to combine POS trigrams and simple word bigrams and tested their enhanced system on Italian language. They reached a Keystroke Saving up to 51% for a standard prediction list of length 10.

C. Spiccia et al. in Spiccia et al. [2015] proposed a two steps word prediction method based on posgrams to predict the missing words in sentences.
The first is a preparatory step which aimed to predict the part of speech of the missing word by using a posgram lookup table, this information is used in the second step to predict the word. The approach is compared with three traditionally word prediction algorithms: n-grams, Latent Semantic Analysis (LSA) and random choice.

They train the model for the Italian and English languages and they train only the POS, where they test it with posgrams, always noun and random choice; the posgrams results were the higher with 43.2% accuracy for the Italian language and 45.0% for the English language.

They also compare the prediction using the two steps and without (traditional prediction), the n-grams make the higher results constantly, where it results without the first step an accuracy of 50.3% for Italian language and 57.8% for English language, but with the preparatory step the accuracy improved for the Italian language, it becomes 51.1% and for the English language it remains stable.

Cavalieri et al. [2016] proposed an interpolation model, which combines word-based n-gram language model with m-POS-based language model where $m < n$. The resulted m-POS interpolation model was evaluated on three different languages: Portuguese, Spanish and English. The results showed an improvements in the KSS, HR and PP parameters, with 1 and 5 words in the prediction lists.

2. **Rule-Based Grammar:** We use the grammatical rules of the language in this approach where the sentence will be parsed using the grammar to know its categories.

The parsing may be top-down or bottom-up and the methods based on grammatical rules are: Phrase Structure Rule Grammar (PSRG), Context Free Grammar (CFG), and Head-driven Phrase Structure Grammar (HPSG). Usually the rules are structured as follows:

$$LEFT \rightarrow [RIGHT]+ \tag{2.24}$$

The previous equation shows that the left part of the rule may be decomposed into the categories in the right part in the order in which they appears, where the right part contains at least one category and all categories are defined (Garay-Vitoria and Abascal [2006]).

Hunnicutt [1986] was one of the firsts who used word frequency with simple structure grammar to predict words in Text-To-Speech system.

Morris et al. [1992] extended PAL with syntactic information and named the new enhanced version Syntax PAL. This version is considered the first prediction system that uses syntax. Syntax PAL performs a syntactic parse of the partial sentence that has been typed to predict a syntactically correct words. It reduced the effort significantly for users with motor dysfunctions, but had a problem when users make syntactic errors.

McCoy and Demasco [1995] presented an intelligent word prediction system based on syntactic information. They implement a grammar of English in a transition network formalism that allows several parsers to be pursued in parallel which assist in predicting. Their long term goal is to add semantic and pragmatic information.

Wood and Lewis [1996] developed and implemented a syntactic pre-processor by using context free grammar along with phrase-structure-rule grammar and simple parsing technique to predict syntactically correct words then apply statistical prediction to order the suggestion list. They create a platform know as WINDMILL to test their technique and compared it with no prediction, statistical and syntactic techniques. The proposed technique gave the better results with 55.1% of Keystroke saving.

(a) **Chart Parsing Method:** Is a bottom-up parsing technique proposed by Allen [1987]. The bottom-up parsing uses the rule and match the sequence of symbols to the right part then identifies them as the left symbol from the point of view of the key wich is a special symbol. The parser require the key to complete or extend a rule after finding the rule that starts with that key. The chart is a special structure that is used by buttom-up parser to keep records of its state. The chart also store the previously matched but not complete rules (Garay-Vitoria and González-Abascal [1994]).

    In this work Garay-Vitoria and González-Abascal [1994], the authors proposed a new approach based on the chart bottom-up technique and syntactic prediction using grammars. They noted that the existing methods such as syntactic approaches make best results than the frequency approaches but this last need simple computation effort than syntactic approaches.

The idea is to associate each word with the semantic information of the conversation theme and to enrich the syntactic approach with semantic categories, which leads to minimize the possible words, which should enhance the hit rate.

They test four prediction methods: word frequencies, syntactic with automaton, syntactic chart and semantic-based chart, on two Spanish texts. For the first text (994 character) and the second (2444 character), the syntactic with automaton and syntactic chart score the best saving percentage (they was quasi similar).

This approach rise a new problem. It is constrained to the topic which it trained on, so the performance of the method decrease if the model changes.

They think that the promising approach which can make best results is the neural network approach because it's similar to the natural human approach in prediction.

Garay-Vitoria and Gonzalez-Abascal [1997] developed a word prediction method based on syntactical analysis of a sentence using the chart parsing method. Word information is stored in dictionary that adapts itself according to the user's vocabulary, this makes the system adaptable. They tested their system on four cases and it gives better results than the purely statistical one but the required computational effort is higher. The word faced some problems as the new words added to the lexicon moreover there is some possible dysfunctions if a user does not look at the proposals offered by the predictor.

(b) **Chunk Parsing Method:** (Abney, 1991, and Abney, 1996) is a simple parsing strategy based on chunking by identifying base phrases and converting them to non-terminal symbols then chunking them after that converting them again into non-terminal symbols. This procedure is repeated until there are no phrases to be chunked. The complete parsing tree is then building on the basis of chunking results (Tsuruoka and Tsujii [2005]).

Schadle [2004] present Sibyl[3] that uses two predictive modules sibyletter which applies n-gram model on letters and sibyword which predict the next word using chunk parsing method where the predictions are based on the last words and the last head chunks. The results show that for a list of 5 words, the keystroke savings reach 57%.

(c) **Automata:** Automatas are graphical representations of tables that contains information about categories which follow each of the category. Its also takes into account the syntactic category of the related words.

Garay and Abascal [1994] used two approachs which are purely statistical that uses unigram and syntactical approach which uses bouth statistical information and syntactical information by using an automaton to predict words and

[3]Sibyl is an Augmentative and Alternative Communication computer system that aims at improving text typing for persons with sever speech and motor impairments.

compare between the results of the used approaches. The syntactical approach gives better result than purely statistical one but had a problem with ambiguous words and new words, those problems can be solved by creating new categories for ambiguous cases and special category for new words.

(d) **Token Automata:** Is a hybrid system based on both symbolic and statistical approaches. Statistical part studies word tokens and symbolic part based on sentence schema and acceptability notions.

Maurel and Le Pévédic [2001] used token automata to get token number of syntactic categories and memorize token number of words in dictionary then compute words probability on the basis of those numbers. These system is incorporated in HandiAS [4] and improve the clicking action saving by more then 43%. HandiAS is independent of the language so they plain to work on English, German and Italian versions.

(e) **Dependency Language Model:** These kind of language models use dependency grammar where they generate words along paths in the dependency tree of the sentence.

Gubbins and Vlachos [2013] uses dependency language models to tackle the sentence completion task. They apply the approach to MSR completion challenge [5]. They used labeled and unlabeled dependency language models where both models outperforms *n*-gram language model and the four labeled dependency model gives the best result which is 8.7 points in accuracy better than the best *n*-gram model.

Semantic Prediction

Syntactic prediction is not sufficient since the predicted item may be syntactically correct but wrong in the semantic side. So attempts are made to suggest words that are both syntactically and semantically correct. Semantic knowledge is added by assigning categories to the existing words in the corpus and find a set of rules that constrain the possible next candidate word (Ghayoomi and Momtazi [2009], Makkar et al. [2015]).

Two methods are used for semantic prediction. The first is using lexical source like WordNet in English which measures the semantic probability of words, the second is the use of lexical chain which gives a high priority to the words which are related semantically.

This method is rarely used in word prediction because it requires complex hand coding and may consume time (Makkar et al. [2015]).

1. **Semantic Grammar:** Semantic information was used initially by categorizing words semantically and defining semantic rules and grammars.

   Hunnicutt [1989] present a study about incorporating syntactic and semantic information in the prediction task. For semantic, a study of marking Swedish lexicon

[4]HandiAS is a prototype software for disabled communication aid, and it is a part of the Research project CNHL of the LI, the Computer Laboratory of the Tours University 2001.

[5]MSR completion challenge is a sentence completion challenge launched by Microsoft in 2011

with semantic categories to optimize ranking in the list of predicted words was done. The tests showed a small savings in keystrokes for syntactic information, and no savings for low-level semantic information.

Ghayoomi and Daroodi [2008] designed three word predictors for the Persian language. Statistical, syntactic with POS tags and the third uses syntactic categories with their morphological, syntactic and semantic subcategories. The best results was given by the third system that uses syntactic and semantic information where it achieves 42% savings in KS.

2. **Latent Semantic Analysis\Indexing (LSA\I):** LSA is a statistical method for extracting and representing usage and meaning of word. It is based on singular value decomposition which is a mathematical matrix decomposition technique. The meanings derived by LSA are capable of simulating a variety of human cognitive phenomena (Landauer et al. [1998]), therefore it is used to extract semantic knowledge of words in word prediction task.

   Zweig and Burges [2011] present MSR sentence completion challenge and test it using models based on n-gram statistics and average LSA similarity where they treated each sentence in the training data as a "document" and performed LSA. The results showed that the model based on LSA performed better than n-gram models and gave 49% correct suggestions.

   Spiccia et al. [2015] proposed an alternative methodology based on LSA for automatic sentence completion. They use a word-word frequency matrix to achieve higher scalability with large training data and use it in addition to a word-document matrix to achieve the best accuracy where they reached 52.3% accuracy.

2.6.3 Heuristic Modeling

Heuristic modeling, also known as adaptation, as its name suggests it adapts the system according to the user making the predictions provided appropriate for him. It have been incorporated in many word predictors and used in many word prediction methods to improve results. There is short-term learning and long-term learning methods (Ghayoomi and Momtazi [2009], Makkar et al. [2015]).

Short-Term Learning

The system is adapted according to the user in the text that he is currently writing. The most common methods used in adaptation include recency promotion, topic guidance, trigger and target, and n-gram cache:

1. **Recency Promotion:** It means that the word that occurred in the text has a high probability that it will be used in the text again. This method takes into account the written words and newly used words, which makes it able to adapt to the requirements of the user, but ignores the grammatical structure when predicting.

2. **Topic Guidance:** This approach adapts to the general theme of the text, as it adds a specific dictionary for the text field that contains repeated words in this field in addition to the general dictionary.

3. **Trigger and Target:** This method is based on the principle of the correlation of words to each other. When a word occurs, it is called a trigger, and it triggers the operation of another word, which is called the target.

4. *N*-**Gram Cache:** This approach is based on the assumption that the once used word may be used again as it captures frequently used words by using the n-gram cache in which the words are placed to gain an increased possibility.

Long-Term Learning

Unlike the previous method, this takes into account the texts previously produced by the user in addition to the one currently produced. The most common methods are adding new words, automatic capitalization, providing inflected form of words, and compounding:

1. **Adding New Words:** This method includes adding words that are unknown to the system and that the user is writing to the system's dictionary to call them in the list of predictions when needed.

2. **Automatic Capitalization:** It allows the user to save more keystrokes by capitalizing the letters that should be capitalized depending on the system language.

3. **Providing Inflected Form of Words:** The system takes into account the inflected forms of words in the case of very inflected languages like German, which makes it more efficient for the user and gives higher percentage of keystrokes saving.

4. **Compounding:** This method adds the compounding, that is the formation of new words from other words to make it easier for the user to write them. Compound words are written as a single unit and used in many languages, such as German.

   Baroni et al. [2002] presented a solution to the compounds problem in German language. The designed word predictor predict the compound by splitting the compounded word into modifier (the left element) and head (the right element) and predicting each part alone. The results showed that the simple model achieved a KSR of 51.5% where the split-compound model achieved a combined KSR of 57.9%.

   Matiasek et al. [2002] presented Fasty [6] that uses combination of different n-gram models and the split-compound model presented in Baroni et al. [2002] with collocation-based predictions based on the heuristic approach of trigger and target. The system achieved a prominent results.

   Hunnicutt and Carlberger [2001] design and implement a word predictor for Swedish. They use probabilistic language model and achieved a KS saving of 46%. They discuss the use of tag model which increase KS savings by 4.2% for one prediction and 2.8% for five predictions, and the use of heuristic models to improve both quality of word prediction and KS savings. Adding new words or new words learning method increased the keystroke savings by 3.2% for one prediction and by 5.1% for five predictions where the recency promotion increased the keystroke savings by 1.0% for one prediction and by 2.1% for five predictions in addition the use of automatic capitalization to monitor the user's employment of uppercase letters in a

[6]Fasty is an intelligent statistically based adaptive word prediction program which is under development for the German, French, Dutch and Swedish language which is expected to be released in 2003

word which is called case sensitivity improved KS for one prediction by 0.5% and for five predictions by 0.2% moreover the method of providing inflected form of words increased KS for one prediction was 0.1% and for five predictions 0.3%.

There's also other classical techniques, the next works describe them:

The paper Even-Zohar and Roth [2000] study learning approach for the problems that need a large number of inputs but actually just a few of them are necessary for the decision. The SNoW architecture is used, they choose the verb prediction task; where a confusion set is constructed by coupled two verbs, so the predictor choose the verb the most likely appeared in the sentence according to the probability and the part of speech. In average, the approach improves the error rate. The success depends on using a large expressive features and a learning approach suitable to it.

The authors of Kapse and Shrawankar [2013] presented a word prediction model for braille users. This work based on matching prefixes to predict word using B+ tree. At the first step, they extract prefixes from data set, then they index it by using ASCII binary encoding technique and a leftshift and xor operations. The B+ tree is then constructed to link the indexes with words. By using four characters for prefix, the model saves over 50% of time typing.

2.7 Machine Learning Prediction Methods

Different ML techniques have been used in word prediction. We mention the most used ones in the next section.

The word prediction problem have been seen as a supervised learning problem. The most used supervised Learning algorithms in word prediction are:

2.7.1 Decision Trees

Decision trees are machine learning technique that can be considered as decision support tool used for classification or regression, where they use a binary rules to calculate a target value. The values are represented in a tree form (Horning [2013]).

An optimal binary-tree is likely to achieve results much better than an optimal n-gram model which is considered as a special case of binary-tree model. The object of this architecture is to maximize the precision of the decided result (Bahl et al. [1989]).

The model proposed by Bahl et al. [1989] predicts the next word the speaker will say according to the previous spoken words based on a greedy algorithm with questions as restrictions: at each node the question which gives minimum entropy is selected. The model predict the 21^{st} word given the 20 previous words; in order to avoid the over-fitting of trees the experiment was as follow: they train the tree construction on a data set of 10 million words, test the reduction of entropy on another 10 million words, 9 million words to compute the smoothed leaf probability distributions, and test the model on $\approx$ 1 million words. The perplexity of the suggested tree gives 90.7, where an equivalent trigram language model gives 94.9. They also create a language model which combine the tree and the trigram, where the perplexity was 82.5 (lower than both). So, the tree based language model achieves best results than a trigram language model, but it is more effective if it is used as an adjunct to the trigram model.

IGTree: Daelemans et al. [1997] is a top-down induction algorithm of decision trees where it compresses a database of labeled examples into a lossless-compression decision-tree structure (Van Den Bosch [2006]). He presented a classification-based word prediction model based on IGTree. Experiments showed that the system exhibits log-linear increases in prediction accuracy and decreases in discrete perplexity, they reach 42.2% correctly predicted tokens on the same type of text when trained on 30 million words of newswire text. Better word prediction accuracy can be attained simply by adding more training examples, where the best rate observed was an 8% increase in performance.

2.7.2 Support Vector Machine

Support Vector Machines (SVM) are supervised machine learning algorithms first proposed by V. N. Vapnik and A. Ya. Chervonenkis in the framework of the "Generalised Portrait Method" for computer learning and pattern recognition in 1962 and they were first published in 1964. SVMs started and become famous when statistical learning theory was developed further by Vapnik (1979) (Chervonenkis [2013], Burges [1998]).

SVM provide higher performance than traditional machines learning algorithms, they are powerful tools for solving classification problems where they map the input points into a high-dimensional feature space and try to find a separating hyperplane that maximizes the margin between two classes in this space (Lin and Wang [2002]).

Al-Mubaid [2007] presented a learning-classification based method for word prediction using context features and machine learning. He use a combination of SVM with various feature selection techniques MI, X^2 and two other feature selection techniques adapted from MI: MI_1 and MI_2. The results showed that the method is effective in predicting the correct words by utilizing small contexts; and the system achieved an accuracy of 91.42% correct predictions with MI_2 .

Al-Mubaid and Chen [2008] presented a word prediction method as an assistive technique for people with mechanical disability to improve textual information entry. They applied SVM and Lsquare into word disambiguation task that is extended and adapted to the word prediction problem for text completion with adaptive learning methods. Supervised learning methods have been integrated with adaptive learning to produce robust learning techniques. They made two evaluations where the first gives 87.9% accuracy with X^2 and vectors of size 40; and for the second the accuracy, precision and recall rates approach or exceed 99% which indicate that their method is highly effective.

2.7.3 Naive Bayes and Bayesian Networks

Naive Bayes is a probabilistic model developed through bayesian networks. It states that its variables can be divided into cause and effect where the effects are independent between themselves.

Goulart et al. [2018] proposed a hybrid word prediction model based on Naive Bayes and LSA. They first created and trained LSA and Naive Bayes sub-models and then performed inferences to predict a word usage probability in the analyzed context based on n of the previously used words and combined the models, and finally they performed the optimization through supervised training. The results in the Microsoft Research Sentence Completion Challenge show that their hybrid model gives 44.2% accuracy.

Troiano et al. [2017] built a predictive model based on bayesian networks which predict the user input when a form is filled according to the past interactions to reduce the amount of time required to fill a form. Results of experimentation proved that the proposed approach is able to provide an effective support in filling a form. The limitation of the model is in the single form context: it does not allow reuse parts of acquired knowledge across different forms. The combination with the method based on ontology might solve this problem.

2.7.4 K-Nearest-Neighbor

K-Nearest-Neighbor (KNN) is a supervised machine learning technique based on the idea that the nearest patterns to a target patterns gives the best label information about them. The K-nearest patterns are assigned to the same class label by KNN where the choice of K defines the locality of KNN (Kramer [2013]).

Kuo [2011] presents a novel simple task of word prediction given a piece of text and other tasks. They compare a version of Co-occurrence (Naive Bayes), two versions of K-Nearest-Neighbor method and Latent semantic indexing, against a baseline algorithm using different datasets on different tasks.

The results of next word prediction task on MED dataset showed that using different simularity functions affects the KNN performance. The K parameter affects the predictions for the KNN and LSI, where for the KNN the K is the number of neighbors and for LSI is the dimensions which represent the number of unique words. Also the number of predicted words affects the KNN and LSI where generally the best k for predicting one word remains the best for predicting many words.

For the ArnetMiner dataset it seems that KNN with citation did not give results better than the KNN method with the randomly sampled Arnet-Miner data, also the LSI did not perform well on this dataset where the baseline algorithm gives better results than it, and for the StackOverflow data the KNN gives the best results.

2.7.5 Neural Networks

We mentioned previously that N-gram language models have a major disadvantage where they assign probabilities of 0 to n-grams that do not appear in the training corpus, this problem was solved with smoothing techniques. However, there are still other problems where the main is the curse of dimensionality, which in order to solve it the neural networks was presented for modeling language in continuous space (Jing et al. [2019]).

Xu and Rudnicky [2000] investigated the use of artificial neural networks to build language models. Their network has $|V|$ input and output units where $|V|$ is the vocabulary size and they use the logarithm of perplexity as error function, for the output units they used softmax activation function. They trained their model with back-propagation algorithm where it performed better than n-gram language model, but gives a poor generalization ability.

Bengio et al. propose in Bengio et al. [2003] a NN approach for probabilistic language modeling, the goal is fighting what they named “the curse of dimensionality” which means that the model in testing may face a different sequence from all sequences that it was trained on.

The idea was make each sentence in training lead the model to learn an exponential number of other sentences with similar semantics; so the model learns in parallel the representations of each word and the probability function for word sequences. The architecture proposed by them is shown in Figure 2.3.

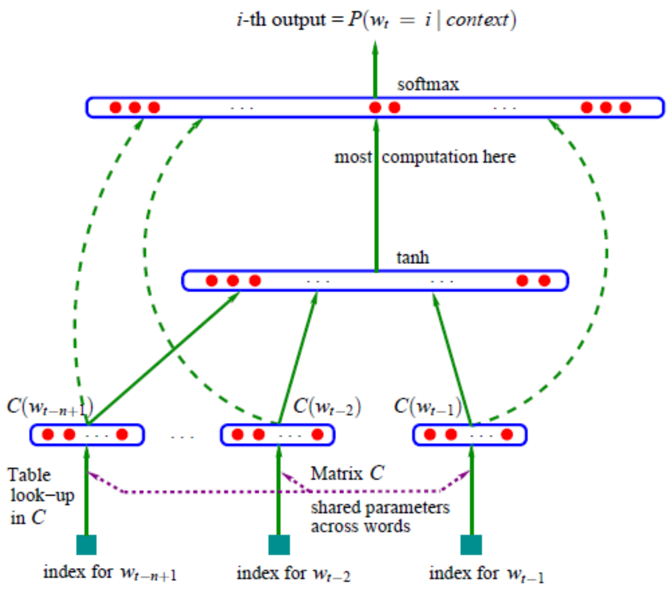


Figure 2.3: FFNNLM arhitecture (Bengio et al. [2003]).

The Figure 2.3 shows that to predict the probability of the word w_i the previous $n-1$ words are projected by C which is the projection matrix that represent the distributed feature vectors associated with each word in the vocabulary.

The paper shows that using neural networks make better results compared with the best of n-grams where the difference in the test perplexity for the Brown data set was 24% and 8% for the Associated Press News data set.

Bengio et al. opened the door to the use of NNs in language modeling and proposed the idea of using RNN for LMs (Jing et al. [2019]). After that, several researches emerged that used neural networks in language modeling, especially in word prediction.

Mnih and Teh [2012] propose a simple algorithm for training Neural Probabilistic Language Models (NPLMs) based on noise-contrastive estimation, a procedure for estimating unnormalized continuous distributions. They investigate the behavior of the algorithm on the Penn Treebank corpus and show that it reduces the training times. They train several neural language models on a 47M-word corpus with a 80K-word vocabulary and test the models on MSR Completion Challenge dataset where they record a new accuracy for the dataset at 54.7%.

There are also those who have combined many machine learning techniques, as we mention:

Cavalieri et al. [2010] D. C. Cavalier et al. who propose an adaptation of existing POS prediction ML algorithms as statistical POS model, Naive Bayes, SVM, LR... and a fusion algorithm in order to improve the performance of Portuguese word prediction. The idea of fusion is to combine algorithms which sometimes perform better than other and sometimes the opposite in order to achieve the best performance. The results of the proposed fusion algorithm showed an increase of 3.42% of keystrokes saved compared with N-gram which don't use grammatical information. The Portuguese POS prediction algorithms evaluated here gave an improvement between 0.92% and 3.42% in the keystrokes saved.

2.8 Deep Learning Prediction Methods

Deep learning was used to solve many problems, including the problems of natural language processing so the next word prediction, as recurrent and convolutional neural networks were used in many work to improve its efficiency, we describe some of them bellow.

Recurrent neural networks are known to be the most suitable for dealing with text data, as they solve problems that need previous knowledge or experience, which made them one of the most used methods for word prediction task.

Mikolov et al. [2010] introduced the first recurrent neural network language model that could be trained by the backpropagation through time. The model overcome FFNNLM and n-gram LM. Note that the LSTM was firstly introduced into LM by Sundermeyer et al. [2012].

In 2013 Mikolov et al. in Mikolov et al. [2013] presented a Skip-gram architecture and explore its performance on Microsoft Sentence Completion Challenge where it did not perform better than LSA similarity, but in combination with RNN-LM it gives the best result with an accuracy of 58.9 %.

Yu et al. inYu et al. [2017] presented a word prediction method for mobile devices based on embedded deep learning. They propose a pipeline which compress the RNN-LM model, this pipeline composed by knowledge distillation (of Hinton et al. [2015]) phase and compression and retraining phase; the Figure 2.4 describe the proposed model.

The results showed that the proposed method improves the KSS and the WPR compared with other commercialized keyboard using manually curated dataset, where it scored the highest rate equal to 65.11% KSS and 34.38% WPR, when the Apple keyboard scored 64.35% KSS and 33.73% WPR, then Swiftkey with 62.39% KSS and 31.14% WPR, after that Samsung with 59.81% KSS and 28.84% WPR, finally Google with 58.89% KSS and 28.02% WPR.
The proposed method satisfies the mobile device memory and runtime constrains. Authors want to make it adaptable to user choices in future work and combine it with n-grams statistics for the full advantage.

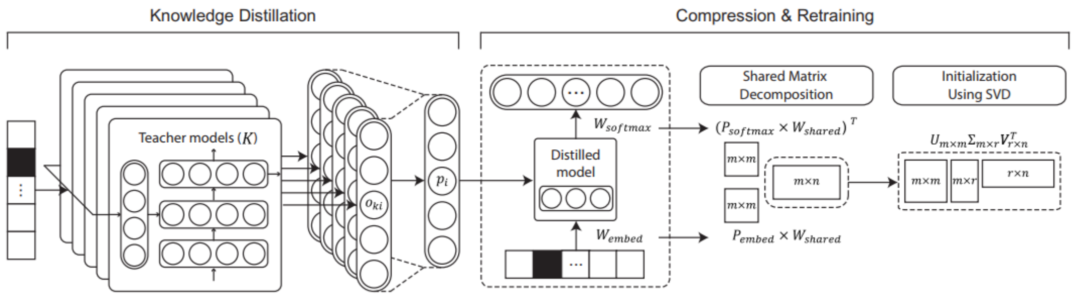


Figure 2.4: Overview of the proposed embedded DL architecture (Yu et al. [2017]).

Another work for mobile device, Grachev et al. [2019] propose a compressed RNNs model for efficient language modeling in real-time offline mobile applications, because of the complexity of the RNN running and memory requirement, and the high dimensional output. This work is one of the first models of RNNs for mobile GPUs and also compressed.

They test different techniques of compression such as pruning, quantization, low-rank matrix factorization and Tensor Train decomposition on different types of RNNs such as RNN, LSTM and GRU. Their experiments show that the low-rank (LR) compression for the model LSTM650-650 is more efficient.

An LSTM model to predict a next word in Assamese language is presented by Barman et al in their paper Barman and Boruah [2018]. According to the complexity of this language, the performance of LSTM is lower than with low inflected languages like English. They have developed a solution facing this problem by storing the phonetics of the Assamese words.

So they used two data set: The first is a collection of words and the second is the phonetics of those words, noting that the number of words increases in the second data set because of the high inflection of the language.

They experiments the model with different number of hidden layer, neurons in each one, learning rate and epochs. The best result for the first data set was 88.20% average accuracy when they use 2 hidden layers with 128 neurons with a learning rate equal to 0.001 for 98000 epochs. And it makes 72.10% accuracy with the second data set when they use 3 hidden layers with 256 neurons with a learning rate equal to 0.001 for 77000 epochs.

The results showed that the maximum accuracy was with the first data set, where they attributed it to its size.

Boldt [2017] propose an LSTM model to predict the next word in a java code, trained with one English dataset and four JAVA codes datasets such as PTB, JDK.

The LSTM model result less perplexity with the Java datasets than the English dataset, where it achieve under 22% of perplexity and above 0.47 of accuracy comparing to the English which score 85% of perplexity and 0.27 accuracy.

The work was for the general representation of the code, it does not account the name of variables or type etc, so as a future work, they want to make in consideration those information, and to compare the model with other programming language.

Oualil et al in Oualil et al. [2017] describe a new approach of multi span architecture that merge the long (which represent the global) and the short(which represent the local) context information through a recurrent network Long Short Range Context (LSRC) which is an adaptation of the LSTM architecture. The idea is to learn both the global and local constraints of language by using two separates hidden layer, than combine their results to perform the word prediction.

The model was experimented using two dataset: Penn Treebank (PTB) and Large Text Compression Benchmark (LTCB); and compared with different LM approaches such as N-gram Kneser-Ney (KN), FFNN based LM, Fixed-size Ordinally Forgetting Encoding (FOFE) approach, the full RNN (without classes), deep RNN (D-RNN) and LSTM model.

The Figure 2.5 shows the results of perplexity for each one with the PTB dataset, which was almost the same with the LTCB dataset:

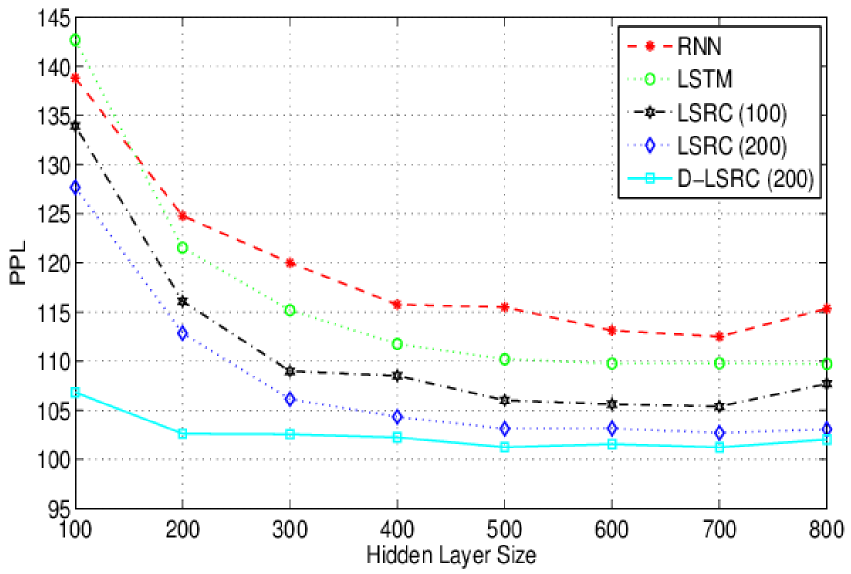


Figure 2.5: Perplexity of the different NN-based LMs with different hidden layer sizes on the PTB test set (Oualil et al. [2017]).

The Figure 2.5 shows that the LSRC model achieves the best result, where the deep LSRC with word embedding size set to 200 reduced it the most.

The novel architecture LSRC which combine the LSTM and RNN improves its effectiveness in reducing the perplexity compared with the most known architecture in the field.

Despite the fact that RNNs are the most used in word prediction task, researchers have also turned to the use of convolutional neural networks, where they have also shown high efficiency.

Wang et al. [2015] was the first work that uses the CNN for word sequence prediction. They propose a novel convolutional architecture named genCNN. Their extensive experiments on text generation and n-best re-ranking in machine translation show that genCNN outperforms the state-of-the-arts with big margins.

Pham et al. [2016] explore also the use of CNN for language modeling where they proposed a CNN-based language models. They incorporate a CNN layer on top of a strong feed-forward model. The results show a reduction in perplexity of 11% to 26% when the model uses MultiLayer Perceptron (MLP) Convolution and combines kernels of different window sizes. This improvement open the door for the use of CNN to LM.

The paper Souri et al. [2019] presents CNN architecture to predict the missing text in Arabic language. First they prepared the data set (about 130MB of text), then they preprocess it to be inputs for the CNN architecture, so they transformed texts to numerical codes.

The architecture was composed of an input layer, two of convolutional layers with the activation function ReLU, two of pooling layers then a fully connected layer.

They train the model on texts of the same author, then train it on texts of the same data set, and finally they train it on all document confused. For the validation and testing, they repeat them with the same strategy. In the best case, the model achieves a 97.8% of accuracy. As a future work, they want to adapt the model to work on GPU.

To go far, there's a new architecture which is a kind of convolutional NN, developed by Lea et al. in Lea et al. [2017] called Temporal Convolutional Nets (TCN), which supposed to make a revolutional results in the field of sequence modeling.

This paper Bai et al. [2018] presents a TCN architecture that outperforms the LSTMs and GRUs in sequence modeling because of its longer memory. It is based on two principals: the length of outputs equal to the length of inputs and the output depend to the past only. The TCN model was compared with the other RNNs model such as RNN, LSTM and GRU for more than 10 sequence modeling tasks, the results was clearly that TCN was the best than others.

So the idea was to change the judgment that says RNNs are the best in the sequence modeling topics, we have here a generic convolutional network that performs so good called TCN.

2.9 Conclusion

In this chapter, we presented a look at the most famous techniques used in next word prediction, as we mentioned a number of researches that dealt with this task, where the most famous and newest of these techniques are RNN and TCN, which we will focus on and use to build a word prediction model in the next chapter.

Chapter 3

Experiment

3.1 Introduction

In this chapter we present our experience to build some next word prediction models. These models use RNN and TCN-based architecture for deep learning.

We describe first the proposed network architectures then the environment and the data set that we worked on, then we move on to provide the implementation steps, after that we discuss the obtained results.

3.2 Networks Architecture

Inspired by the neurons work in the previous chapter, we have used two deep learning networks namely RNN and TCN. These two architecture are detailed in the following section.

3.2.1 Recurent Neural Network

For the RNN models, our proposed architecture to face the next word prediction problem illustrated in Figure 3.1.

The architecture contains three main layers: embedding, RNN and dense layer.

Embedding Layer

The embedding layer is used to obtain the word vectors where they are randomly selected. It takes as parameters the number of tokens which is in our case the vocabulary size, the embedding dimension and the input length.

RNN Layer

The RNN layer is considered as the core layer where it is responsible of processing data, it can take many entries but in our case we define just the number of hidden units. We have used two versions of RNN: SimpleRNN and bidirectional LSTM.

Dense Layer

The dense layer is responsible of processing the output, where it takes as parameter the units which is in our case the vocabulary size and the activation function where we

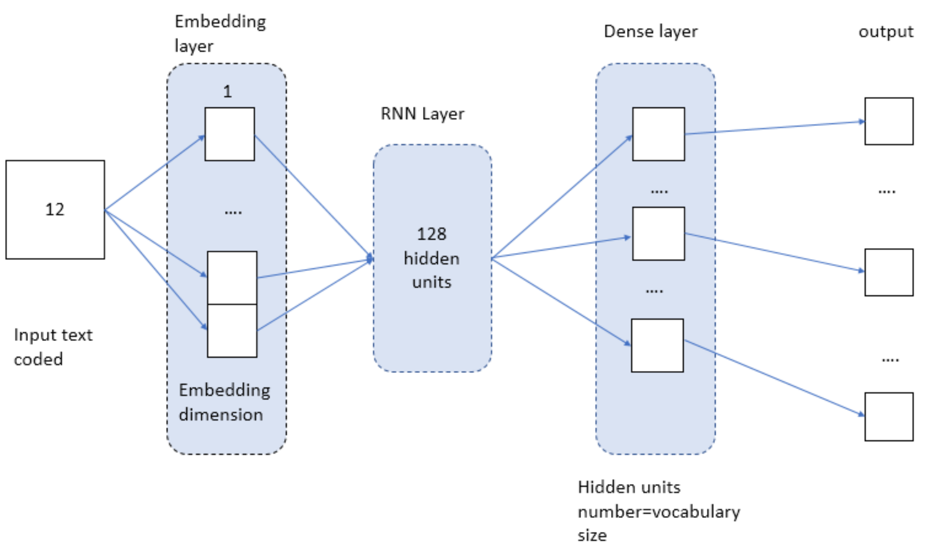


Figure 3.1: The proposed RNN Architecture

use the Softmax. It gives a probability for each class, where in our case the output classes is the vocabulary size, the class that contains high probability is the most likely next probable word.

3.2.2 Temporal Convolutional Networks

Temporal Convolutional Networks (TCNs) arose from the idea of using the advantages of CNNs and RNNs together as the idea appear in the paper Lea et al. [2016], and developed where in 2018 the famous structure know was proposed in Bai et al. [2018]. TCN has two main characteristics:

- It can take a sequence of any length and map it to produce a sequence of the same length like the RNNs.
- It use a causal convolutions which means that there is no leakage between information from future to past.

It has two main concepts:

Dilated Convolution

Dilated convolutions are used to explore large receptive field where for 1-D sequence input x and filter $f : 0, ..., k-1$, the dilated convolution operation F on the element s of the input sequence can be defined as:

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i).x_{s-d.i} \tag{3.1}$$

Where d is dilation factor and k is the size of the filter and $s - d.i$ are accounts for the direction of the past.

Residual Block

Residual block is employed in place of convolutional layer where each one contains a branch leading out to a series of transformations where its outputs are added to the input of the block.

Our proposed TCN model architecture is shown in the Figure 3.2.

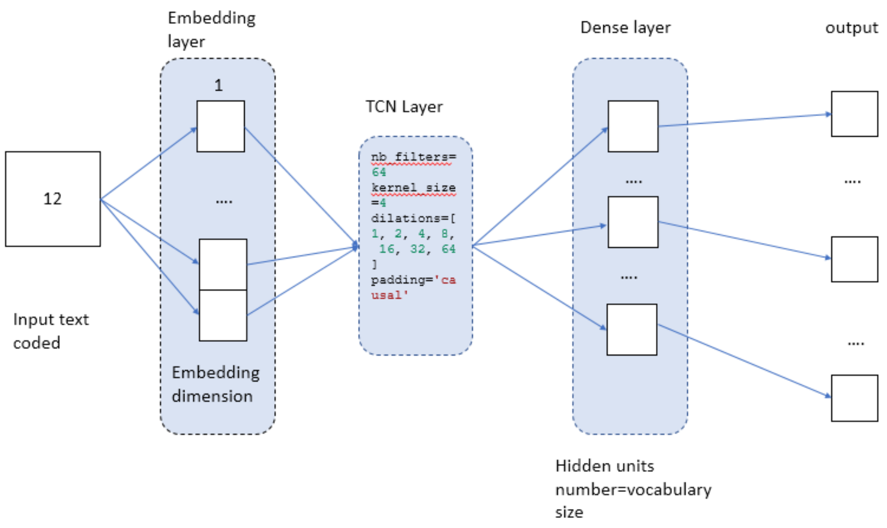


Figure 3.2: The proposed TCN architecture

The embedding and dense layers are the same as the first architecture.

TCN Layer

The TCN layer has many arguments, we define the followings:

- nb_filters: represent the number of used filters in the convolutional layers. We set it to 64.
- kernel_size: represent the size of the used kernel which is the size of filters in each convolutional layer. We set it to five for the second data set and four for the third data set.
- dilations: represent the list of the used dilations. We define the list as follow : [1, 2, 4, 8, 16, 32, 64].
- padding: represent the used padding in the convolutions where we set it to 'causal' for a causal network.

3.3 Environment

Due to the low efficiency of our computers, we used Google Colab and Kaggle to develop our deep learning models.

- Google Colab: we use it in the cleaning and pre-processing steps and in building and training models with the second and third data sets with the following proprieties:

  - RAM: 12.72 GB
  - Disk: 107.77 GB
  - CPU: There are two CPUs.
  - GPU: Nvidia Tesla K80.

- Kaggle: we use it in building and training models with first data set with the next propreties:
  - RAM: 16 GB
  - Disk: 4.9 GB
  - CPU: there are two CPUs as Google Colab.
  - GPU: Tesla P100.

We write our code using: Python3 programming language under jupyter notebook with the following libraries:

- **re:** is a library that provides regular expression matching operations similar to Perl.
- **Keras:** is neural-network library designed to enable fast experimentation with deep neural networks and it is supported in TensorFlow's core library since 2017.
- **io:** is a library designed to deal with various types of I/O and provides the Python interfaces to stream handling.
- **json:** is a library that provides a familiar API to users of the marshal and pickle modules in the standard library inspired by the syntax of literal JavaScript objects.
- **csv:** is a library that implements classes to read and write tabular data in CSV format.
- **Tensorflow:** is an open source library for fast numerical computing used to create deep learning models .
- **Matplotlib:** is a comprehensive plotting library where its numerical mathematics extension NumPy.
- **NumPy:** is scientific computing library that support large and multi-dimensional arrays and matrices with a large collection of high-level mathematical functions to operate on these arrays.
- **Keras-tcn:** is a library that supports the new TCN architecture.

3.4 Dataset

We used three data sets in this project.

- The first one is Coursera Swiftkey data obtained from: `https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip`. The data contains four folders:
  - de_DE: with size of 244 MB which contains data of Dutch language.

  - en_US: with size of 556 MB which contains data of English language.
  - fi_FI: with size of 217 MB which contains data of Finnish language.
  - ru_RU: with size of 325 MB which contains data of Russian language.

  Where each folder contains three files. We have used only the english data that contains:

  - en_US.blogs with size of 205,235 KB which contains data obtained from blogs.
  - en_US.news with size of 200,989 KB which contains data obtained from newspapers.
  - en_US.twitter with size of 163,189 KB which contains data obtained from Twitter.

  Our idea is to use multi-domain data of the first data set.

- The second data set is: nietzsche.txt obtained from `https://www.kaggle.com/heudanlv/nietzschetxt` with size of 586 KB from the book: Writings of Nietzsche: Volume 1 by Friedrich Nietzsche.

- The third data set is a News category from the Brown corpus in nltk library, description in `https://www.kaggle.com/nltkdata/brown-corpus`.

  For the third data, we took the idea of n -gram, where we used the last three words to predict the next word.

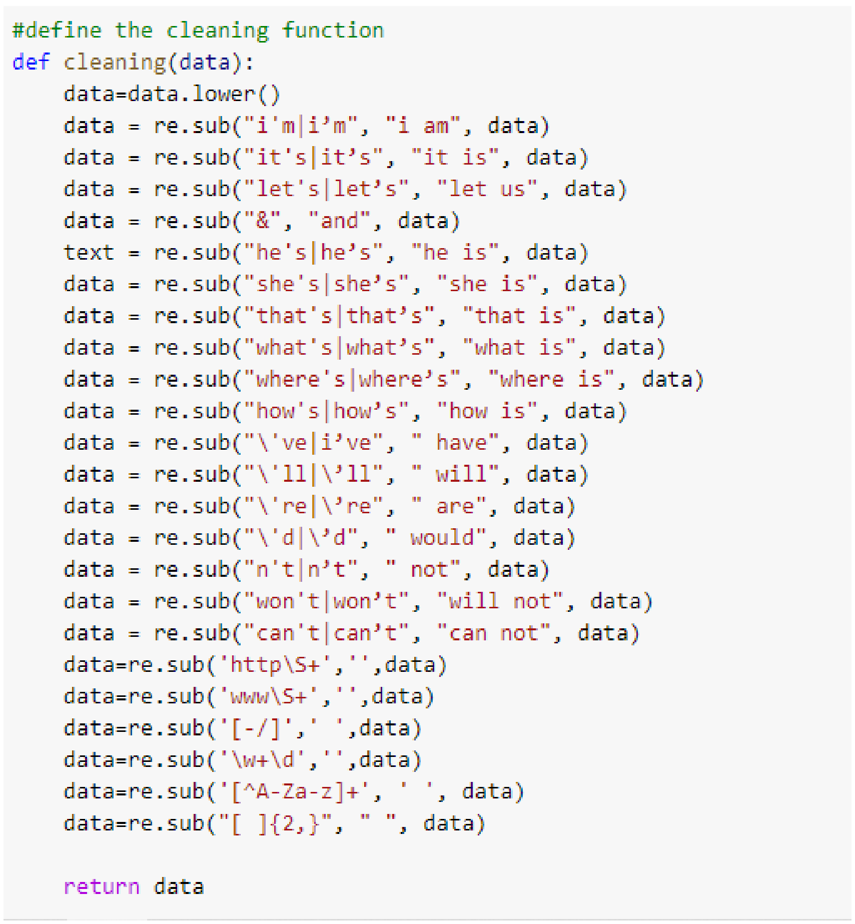
3.5 Implementation

For the implementation, we started with cleaning the data sets, where we cleaned the first and second data sets, while the third data set does not need to be cleaned, then we pre-processed all of them for training. **Cleaning**
We started with cleaning the data set from numbers, special characters and punctuation marks where the used cleaning function is shown in the Figure 3.3.

Preprocessing
For the preprocessing step we have eliminate the words with length bigger then 25 characters, then we split the data set, noting that we are using a specific percentage from the first data due to the limits of the used RAM, where we choose to use only 1% then 2% of each file.

After that we encode the data using Tokenizer function imported from keras.preprocessing.text and saved it to use it later.

Finally, we define the input and output labels. The input is the past word and the output is the next word for the first and second data sets and for the third data set the input is the past three words and the output is the next word. Then we split the data set, where for first data we split it into 60% for training, 20% for validation and 20% for testing; the second and the third data sets are split into 80% for training and 20% for test, where for the 80% used for training we split it using validation_split into 90% for training and 10% for validation.



```
#define the cleaning function
def cleaning(data):
    data=data.lower()
    data = re.sub("i'm|i’m", "i am", data)
    data = re.sub("it's|it’s", "it is", data)
    data = re.sub("let's|let’s", "let us", data)
    data = re.sub("&", "and", data)
    text = re.sub("he's|he’s", "he is", data)
    data = re.sub("she's|she’s", "she is", data)
    data = re.sub("that's|that’s", "that is", data)
    data = re.sub("what's|what’s", "what is", data)
    data = re.sub("where's|where’s", "where is", data)
    data = re.sub("how's|how’s", "how is", data)
    data = re.sub("\'ve|i’ve", " have", data)
    data = re.sub("\'ll|\’ll", " will", data)
    data = re.sub("\'re|\’re", " are", data)
    data = re.sub("\'d|\’d", " would", data)
    data = re.sub("n't|n’t", " not", data)
    data = re.sub("won't|won’t", "will not", data)
    data = re.sub("can't|can’t", "can not", data)
    data=re.sub('http\S+','',data)
    data=re.sub('www\S+','',data)
    data=re.sub('[-/]',' ',data)
    data=re.sub('\w+\d','',data)
    data=re.sub('[^A-Za-z]+', ' ', data)
    data=re.sub("[ ]{2,}", " ", data)

    return data
```

Figure 3.3: Cleaning Function

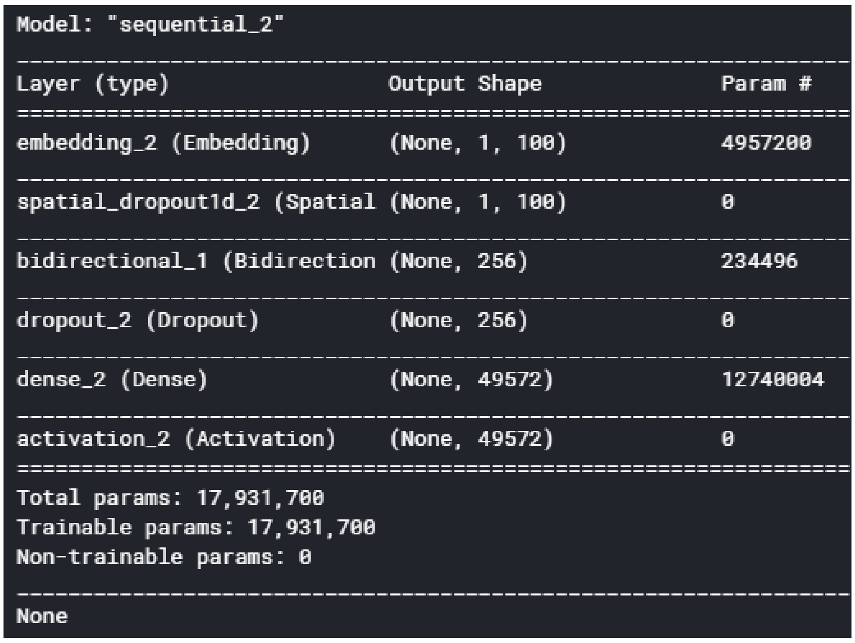
3.6 Results and Discussion

We train and test our models using the three data sets, here we discuss the obtained results:

3.6.1 First dataset

We train on it an RNN and bidirectional LSTM models, where when increasing the embedding dimension from 50 to 100 the bidirectional LSTM model gives the best results when using 1% and 2% of data from each file. Here we present only the bidirectional LSTM model, because of its effectiveness.

- In case of one percent: the model was trained on 591545 samples, and was validated on 197180 samples and evaluated on 197180 samples, where the used loss function was categorical_crossentropy and the optimization was Adam as shown in Figure 3.4.

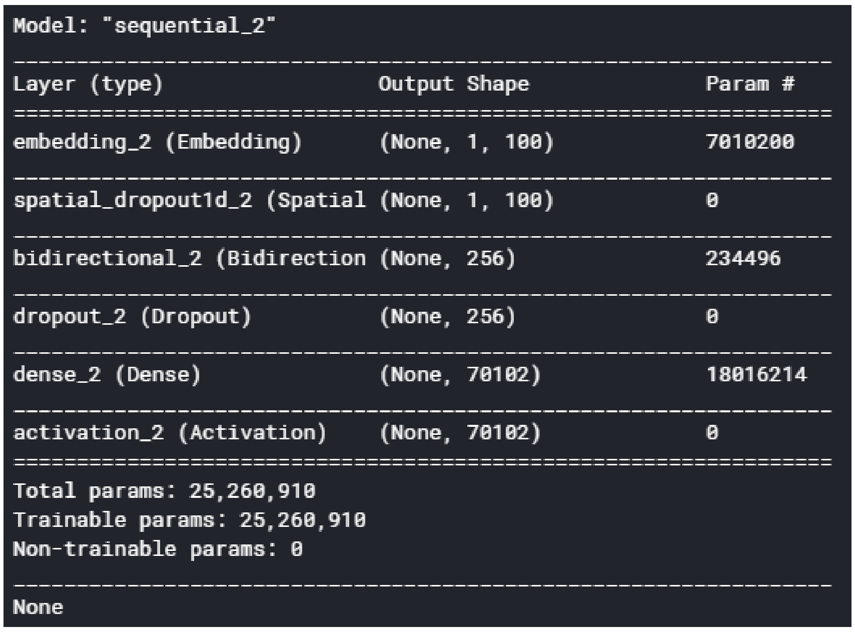


```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1, 100)            4957200
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 1, 100)            0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 256)               234496
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 49572)             12740004
_________________________________________________________________
activation_2 (Activation)    (None, 49572)             0
=================================================================
Total params: 17,931,700
Trainable params: 17,931,700
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3.4: Bideractional LSTM model (A).

The model has been trained for 12 epochs. We notice that the best result was achieved when using an embedding dimension equal to 100 where the accuracy reached 12.12%.

- In case of two percent: the model was trained on 1182083 samples, and validated on 394027 samples and evaluated on 394027 samples as shown in Figure 3.5.



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 1, 100)            7010200
_________________________________________________________________
spatial_dropout1d_2 (Spatial (None, 1, 100)            0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 256)               234496
_________________________________________________________________
dropout_2 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 70102)             18016214
_________________________________________________________________
activation_2 (Activation)    (None, 70102)             0
=================================================================
Total params: 25,260,910
Trainable params: 25,260,910
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3.5: Bideractional LTSM model (B).

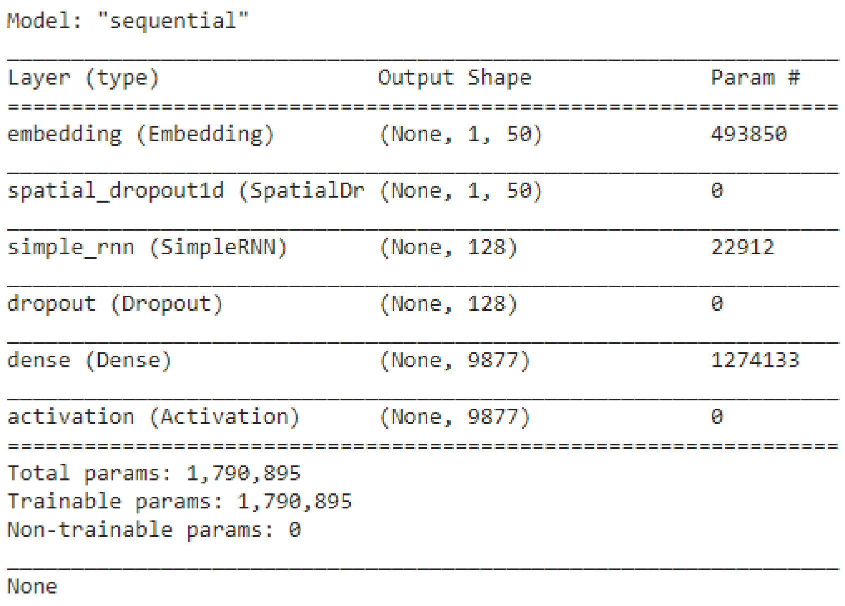
When using two percent of the data and training the model for 12 epochs, we expected to get more accuracy since we increased the amount of data, but the results were less than when using one percent of the data where it was 11.48%.

Due to the consumption of the TCN model for more RAM, we could not use it with this data.

3.6.2 Second dataset

With this data set we use:

- SimpleRNN model: we trained this model on 80835 samples and validated it on 8083 samples and evaluated it on 20208 samples, the used loss and optimization functions were the same as the first data as shown in Figure 3.6.

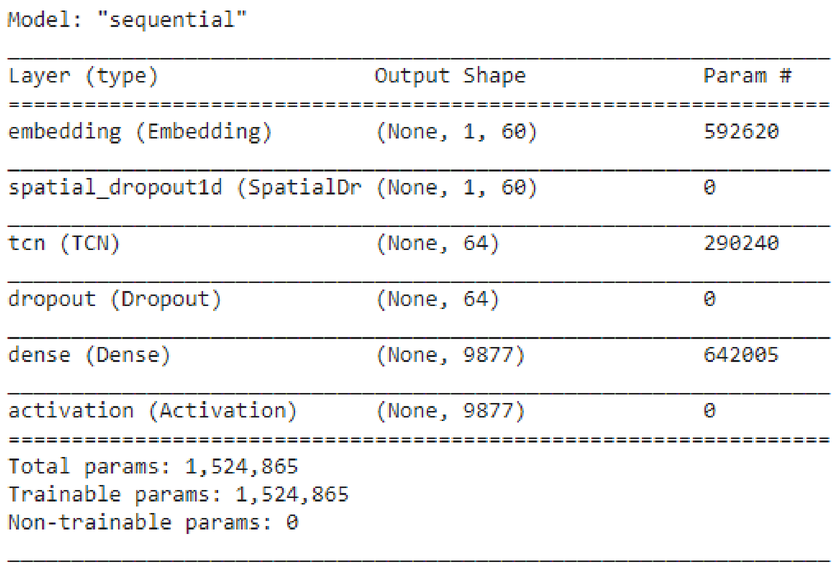


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1, 50)             493850
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 1, 50)             0
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 128)               22912
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 9877)              1274133
_________________________________________________________________
activation (Activation)      (None, 9877)              0
=================================================================
Total params: 1,790,895
Trainable params: 1,790,895
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3.6: SimpleRNN model (C).

This model has been trained for 50 epochs. The best results were achieved using an embedding dimension of 50 where the accuracy reached 20.63%.

- TCN model: the TCN model was trained as same as the precedent model, as shown in Figure 3.7



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 1, 60)             592620
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 1, 60)             0
_________________________________________________________________
tcn (TCN)                    (None, 64)                290240
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 9877)              642005
_________________________________________________________________
activation (Activation)      (None, 9877)              0
=================================================================
Total params: 1,524,865
Trainable params: 1,524,865
Non-trainable params: 0
_________________________________________________________________
```

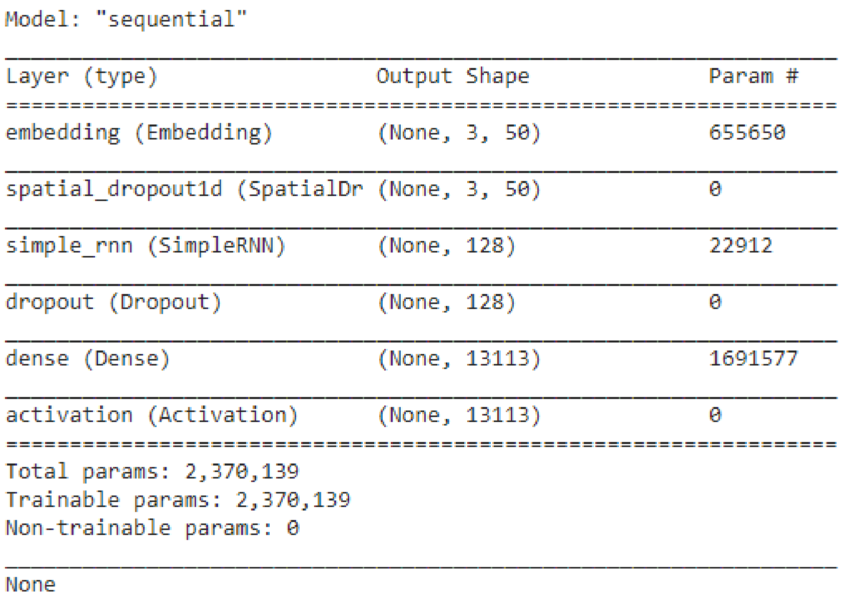
Figure 3.7: TCN Model (D).

This model was also trained on 50 epochs. The best result was achieved using an embedding dimension of 60 where the accuracy reached 21.25 %.

3.6.3 Third dataset

We use two models:

- SimpleRNN model: we trained this model on 80440 samples and validated it on 8044 samples and evaluated it on 20110 samples, the used loss and optimization functions were the same as the others, as shown in Figure 3.8.

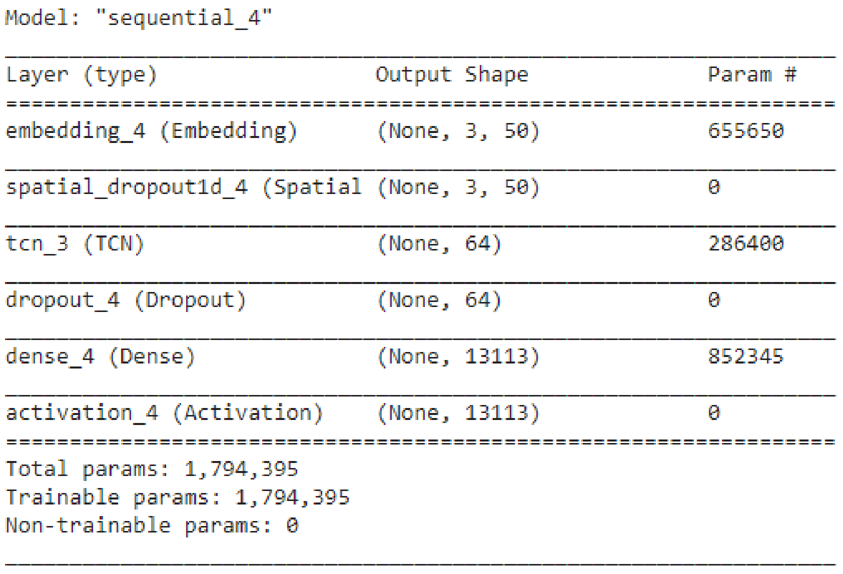


```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 3, 50)             655650
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 3, 50)             0
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 128)               22912
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense (Dense)                (None, 13113)             1691577
_________________________________________________________________
activation (Activation)      (None, 13113)             0
=================================================================
Total params: 2,370,139
Trainable params: 2,370,139
Non-trainable params: 0
_________________________________________________________________
None
```

Figure 3.8: SimpleRNN model (E).

The model has been trained for 100 epochs. The best results were achieved using an embedding dimension of 50 where the accuracy reached 71.51%.

- TCN model: the TCN model was trained as same as the precedent model, as shown in Figure 3.9



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 3, 50)             655650
_________________________________________________________________
spatial_dropout1d_4 (Spatial (None, 3, 50)             0
_________________________________________________________________
tcn_3 (TCN)                  (None, 64)                286400
_________________________________________________________________
dropout_4 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 13113)             852345
_________________________________________________________________
activation_4 (Activation)    (None, 13113)             0
=================================================================
Total params: 1,794,395
Trainable params: 1,794,395
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.9: TCN model (F).

This model also has been trained for 100 epochs. The best results were achieved using an embedding dimension of 50, where the accuracy reached 65.20%.

3.6.4 Discussion

After many trials, we draw the following results shown in table 3.1. The characters A,B,..F reference the previous models.

Model	Training		Validation		epochs	batch	Test		batch
/	Accuracy	Loss	Accuracy	Loss	/	/	Accuracy	Loss	/
A	12.05 %	5.96	8.30 %	7.34	12	500	8.32 %	7.38	128
B	11.45 %	6.01	8.79 %	7.21	12	500	8.81 %	7.19	128
C	20.63 %	4.09	12.51 %	7.78	50	150	12.10 %	7.89	80
D	21.25 %	3.93	12.59 %	11.75	50	50	12.07 %	11.81	10
E	71.51 %	1.14	9.14 %	10.98	100	100	8.32 %	11.41	80
F	65.20 %	1.38	8.22 %	18.33	100	100	7.18 %	18.59	10

Table 3.1: Results of all trained models.

From the obtained results we can conclude that:

- The idea of using multi-domain data did not give promising results, as we think this is due to our use of a small amount of data. Likewise, the use of the previous word only to predict the next word causes confusion to the model due to the multiplicity of options for the next word, which reduces its accuracy. Using the history to predict the next word reduces confusion and significantly improves the accuracy of the model due to the availability of sufficient information to predict the next word, and makes up for lack of data

- The accuracy of the models is affected by many parameters, such as the number of epochs, the batch size and the embedding dimension, where a greater value of this parameters gives better result, but with limits where a very large value may give poor results and slow the learning processes, we do not have any rule that tell us what are the values we should choose, so we should try until we reach the appropriate values.

- The accuracy does not depend only on the model and parameters but it depends heavily on the used data set, as the first data was very noisy compared to the second and the third, so we conclude that cleaner and less noisy data gives better results than a big and noised data. Making in consideration the linguistic universe, our used dataset was so small and not sufficient to represent the whole language. The best solution is a big cleaned data.

We faced the problem of over-fitting where in all the models the training loss keeps decreasing and validation loss keeps increasing. We try many regularization techniques such as L1, L2, L1_L2, but they make the results much worst instead of improving them. When we use SGD optimization function with gradient clipping technique like above:

$$SGD(lr = 0.01, momentum = 0.9, clipvalue = 0.5) \tag{3.2}$$

It prevents the over-fitting in different RNN models for the first and second data sets but not in TCN models, it also makes the accuracy stuck at 5% where it stops the model from learning for the first and second data sets, and for the third one it makes the accuracy of the model decreased as it decreased to 28.55% and did not completely solve the problem of over-fitting, but only reduced it, so we gave up the idea of using it.

- After many experiments and according to what we learned about the problem of NWP we think that the over-fitting difficult if not to say impossible to solve it in this task because no matter how much we try, we cannot predict all word formations, and we cannot guarantee how words are distributed in the data, it needed a really huge amount of data with knowledge about all possible words formations and a very high computing capacity computer which we can not achieve.

On the other hand, we cannot judge a model with acceptable or good accuracy as not good or very good since the last judgment depends upon application. When a model is used in a prediction system, it is highly dependent on the user, where we cannot guarantee or know what the user intends to write. We mention that there are several factors that play a role in evaluating prediction systems, which we mentioned earlier in chapter two.

We also mention that the advantage of the RNN is that when giving it even one word, it gives predictions, even though it has trained to take three words to predict the next word, unlike n-gram, which requires the use of n previous word and does not work without it.

3.7 Conclusion

In this chapter, we presented the best models that we found, where the highest accuracy was 71.51% obtained from the SimpleRNN model trained on the third dataset using the past three words to predict the next word, followed by the TCN model for the same dataset with an accuracy of 65.20% where the difference was not that much big, indicating the strength and effectiveness of the TCN hybrid neural network. The achieved results can be improved with many ways such as increasing data volume which we cannot do due to the low efficiency of our devices and the resource limitations imposed by Google Collab and Kaggle.

Conclusion

In our work we investigate the field of machine learning where we focus on neural networks and deep learning with application from the field of natural language processing. We dive into language modeling to build deep learning models that solve the problem of predicting the next word which is one of the most important goals of language modeling due to its multiple uses especially in our time.

We develop several deep learning models to predict the next word using the RNN as well as the TCN structure using three databases: the first is Coursera Swiftkey data, the second is the book Writings of Nietzsche: Volume 1 by Friedrich Nietzsche and the third is the news category from the Brown corpus in nltk library. Results are satisfactory and would have been better had it not been for the restrictions imposed by Google Colab and Kaggle, the maximum accuracy we got is 71.51%, which was recorded by the SimpleRNN structure trained on the third data set by using the past three words to predict the next word followed by the TCN model for the same data with an accuracy of 65.20%. We can say that TCN architecture competes with RNN architecture in language modeling.

In order to obtain better results, we suggest using a large database or several databases to cover the largest possible number of vocabulary, as well as changing parameter values or using a larger network. Deep neural networks can also be used with other algorithms to give better results.

We also suggest to work with specific domain (sport, news,...) datasets, this can reduce the vocabulary size and gives more focused results.

The study of some recent deep learning platforms for mobile devices can also be proposed as an extension.

Due to the limited resources, we do not investigate NWP for Arabic datasets. It would be very interesting to move in this direction.

Bibliography

Ayon Dey. Machine learning algorithms: a review. *International Journal of Computer Science and Information Technologies*, 7(3):1174–1179, 2016.

Richard Smith, (author.) Lynch, David, and (author.) Knight, Bruce Allen. *Learning management : transitioning teachers for national and international change*. Frenchs Forest, New South Wales : Pearson Education Australia, 2007. ISBN 978-0-7339-8959-9. "A Pearson Education Australia sprintprint"–top of title page.

El Rharras Abdessamad, Sami el Moukhlis, Saadane Rachid, Mohammed Wahbi, and A. Hamdoun. Fpga-based fully parallel pca-ann for spectrum sensing. *Computer and Information Science*, 8, 12 2015. doi: 10.5539/cis.v8n1p108.

Charu C. Aggarwal. *Neural Networks and Deep Learning - A Textbook*. Springer, 2018a. ISBN 978-3-319-94462-3. doi: 10.1007/978-3-319-94463-0. URL `https://doi.org/10.1007/978-3-319-94463-0`.

Lyes Khacef, Nassim Abderrahmane, and Benoît Miramond. Confronting machine-learning with neuroscience for neuromorphic architectures design. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

Meriem Bahi and Mohamed Batouche. Deep learning for ligand-based virtual screening in drug discovery. In *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*, pages 1–5. IEEE, 2018.

Akshay Kulkarni and Adarsha Shivananda. *Natural language processing recipes*. Springer, 2019.

Mário P Véstias. A survey of convolutional neural networks on edge with reconfigurable computing. *Algorithms*, 12(8):154, 2019.

Wartini Ng, Budiman Minasny, Maryam Montazerolghaem, Jose Padarian, Richard Ferguson, Scarlett Bailey, and Alex B McBratney. Convolutional neural network for simultaneous prediction of several soil properties using visible/near-infrared, mid-infrared, and their combined spectra. *Geoderma*, 352:251–267, 2019.

Weijiang Feng, Naiyang Guan, Yuan Li, Xiang Zhang, and Zhigang Luo. Audio visual speech recognition with multimodal recurrent neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 681–688. IEEE, 2017.

Vishnu Subramanian. *Deep Learning with PyTorch: A practical approach to building neural network models using PyTorch*. Packt Publishing Ltd, 2018.

Xu Tao and Yun Zhou. Fall prediction based on biomechanics equilibrium using kinect. *International Journal of Distributed Sensor Networks*, 13:155014771770325, 04 2017. doi: 10.1177/1550147717703257.

Mohammed Jabreel and Antonio Moreno. A deep learning-based approach for multi-label emotion classification in tweets. *Applied Sciences*, 9:1123, 03 2019. doi: 10.3390/app9061123.

Arindam Chaudhuri. Experimental setup: Visual and text sentiment analysis through hierarchical deep learning networks. In *Visual and Text Sentiment Analysis through Hierarchical Deep Learning Networks*, pages 25–49. Springer, 2019.

Daniel C Cavalieri, Sira E Palazuelos-Cagigas, Teodiano F Bastos-Filho, and Mário Sarcinelli-Filho. Combination of language models for word prediction: an exponential approach. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(9):1481–1494, 2016.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. An embedded deep learning based word prediction. *arXiv preprint arXiv:1707.01662*, 2017.

Youssef Oualil, Mittul Singh, Clayton Greenberg, and Dietrich Klakow. Long-short range context neural networks for language modeling. *arXiv preprint arXiv:1708.06555*, 2017.

Yann Lecun. Les enjeux de la recherche en intelligence artificielle. *Interstices*, February 2016. URL `https://hal.inria.fr/hal-01350469`.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach, Third International Edition.* Pearson Education, 2010. ISBN 978-0-13-207148-2. URL `http://vig.pearsoned.com/store/product/1,1207,store-12521_isbn-0136042597,00.html`.

John Haugeland. *Artificial intelligence: The very idea.* MIT press, 1989.

Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley Longman Publishing Co., Inc., USA, 1985. ISBN 0201119455.

Ray Kurzweil, Robert Richter, Ray Kurzweil, and Martin L Schneider. *The age of intelligent machines*, volume 579. MIT press Cambridge, 1990.

David Poole, Alan K. Mackworth, and Randy Goebel. *Computational intelligence - a logical approach.* Oxford University Press, 1998. ISBN 978-0-19-510270-3.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Tom M. Mitchell. *Machine learning, International Edition.* McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. ISBN 978-0-07-042807-2. URL `https://www.worldcat.org/oclc/61321007`.

Taiwo Oladipupo Ayodele. Types of machine learning algorithms. *New advances in machine learning*, pages 19–48, 2010.

Pierre Lison. An introduction to machine learning. *Language Technology Group: Edinburgh, UK*, 2015.

Ke-Lin Du and Madisetti NS Swamy. *Neural networks and statistical learning.* Springer Science & Business Media, 2013.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Samantha Hayman. The mcculloch-pitts model. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, volume 6, pages 4438–4439. IEEE, 1999.

Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms.* " O'Reilly Media, Inc.", 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Jinming Zou, Yi Han, and Sung-Sau So. *Overview of Artificial Neural Networks*, pages 14–22. Humana Press, Totowa, NJ, 2009. ISBN 978-1-60327-101-1. doi: 10.1007/978-1-60327-101-1_2. URL https://doi.org/10.1007/978-1-60327-101-1_2.

Steven Walczak and Narciso Cerpa. Artificial neural networks. In Robert A. Meyers, editor, *Encyclopedia of Physical Science and Technology (Third Edition)*, pages 631 – 645. Academic Press, New York, third edition edition, 2003. ISBN 978-0-12-227410-7. doi: https://doi.org/10.1016/B0-12-227410-5/00837-1. URL http://www.sciencedirect.com/science/article/pii/B0122274105008371.

Maad M Mijwel. Artificial neural networks advantages and disadvantages. *Retrieved from LinkedIn: https://www. linkedin. com/pulse/artificial-neuralnet works-advantages-disadvantages-maad-m-mijwel*, 2018.

Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and SS Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):1–36, 2018.

Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

John A Bullinaria. Recurrent neural networks. *Neural Computation: Lecture*, 12, 2013.

Roger Grosse. Lecture 15: Exploding and vanishing gradients. *University of Toronto Computer Science*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Alex Graves, Marcus Liwicki, Santiago Fernández, Roman Bertolami, Horst Bunke, and Jürgen Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5): 855–868, 2008.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors, 2012.

Yann LeCun, Leon Bottou, G Orr, and Klaus-Robert Muller. Efficient backprop. *Neural Networks: Tricks of the Trade. New York: Springer*, 1998.

Andrej Andrejewitsch Markov. An example of statistical investigation of the text eugene onegin concerning the connection of samples in chains. classical text in translation. *Lecture at the physical-mathematical faculty, Royal Academy of Sciences, St. Petersburg*, 23:591–600, 1913.

Claude E Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951.

Noam Chomsky. Three models for the description of language. *IRE Transactions on information theory*, 2(3):113–124, 1956.

Gerald R Gendron and GR Gendron. Natural language processing: A model to predict a sequence of words. *MODSIM World*, 2015:1, 2015.

Tamas E Doszkocs. Natural language processing in information retrieval. *Journal of the American Society for Information Science*, 37(4):191–196, 1986.

Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

Jalaj Thanaki. *Python natural language processing.* Packt Publishing Ltd, 2017.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Daniel Bastos Pereira and Ivandré Paraboni. A language modelling tool for statistical nlp. In *Anais do V Workshop em Tecnologia da Informação e da Linguagem Humana–TIL*, pages 1679–1688, 2007.

Stanley F Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394, 1999.

Imed Zitouni. *Natural language processing of semitic languages.* Springer, 2014.

Piotr Kłosowski. Deep learning for natural language processing and language modelling. In *2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pages 223–228. IEEE, 2018.

Tonio Wandmacher. *Adaptive word prediction and its application in an assistive communication system.* PhD thesis, FRENCH-RABELAIS UNIVERSITY OF TOURS, 2009.

Ben Allison, David Guthrie, and Louise Guthrie. Another look at the data sparsity problem. In *International Conference on Text, Speech and Dialogue*, pages 327–334. Springer, 2006.

Martin Christian Körner and Steffen Staab. *Implementation of Modified Kneser-Ney Smoothing on Top of Generalized Language Models for Next Word Prediction*. Universität Koblenz-Landau, Campus Koblenz, 2013. URL `https://books.google.dz/books?id=w72QnQAACAAJ`.

Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE transactions on acoustics, speech, and signal processing*, 35(3):400–401, 1987.

Masood Ghayoomi and Saeedeh Momtazi. An overview on the existing language models for prediction systems as writing assistant tools. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 5083–5087. IEEE, 2009.

Pertti Alvar Väyrynen, Kai Noponen, and Tapio Seppänen. Analysing performance in a word prediction system with multiple prediction methods. *Computer Speech & Language*, 21(3):479–491, 2007.

Nestor Garay-Vitoria and Julio Abascal. Text prediction systems: a survey. *Universal Access in the Information Society*, 4(3):188–203, 2006.

Carlo Aliprandi, Nicola Carmignani, Nedjma Deha, Paolo Mancarella, and Michele Rubino. Advances in nlp applied to word prediction. *University of Pisa, Italy February*, 2008.

Riya Makkar, Manjinder Kaur, and Dharam Veer Sharma. Word prediction systems: a survey. *Adv Comput Sci Inform Technol*, 2(2):177–80, 2015.

Andrew Swiffin, John Arnott, J Adrian Pickering, and Alan Newell. Adaptive and predictive techniques in a communication prosthesis. *Augmentative and Alternative Communication*, 3(4):181–191, 1987.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009. ISBN 0131873210.

P Majumder, M Mitra, and BB Chaudhuri. N-gram: a language independent approach to ir and nlp. In *International conference on universal knowledge and language*, 2002.

Masood Ghayoomi and Seyyed Mostafa Assi. Word prediction in a running text: A statistical language modeling for the persian language. In *Proceedings of the Australasian Language Technology Workshop 2005*, pages 57–63, 2005.

Md Haque, Md Habib, Md Rahman, et al. Automated word prediction in bangla language using stochastic language models. *arXiv preprint arXiv:1602.07803*, 2016.

Manash Pratim Bhuyan and Shikhar Kumar Sarma. Automatic formation, termination & correction of assamese word using predictive & syntactic nlp. In *2018 3rd International Conference on Communication and Electronics Systems (ICCES)*, pages 544–548. IEEE, 2018.

Gregory W Lesher, Bryan J Moulton, D Jeffery Higginbotham, et al. Effects of ngram order and training text size on word prediction. In *Proceedings of the RESNA '99 Annual Conference*, pages 52–54. Citeseer, 1999.

Jianfeng Gao, Joshua Goodman, Mingjing Li, and Kai-Fu Lee. Toward a unified approach to statistical language modeling for chinese. *ACM Transactions on Asian Language Information Processing (TALIP)*, 1(1):3–33, 2002.

Keith Trnka and Kathleen F McCoy. Corpus studies in word prediction. In *Proceedings of the 9th international ACM SIGACCESS conference on Computers and accessibility*, pages 195–202, 2007.

Rene Pickhardt, Thomas Gottron, Martin Körner, Paul Georg Wagner, Till Speicher, and Steffen Staab. A generalized language model as the combination of skipped n-grams and modified kneser-ney smoothing. *arXiv preprint arXiv:1404.3377*, 2014.

Charu C Aggarwal. *Machine learning for text.* Springer, 2018b.

Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

Sean R Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

Lawrence Rabiner and B Juang. An introduction to hidden markov models. *ieee assp magazine*, 3(1):4–16, 1986.

Monica Jordan, Guilherme N Nogueira Neto, Alceu Brito, and Percy Nohama. Virtual keyboard with the prediction of words for children with cerebral palsy. *Computer Methods and Programs in Biomedicine*, page 105402, 2020.

Nitin Indurkhya and Fred J Damerau. *Handbook of natural language processing.* Chapman and Hall/CRC, 2010.

Afsaneh Fazly. *The use of syntax in word completion utilities.* Master's thesis, Department of Computer Science, University of Toronto, 2002.

Afsaneh Fazly and Graeme Hirst. Testing the efficacy of part-of-speech information in word completion. In *Proceedings of the 2003 EACL Workshop on Language Modeling for Text Entry Methods*, 2003.

Carmelo Spiccia, Agnese Augello, Giovanni Pilato, and Giorgio Vassallo. A word prediction methodology for automatic sentence completion. In *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*, pages 240–243. IEEE, 2015.

Sheri Hunnicutt. Lexical prediction for a text-to-speech system. *Communication and Handicap: Aspects of Psychological Compensation and Technical Aids.: Elsevier Science Publishers*, 1986.

Corinne Morris, Alan Newell, Lynda Booth, Ian Ricketts, and John Arnott. Syntax pal: A system to improve the written syntax of language-impaired users. *Assistive Technology*, 4(2):51–59, 1992.

Kathleen McCoy and Patrick Demasco. Some applications of natural language processing to the field of augmentative and alternative communication. In *Proceedings of the IJCAI-95 Workshop on Developing AI Application for People with Disabilities.* Citeseer, 1995.

Matthew EJ Wood and Eric Lewis. Windmill-the use of a parsing algorithm to produce predictions for disabled persons. *PROCEEDINGS-INSTITUTE OF ACOUSTICS*, 18: 315–322, 1996.

James Allen. *Natural Language Understanding*. Benjamin Cummings, 1987.

Nestor Garay-Vitoria and Julio González-Abascal. Application of artificial intelligence methods in a word-prediction aid. In *International Conference on Computers for Handicapped Persons*, pages 363–370. Springer, 1994.

Nestor Garay-Vitoria and Julio Gonzalez-Abascal. Intelligent word-prediction to enhance text input rate (a syntactic analysis-based word-prediction aid for people with severe motor and speech disability). In *Proceedings of the 2nd international conference on Intelligent user interfaces*, pages 241–244, 1997.

Yoshimasa Tsuruoka and Jun'ichi Tsujii. Chunk parsing revisited. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 133–140. Association for Computational Linguistics, 2005.

Igor Schadle. Sibyl: Aac system using nlp techniques. In *International Conference on Computers for Handicapped Persons*, pages 1009–1015. Springer, 2004.

Nestor Garay and J Abascal. Using statistical and syntactic information in word prediction for input speed enhancement. *Information Systems Design and Hypermedia*, pages 223–230, 1994.

Denis Maurel and Brigitte Le Pévédic. The syntactic prediction with token automata: application to handias system. *Theoretical computer science*, 267(1-2):121–129, 2001.

Joseph Gubbins and Andreas Vlachos. Dependency language models for sentence completion. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1405–1410, 2013.

Sheri Hunnicutt. Using syntactic and semantic information in a word prediction aid. In *First European Conference on Speech Communication and Technology*, 1989.

Masood Ghayoomi and Ehsan Daroodi. A pos-based word prediction system for the persian language. In *International Conference on Natural Language Processing*, pages 138–147. Springer, 2008.

Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

Geoffrey Zweig and Christopher JC Burges. The microsoft research sentence completion challenge. *Microsoft Research Technical Report MSR-TR-2011–129*, 2011.

Marco Baroni, Johannes Matiasek, and Harald Trost. Predicting the components of german nominal compounds. In *ECAI*, pages 470–474, 2002.

Johannes Matiasek, Marco Baroni, and Harald Trost. Fasty—a multi-lingual approach to text prediction. In *International Conference on Computers for Handicapped Persons*, pages 243–250. Springer, 2002.

Sheri Hunnicutt and Johan Carlberger. Improving word prediction using markov models and heuristic methods. *Augmentative and Alternative Communication*, 17(4):255–264, 2001.

Yair Even-Zohar and Dan Roth. A classification approach to word prediction. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 124–131. Association for Computational Linguistics, 2000.

Bharat Kapse and Urmila Shrawankar. Word prediction using b+ tree for braille users. In *2013 Students Conference on Engineering and Systems (SCES)*, pages 1–5. IEEE, 2013.

Ned Horning. Introduction to decision trees and random forests. *Am. Mus. Nat. Hist*, 2: 1–27, 2013.

Lalit R Bahl, Peter F Brown, Peter V de Souza, and Robert L Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(7):1001–1008, 1989.

Walter Daelemans, Antal Van Den Bosch, and Ton Weijters. Igtree: using trees for compression and classification in lazy learning algorithms. In *Lazy learning*, pages 407–423. Springer, 1997.

Antal Van Den Bosch. Scalable classification-based word prediction and confusible correction. *Traitement Automatique des Langues*, 46(2):39–63, 2006.

Alexey Ya. Chervonenkis. *Early History of Support Vector Machines*, pages 13–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-41136-6. doi: 10.1007/978-3-642-41136-6_3. URL `https://doi.org/10.1007/978-3-642-41136-6_3`.

Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

Chun-Fu Lin and Sheng-De Wang. Fuzzy support vector machines. *IEEE transactions on neural networks*, 13(2):464–471, 2002.

Hisham Al-Mubaid. A learning-classification based approach for word prediction. *Int. Arab J. Inf. Technol.*, 4(3):264–271, 2007.

Hisham Al-Mubaid and Ping Chen. Application of word prediction and disambiguation to improve text entry for people with physical disabilities (assistive technology). *IJSHC*, 1(1):10–27, 2008.

Henrique X Goulart, Mauro DL Tosi, Daniel Soares Gonçalves, Rodrigo F Maia, and Guilherme A Wachs-Lopes. Hybrid model for word prediction using naive bayes and latent information. *arXiv preprint arXiv:1803.00985*, 2018.

Luigi Troiano, Cosimo Birtolo, and Roberto Armenise. Modeling and predicting the user next input by bayesian reasoning. *Soft Computing*, 21(6):1583–1600, 2017.

Oliver Kramer. K-nearest neighbors. In *Dimensionality reduction with unsupervised nearest neighbors*, pages 13–23. Springer, 2013.

Darren Kuo. On word prediction methods. *Technical report, Technical report, EECS Department*, 2011.

Kun Jing, Jungang Xu, and Ben He. A survey on neural network language models. *arXiv preprint arXiv:1906.03591*, 2019.

Wei Xu and Alex Rudnicky. Can artificial neural networks learn language models? In *Sixth international conference on spoken language processing*, 2000.

Andriy Mnih and Yee Whye Teh. A fast and simple algorithm for training neural probabilistic language models. *arXiv preprint arXiv:1206.6426*, 2012.

Daniel C Cavalieri, Sira E Palazuelos-Cagigas, Teodiano F Bastos-Filho, and Mário Sarcinelli-Filho. Evaluation of machine learning approaches to portuguese part-of-speech prediction. In *Computational Processing of the Portuguese Language, 9th International Conference, Proceedings (PROPOR 2010), Porto Alegre, Brasil*, pages 27–30, 2010.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

Artem M Grachev, Dmitry I Ignatov, and Andrey V Savchenko. Compression of recurrent neural networks for efficient language modeling. *Applied Soft Computing*, 79:354–362, 2019.

Partha Pratim Barman and Abhijit Boruah. A rnn based approach for next word prediction in assamese phonetic transcription. *Procedia computer science*, 143:117–123, 2018.

Brendon Boldt. Using lstms to model the java programming language. In *International Conference on Artificial Neural Networks*, pages 268–275. Springer, 2017.

Mingxuan Wang, Zhengdong Lu, Hang Li, Wenbin Jiang, and Qun Liu. *gen* cnn: A convolutional architecture for word sequence prediction. *arXiv preprint arXiv:1503.05034*, 2015.

Ngoc-Quan Pham, German Kruszewski, and Gemma Boleda. Convolutional neural network language models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1153–1162, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1123. URL `https://www.aclweb.org/anthology/D16-1123`.

Adnan Souri, Mohamed Alachhab, Badr Eddine Elmohajir, and Abdelali Zbakh. Convolutional neural networks in predicting missing text in arabic. *International Journal of Advanced Computer Science and Applications*, 10(6), 2019. doi: 10.14569/IJACSA.2019.0100668. URL `http://dx.doi.org/10.14569/IJACSA.2019.0100668`.

Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Colin Lea, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *European Conference on Computer Vision*, pages 47–54. Springer, 2016.