UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER UNIVERSITARIO DE INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Prototype and Evaluation of a Conversational Bot based on Recurrent Neural Networks using TensorFlow

Diego San Cristóbal Bastardo
2018

TRABAJO DE FIN DE MASTER

Título:	Prototipo y Evaluación de Bot Conversacional basado en Redes Neuronales Recurrentes usando TensorFlow
Título (inglés):	Prototype and Evaluation of a Conversational Bot based on Recurrent Neural Networks using TensorFlow
Autor:	Diego San Cristóbal Bastardo
Tutor:	Óscar Araque Iborra
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

Prototype and Evaluation of a Conversational Bot based on Recurrent Neural Networks using TensorFlow

Febrero 2018

Resumen

La inteligencia artificial es una disciplina científica de carácter multidisciplinar que tiene como objetivo la comprensión de la inteligencia así como la construcción de entidades inteligentes. Debido a dicho caracter multidisciplinar, la inteligencia artificial está experimentando un gran auge pues permite abordar multitud de problemas distintos obteniendo buenos resultados.

Concretamente, uno de los campos de la inteligencia artifcial que más desarrollo está experimentando es el del aprendizaje automático o machine learning. Este subcampo se encarga del desarrollo de programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos. Dentro de esta rama destacan especialmente las redes neuronales que constituyen un paradigma inspirado en las neuronas del sistema nervioso de los animales.

Por otro lado, uno de los grandes objetivos de la inteligencia artificial es dotar a las máquinas de la capacidad de comprender y generar lenguaje natural, permitiendo la comunicación entre humanos y máquinas. El Natural Language Processing (NLP) es el campo de la ciencia encargado del estudio de dicho proceso. Por tanto, el propósito de este proyecto es el de analizar el problema de la comunicación humano-máquina mediante lenguaje natural, así como el desarrollo de una solución utilizando técnicas del campo de la inteligencia artificial.

Para la consecución de tal objetivo, se ha desarrollado un prototipo basado en redes neuronales. Para ello, se han analizado diferentes herramientas y tecnologías que brindan la capacidad de realizar este tipo de desarrollos. Además, se han estudiado diferentes fuentes de información para la formación del conjunto de datos necesario para que el prototipo pudiese desempeñar el proceso de aprendizaje.

Por último, se han realizado diferentes experimentos con diferentes conjuntos de datos y se han analizado los resultados obtenidos.

Palabras clave: Inteligencia artificial, aprendizaje automático, redes neuronales, deep learning, algoritmos de aprendizaje.

Abstract

Artificial intelligence is a multidisciplinary scientific discipline that aims to understand intelligence as well as the construction of intelligent entities. Due to this multidisciplinary character, artificial intelligence is experiencing a great boom because it allows to address a multitude of different problems obtaining good results.

Specifically, one of the fields of artificial intelligence that is suffering more development is machine learning. This sub-field is responsible for the development of programs capable of generalizing behaviors from information provided in the form of examples. Within this branch, especially stand out neural networks that constitute a paradigm inspired by the neurons of the animals nervous system.

On the other hand, one of the main objectives of artificial intelligence is to equip machines with the ability to understand and generate natural language, allowing communication between humans and machines in an easier way. The Natural Language Processing (NLP) is the field of science responsible for the study of this process. Therefore, the purpose of this project is to analyze the problem of human-machine communication through natural language, as well as the development of a solution using techniques from the field of artificial intelligence.

To achieve this goal, a prototype based on neural networks has been developed. For this purpose, different tools and technologies that provide the capacity to carry out this type of development have been analyzed. In addition, different sources of information have been studied for structuring the data set necessary for the prototype to perform the learning process.

Finally, different experiments have been carried out with different data sets and the results obtained have been analyzed.

Keywords: Artificial intelligence, machine learning, neural networks, deep learning, learning algorithms.

Agradecimientos

A mi tutor Óscar Araque, por su inestimable ayudada con este proyecto y por haberme brindado la posibilidad de utilizar la infraestructura del departamento.
A mis padres, hermano y amigos por preocuparse y apoyarme siempre durante todo este largo camino.
A Laura, por siempre estar ahí para animarme en este último reto.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Artificial Intelligence is a scientific field which is very useful in many other scientific and non-scientific fields and is reaching more and more importance each day.

Historically, the first attempts to use logical means to produce knowledge, appeared during the thirteenth century, and an example of this is the theoretical basis that Ramón Llul developed and that served to lay the foundations of the Ars Magna Generails, a logical machine considered as one of the first approaches to artificial intelligence.

In 1847, George Boole develops and establishes the foundations of the propositional logic. Shortly after, in 1879, Gottlob Frege presents the first system of predicate logic. This caused a significant advance in the field of logic.

Later, in 1936, the mathematician Alan Turing published a study called "On computable numbers, with an application to the Entscheidungsproblem". This study involved the resolution of a challenge in the symbolic logic that consisted in finding a general algorithm that would decide whether a calculation formula of first order is or not a theorem. Turing, at the same time and independently of Alonzo Church, demonstrated that it was impossible to develop such an algorithm. This allowed him to revolutionize the world of mathematical logic by presenting what is known as the Turing machine, which was capable of simulating the logic of any computer algorithm.

In 1950, again Alan Turing, publishes the article “Computing machinery and intelligence” where the Turing test is presented, a test to determine if a machine is capable of exhibiting intelligent behavior in a similar way to humans.

However, the conception of the artificial intelligence (AI) term did not occur until 1956 during the Dartmouth conference organized and proposed by John McCarthy, Marvin L. Minsky, Nathaniel Rochester and Claude E. Shannon. Later, there is a great development in different areas of AI, from programming languages (such as PROLOG) to intelligent systems (such as SHRDLU), through models of knowledge representation (such as semantic networks developed in 1963).

During this period of exponential development of technologies and algorithms focused on the AI, one of the great milestones occurred in 1997, when Gari Kaspárov (at that time world chess champion) lost a chess game with a IBM’s computer called “Deep Blue” [43, 16].

Currently, AI has become a technique that uses the scientific method. This allows that, for a hypothesis to be accepted, it must be subjected to rigorous empirical experiments and the results must be statistically analyzed for their importance [12]. In addition, today it is possible to replicate the experiments thanks to the technological advances that have allowed the creation of repositories of data and repositories of code that allow their sharing.

1.1.1 Artificial intelligence nowadays

After its brief introduction about what Artificial Intelligence is and its historical context, we can affirm that AI has a multidisciplinary character that affects very specific aspects of mathematics, science, information sciences and Medicine. This multidisciplinary nature, together with advances in computational technology, that allows greater speed in the performance of calculations, is causing an increment in the use of the new AI techniques, since it allows to address problems and find solutions much more faster. AI is currently solving problems in the following areas [41]:

- Natural Language
  - Generation
  - Classification
  - Translation
- Perception
  - Computer Vision
- Reasoning

- Math
- Robotics
- Games
- Engineering
- Medicine
  - Diagnosis
  - Image Analysis

These are some of the fields in which AI is currently having a greater presence. However, it also begins to have a presence in disciplines that are more artistic such as design, fashion or even painting.

From those fields, there are one that is reaching more importance each day which is generation and understanding of natural language. Providing machines with the ability to understand and generate information in natural language is currently a main objective for the field of machine learning. It will produce a mass approach of artificial intelligence to users.

1.2 Project goals

The main objectives of this project are the following:

- Study the use of the artificial intelligence, specifically the use of neural networks, as a solution to common problems and , especially, to solve the problem of generating conversations using natural language .
- Selection and study of a specific machine learning algorithm able to solve mentioned problem.
- Development of a solution using current technologies and study the results of the mentioned solution.

1.3 Structure of this document

In this section is provided a brief overview of the chapters included in this document. The structure is as follows:

- ***Chapter 1 - Introduction:*** A general description of the field of study of the thesis is made with a justification of it. The objectives and methodologies followed are also indicated.
- ***Chapter 2 - Theoretical Background:***The theoretical foundations of artificial intelligence are established. The main algorithms of machine learning will be analyzed and it will be indicated which of them will be chosen for the development of the solution.
- ***Chapter 3 - Neural Networks:*** This section will give a general overview of neural networks and, in particular, will explain Recurrent Neural Networks deeply. LSTM units will also be widely explained and how they are combined with recurrent networks. Finally there will be an approach to existing technologies.
- ***Chapter 4 - The Problem:*** This chapter has the objective of presenting the problem that this project tries to solve, giving a general view of it and explaining general techniques and tools that can be used to solve it.
- ***Chapter 5 - The Solution:*** Here is presented the particular solution that this project gives to the previous problem. It explains the implementations used and presents the results obtained.
- ***Chapter 5 - Conclusions:*** It constitutes the final chapter were the conclusions are presented. Firstly, the difficulties encountered are commented. Then the conclusions obtained from the results and finally, there is section of possible future work.

CHAPTER 2

Theoretical Background

2.1 Artificial Intelligence

Artificial intelligence (from now on AI) is a scientific field which tries to understand what is the meaning of intelligence and to build entities that have it. This is a wide area of knowledge because it includes multiple disciplines.

If we search for the definition of intelligence in a dictionary we can find many meanings, but it this case only some of them have been selected:

1. The ability of understanding
2. The ability of resolving problems
3. Knowledge, understanding
4. Sense in which a proposition, saying or expression can be taken
5. Ability, skill and experience
6. Purely spiritual substance

As we can observe, there are varied definitions. In addition, if the word goes together with the adjective “artificial” it is even harder to give an accurate definition.

There are four approaches to define AI[43]:

- Systems that think as humans
- Systems that think rationally
- Systems that act like humans
- Systems that act rationally

These approaches has been defined by many authors as it follows below:

Systems that think as humans: In this kind of systems the important thing is how to achieve the reasoning and not the result of this reasoning. The proposal here, is to develop systems that reason in the same way as people. Cognitive science use this point of view.

"The exciting new effort to make computers think . . . machines with minds, in the full and literal sense" (Haugeland, 1985)

Systems that think rationally: In this case, the definition also focuses on reasoning, but here we start from the premise that there is a rational way of reasoning. Logic allows the formalization of reasoning and is used for this purpose.

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

Systems that act like humans: The model to follow for the evaluation of programs, corresponds to human behavior. The Turing Test (1950) also uses this point of view. The system Eliza, a conversational boy is an example of this.

"The art of creating machines that perform functions that require intelligence when performed by people" (Kurzweil, 1990)

Systems that act rationally: Again the objective is the results, but but in this case they are evaluated in a objectively way. For example, the goal of a program, in a game like chess, will be to win. To fulfill this objective it is indifferent the way to calculate the result

In addition to the definitions mentioned above we can find another classification of artificial intelligence according to the researching objectives in this field:

Weak artificial intelligence: It is considered that computers can simulate reasoning. The supporters of weak artificial intelligence believe that it will never be possible to build conscious computers, and that a program is a simulation of a cognitive process but not a cognitive process in itself.

Strong artificial intelligence: In this case, it is considered that a computer can have a mind and mental state, therefore, some day it will be possible to build one with all the capabilities of the human mind. This computer will be able to reason, imagine, etc.

2.2 Machine Learning

Machine learning is a branch of artificial intelligence focused on the field of computer science and it was in 1959 when Arthur Samuel defined it as the field of study that offers computers the possibility of learning without being explicitly programmed.

Machine learning is a central part in AI since the ability to learn is strongly related to intelligence. Currently this area of AI is playing an increasingly important role because it provides support to technology areas such as information extraction, information retrieval, adaptive user interfaces, intelligent agents, robotics, etc.[30]

There are different learning techniques inside the machine learning concept:

Descriptive Learning: Learning will be based on finding patterns among the data. As its objective is to find patterns, this type of learning has its main field of action in data mining.

Inductive Learning: In this case we have a type of predictive learning, as it is suggested by its name. The algorithms that use this type of learning are intended to make predictions of new cases, producing a function that analyzes the behavior of the available data.

Probabilistic Approaches: In this approach, the Bayesian learning algorithms stand out.

Numeric Approaches: Where we can find algorithms such as: artificial neural networks and vector support machines

Symbolic Approaches: Within this category there are algorithms such as: version space, inductive decision trees, induction of rules and inductive logic programming.

Others Approaches: In this category we find other types of algorithms that do not fit well in previous ones such as genetic algorithms or learning by reinforcement

Now that the different approaches and techniques have been described, we can begin to explain the different types of learning that exist and that coexist with the previous classifications.

2.2.1 Supervised Learning

In this case, the machine learning algorithm will be fed with correct input data and the expected results. This happens during the training phase.

It is a type of predictive learning where the objective is to achieve a function $f : X \to Y$ that is able to predict the value of an attribute Y from the values of X. This is achieved by providing the algorithm with the examples of the form $(x, y)|y = f(x)$

2.2.2 Semi-Supervised Learning

In this type of learning the aim is also to achieve a function like the supervised one and also starting from a set of data. However, only a part of the data that is given to the algorithm is labeled, that is, only some samples will have an associated Y value and the rest will have an unknown value. This forces the algorithm to improve the prediction but this makes the algorithm to provide worse results.

2.2.3 Unsupervised Learning

In this last type the expected results are not provided to the algorithm and only the data is provided. This means that the algorithm will not have any type of knowledge to compare its results. As expected, its results are generally worse than the previous two types, however it fulfills a very important functionality that is to classify unknown data, helping its understanding.

2.2.4 More used Machine Learning algorithms

At this point, we have a general idea of machine learning functionality as well as the approaches used to solve the problems. Therefore, the following will briefly explain the most used machine learning algorithms.

- Supervised and classification-oriented machine learning algorithms
  - Support Vector Machine (SVM): This is a kind of algorithm that performs representations of the different examples in the space and separates them by classes.
  - Discriminant analysis: Its main objective is to give a description of the differences among sets of objects.
  - Naïve Bayes classifier: This algorithm is based on Bayes' theorem to constitute a probabilistic classifier capable of predicting possible results from models.
  - KNN (K-Nearest Neighbor): Generates groups from the input elements. When in it has new input data, it calculates the distance to the different existing groups and assigns it to one of the new groups.
- Regression supervised learning algorithms
  - Linear regression: Algorithm that generates a linear equation that is the best adapted to a set of input data. This equation allows to predict other values.
  - Decision trees: the algorithm generates diagrams from a set of data that serve to represent and classify series of common conditions in the data.

  - Support Vector Regression (SVR): Relative of the SVM, this algorithm uses linear regressions to make the classifications.

- Unsupervised clustering learning algorithms
  - K-means: The set of observations is partitioned into K groups that have a centroid. The addition of new data implies the calculation of new centroids, so that the groups can change in each iteration.
  - Hidden Markov model: This case is more used when it is assumed that the system to be analyzed is a Markov process.
- Searching Algorithms
  - Genetic Algorithms: A family of algorithms that perform heuristic search processes. They are based on natural processes such as reproduction to find solutions to problems.
- Multi-application
  - Neural Networks: This type of architecture is based on brain structures to generate models of a large number of units that imitates neuronal functioning. They can be classified in supervised regression or unsupervised grouping

In the specific case of this project, the solution has been based on the use of neural networks so they will be explained in depth later.

2.3 Deep Learning

The term Deep Learning, also known as deep structured learning, is part of the machine learning family based on the learning of data representations. It is possibly the future of machine learning because it is expected to take the non-supervised learning direction.

As mentioned above, in this case the algorithms are able to learn without the need of human supervision, that is, they are able to find the semantics inherent to the data they process and are able to draw conclusions from them. However, there are some advantages of the supervised learning that the unsupervised will hardly reach as the fact of getting a nearly perfect decision boundary due to the specific definition of the classes.

In this case, the architectures used try to simulate the patterns of biological nervous systems such as deep neural networks, deep belief networks or recurrent neural networks.

Therefore, we could classify deep learning algorithms as those machine learning algorithms that:

- Use a cascade of multiple layers made up of non-linear processing units. Each layer uses the output of the previous layer as input
- Learning can be both supervised and unsupervised
- They are able to learn multiple levels of representation that correspond to different levels of abstraction
- They use descend gradients to train by back-propagation.

CHAPTER 3

Neural Networks

3.1 Introduction

Artificial neural networks are a type of machine learning algorithms used to address problems where there have been a process of collecting data previously and this data is available. This type of algorithms are usually considered as non-linear approximations of the functionality of human brain, that is, these networks base their functionality on the process of transmitting information among neurons. However, they do not intend to simulate these procedures accurately.

3.2 Common Characteristics

The common characteristics that we find among them are the following [9]:

- Non-linearity: a neuron is basic and non-linear element; consequently, a neural network is formed by the interconnection of neurons that will also be non-linear.
- Input-output transformation: the learning process of a neural network involves the modification of internal parameters of the different neurons that conform the network. In addition, information tends to change inside the network
- Adaptability: These types of networks are designed to be altered with simple modifi-

cations of their configuration parameters. This means that a neural network that has been designed to work in a given experiment can be modified to work in a similar but not identical environment.

- Indicative response: in the context of pattern recognition, a neural network can be designed to provide information not only of the selected pattern, but also of the degree of confidence in the decision made. This information can be used later to reject old patterns.
- Contextual information: learning and knowledge are represented by the activation state of the neural network. Each neuron is affected by the activity of the rest of the neurons with which it is connected.
- Fault Tolerance: The massive interconnection of neurons provides this ability, since the death of some of them or their malfunctioning should not greatly affect the final response. This always assuming a relatively large size of neural network.
- Uniformity in analysis and design: architecture, designs and nomenclature are in a state prior to standardization. This is manifested in the use of the same notation in the different areas in which they are used. This characteristic favors universalizing knowledge and offers the ability to work with modular architectures where each module has been designed by different researchers.

3.3 The Neuron

Artificial neural networks are complex systems built on the basis of simple units called neurons. The neuron is the basic and fundamental processing unit of a neural network and it can be observed in the next figure:

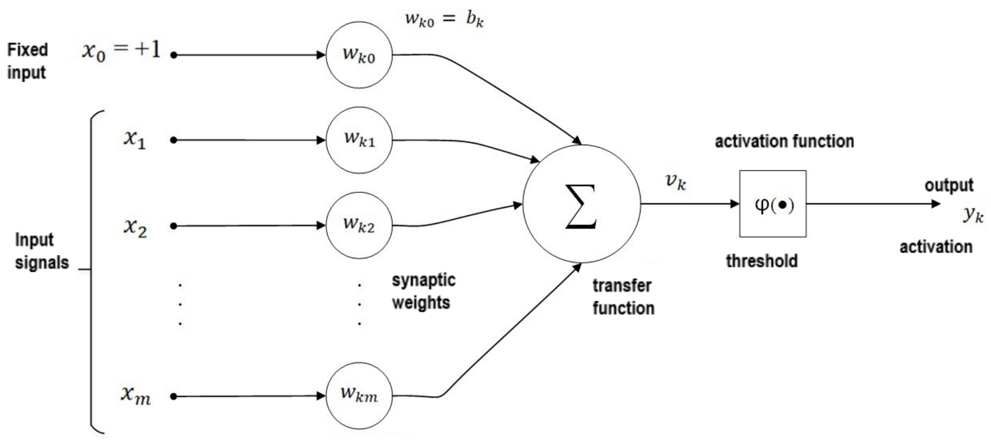


Figure 3.1: Neuron Model

In the scheme it can be observed the next elements:

- Input signals: constitute the base input information to the neuron. They can constitute the first input of information to the network or they can come from the output of other neurons. The neuron will manage many input values as if they were one, which is called the global input.
- Fixed input: known as bias, it is responsible for balancing the input value.
- Synaptic connections: set of connections each of them characterized by their synaptic weight.
- Transfer function: Sum the result of the multiplication of each input signal by the weight of the synaptic connection.
- Activation function: It is responsible of limiting the output between certain values. It is a non-linear transformation.

Mathematically, the neuron is represented in the following way:

$$y_k = \varphi(v_k + b) = \varphi(b + \sum_{j=1}^{m} w_{kj} * x_j) \tag{3.1}$$

The activation function φ can have different forms and it is selected depending on the experiment:

1. Step function:
$$\varphi(v) = \begin{cases} 1 & v_k \geq 0 \\ 0 & v_k \leq 0 \end{cases} \tag{3.2}$$
2. Piecewise linear function: It can be used to reflect the increase of the activation potential that occurs when the input increases.
$$\varphi(v) = \begin{cases} 1 & v_k \geq 0.5 \\ v & 0.5 > v_k > -0.5 \\ 0 & v_k \leq -0.5 \end{cases} \tag{3.3}$$
3. ReLu function: It is a piecewise linear function analogous to half wave rectification function. It constitutes the standard of the activation functions.

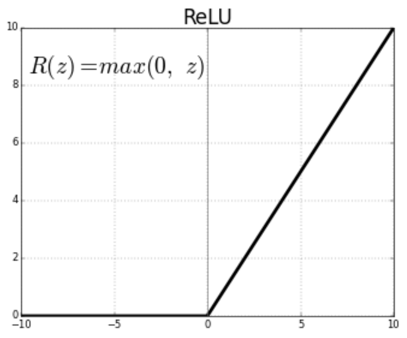


Figure 3.2: ReLU function

4. Sigmoid function: it is a strictly increasing type of function that has an asymptotic behavior and is commonly used for the modeling of many natural processes and learning curves:

$$\varphi(v) = \frac{1}{1 + e^{-at}} \tag{3.4}$$

In the previous equation the slope of the sigmoid is determined by the parameter a.

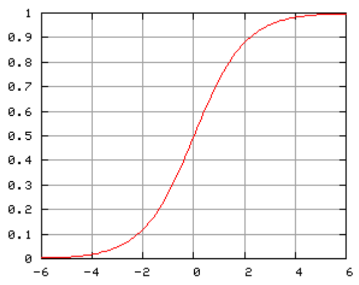


Figure 3.3: Logistic function

5. Softmax function: It constitutes a generalization of the logistic function and is used to compress a K-dimensional vector z of arbitrary values in a K-dimensional vector $\sigma(z)$ of real values belonging to [0,1]. It is a very useful function when there are many possible labeled outputs:

$$\sigma(v) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.5}$$

6. Hyperbolic tangent function: It is a case of sigmoid function usually used

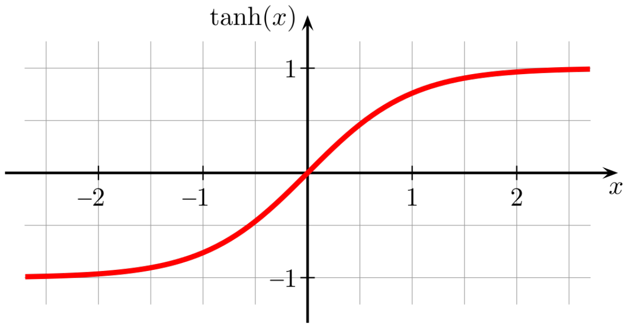


Figure 3.4: tanh

Finally, the output that we find in the scheme will constitute the input for the next neuron. There is an output function, although the identity function is usually taken in such a way that the output is the activation state of the neuron by itself. However, there are neural networks in which the activation state is sometimes transformed to a binary output.

Now that we know the structure and functionality of the different parts of a neuron, we can begin to explain the arrangement of these in a neural network.

3.3.1 Neural Networks Topologies

Neural networks are typically conform by a series of layers of neurons that are linked together by synapses. On the other hand, artificial neurons, as independent units, are not very effective for the treatment of information and, for this reason, they are grouped into larger structures which are called networks.

The distribution of neurons within the network is done in levels or layers where each of them has a certain number of these units. We can distinguish three types of layers:

- Input layer: the layer that receives the input information from outside.
- Output layer: the layer that outputs the information from the network to outside.
- Hidden layer: These are the intermediate layers that are used to process information and communicate with other layers. The number of them is a design decision.

As it has been told, the neurons that constitute each layer are connected to neurons of other layers and these connections can be the following:

- Fully connected: Each neuron of a layer will be linked with all the neurons of the next layer. It is the most used type of connection topology and is used in all types of connections from the Perceptron to BAM networks.

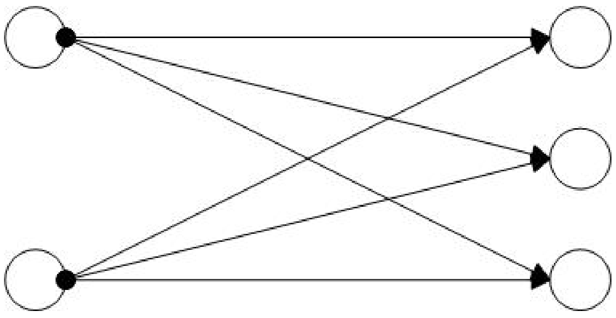


Figure 3.5: Fully-connected topology

- Linear Link: It consists of joining each neuron with another neuron of the other layer. This type of union is used less than the previous one and its objective is usually to join the input layer with the processing layer. It is also commonly used in competitive learning networks.

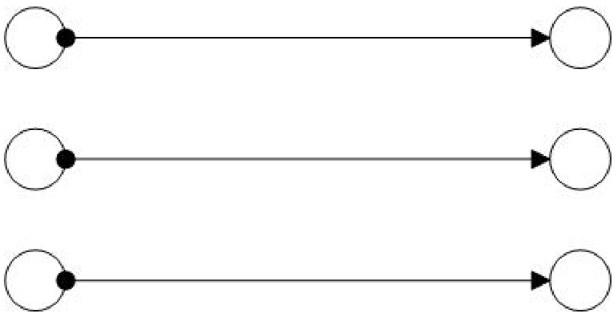


Figure 3.6: Linear link topology

- Default: This type of connection appears in networks that have the property of adding or removing neurons from their layers and also deleting connections.

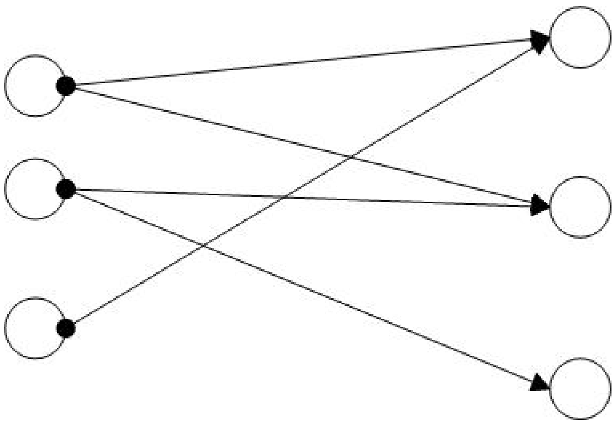


Figure 3.7: Default topology

- Lateral Connections: This type of connections occur between neurons of the same layer and are very common in single-layer networks.

In addition to these types of connections, we can also define an order in the layers, this means, the order of layers through which the processed information is going to advance. Thus, we have two basic topologies which are feedforward and feedback networks. Furthermore, networks that admit its neurons to have connections to themselves constitute other kind called recurrent networks.

Once we know the types of possible connections between neurons we can go on to define the different types of neural networks that we find based on the number of layers as well as the type of connections that exists between them.

3.3.2 Single Layer Neural Network

The simplest type of topology is the single layer neural networks where, as its name suggests, we only have one layer. It must be mentioned that the input layer is not taken in count in the number of layers since it is the layer that receives the information. That is why they are called single layer networks. In the following image we can see the structure of a single layer.

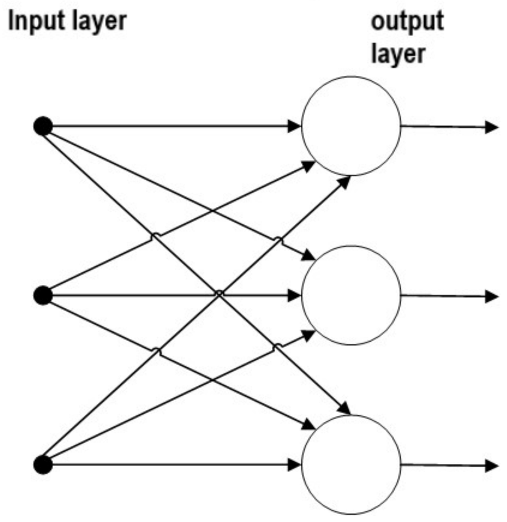


Figure 3.8: Single layer topology

As the input layer connects to the output layer but this does not happen in the opposite direction, we can affirm that it is a feed-forward network.

3.3.3 Multi Layer Neural Network

In this case we find neural networks that have one or more hidden layers between the input layer and the output layer. These layers, as mentioned before, are the layers with the highest processing load. For example, in the case of convolutional neural networks for image processing, these hidden layers are the ones that process the convolutions while the input and output networks only resize the information. In the following image we can appreciate a multilayer network:

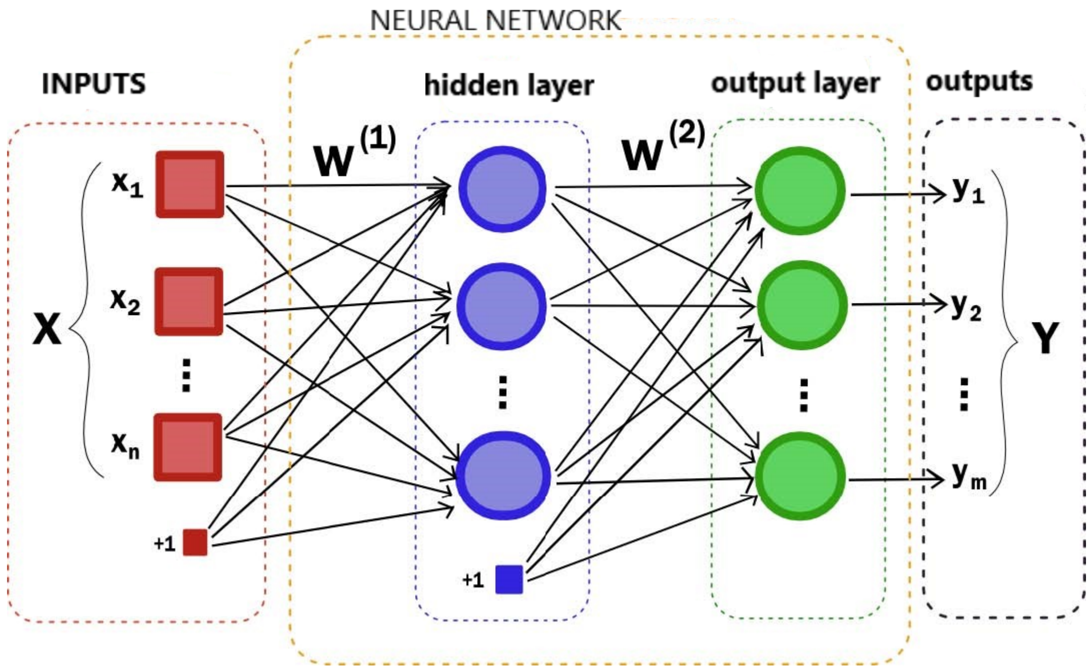


Figure 3.9: Multi layer topology

In the specific case of this figure 3.9 we can see that there is only one hidden layer, but there can be many more. The fact that each neuron in each layer is connected to all the neurons and only in one direction give us the hint that we are facing a fully-connected feed-forward network.

3.3.4 Recurrent Neural Network

These networks differ from the previous ones because they have at least one feedback connection as we can see below.

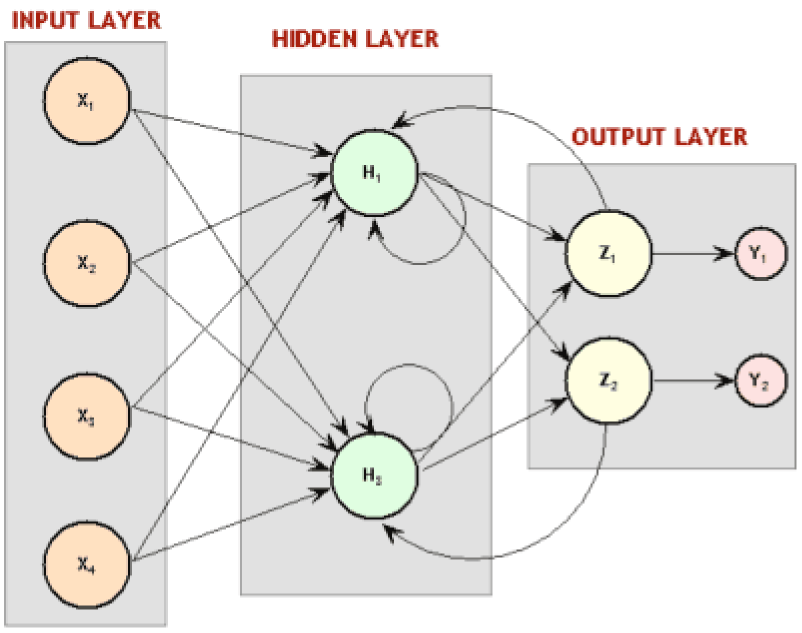


Figure 3.10: Recurrent network topology

This has been the family of networks chosen to develop the solution and, therefore, it is going to be explained in more detail.

Watching at figure 3.10 we can think that the architecture of those neurons differs from the one explained in 3.1. In this case we can imagine these units as it follows:

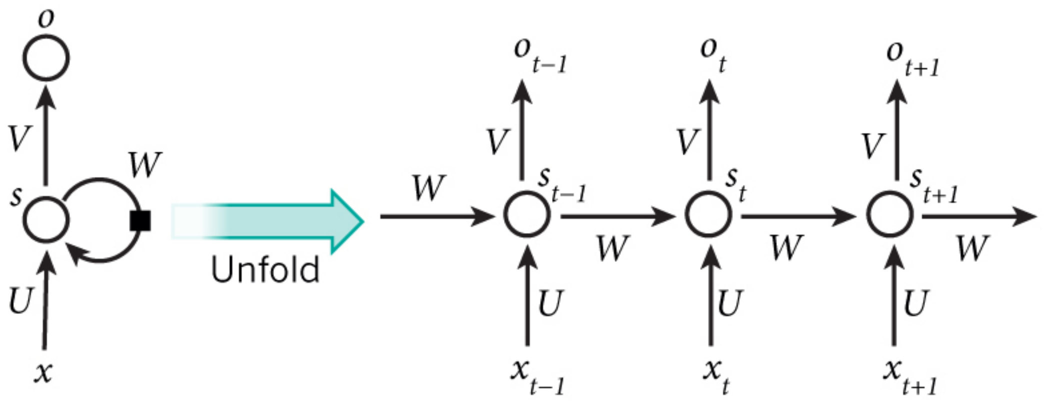


Figure 3.11: Recurrent neuron structure

This means that each neuron can be considered as a time sequence of neurons connected or as multiple copies of the same network, each passing a message to a successor.

These networks were born with the objective of imitating the way of humans think. When a human is performing a task, he understands each part of it and moves forward without forgetting what he learned in the previous steps. This type of networks try to simulate this behavior allowing the information to persist thanks to the use of loops [39, 26].

This chain-like nature reveals that recurrent neural networks are intimately related to sequences and lists. They're the natural architecture of neural network to use for such data.

3.3.5 Common Notation

At this point, it is necessary to give an example of a possible notation in order to understand how the previous explanations fit with what have been explained yet and the next explanations. The following notation is extracted from Michael Nielsen's book [37]

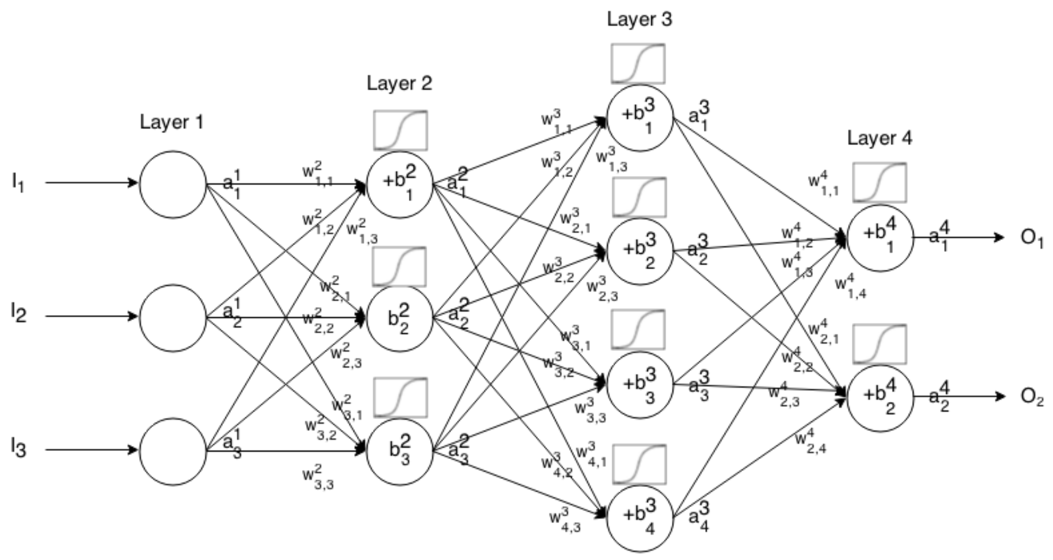


Figure 3.12: Feedforward Neural Network General Notation

As we can observe in the figure, it is a general feedforward neural network that will serve to explain how outputs of one layer are related with inputs of next layer. Attending to the notation of the figure we can define a^i_j as the activation or output of the j^{th} neuron in the i^{th} layer. Therefore the a^1_j is the j^{th} element in the input vector. Hence we can affirm that the input for a layer, related with the previous layer, would be:

$$a^i_j = \sigma(\sum_k (w^i_{jk} \cdot a^{i-1}_k) + b^i_j) \tag{3.6}$$

Here we can appreciate:

1. σ as the activation function.
2. w^i_{jk} as the weight from the k^{th} neuron of the $(i-1)^{th}$ layer to the j^{th} neuron of the i^{th} layer.
3. b^i_j is the bias of the j^{th} neuron in the i^{th} layer.

The term affected by σ can be also expressed as z^i_j, simplifying the equation.

3.4 Learning Process

Going back to neural networks in general, we must emphasize the way of learning. The way in which they interact with the environment is called learning paradigm. These paradigms specify both the type of the input data and the way in which it works with this data. The paradigms were briefly explained in the section 2.2.1 and they are the supervised learning, the semi-supervised and the unsupervised.

In the specific case of this project, supervised learning has been used. It is worth deepening that, in the case of neural networks, we can find different types of supervised learning.

Error correction learning: It consists of adjusting the learning parameters according to the difference between the desired values and those obtained in the output, that is, based on the error obtained at the output. An example of this type of algorithm is the Perceptron learning rule, which is used in neural networks. This is a very simple rule in which for each neuron of the output layer the deviation is calculated with respect to the target output. This constitutes the error which will be used to modify the synaptic weights of the preceding neuron.

Another known algorithm is the LMS Error that also uses the deviation with respect to the target output, but takes in count all the predecessor neurons that the output neuron has. This allows to quantify the global error, at any time of the training, facilitating the learning process because the more information the network has about the error the faster it can learn.

These are usually used in combination with the **backpropagation** algorithm, configuring the most popular tool in training neural networks. This algorithm has two phases. The first one propagates the processed input through the network, layer by layer and finally obtaining an output. This output is compared with the desired output and the error is computed as it has been explained just before.

The second phase has the duty of back propagate the error through the previous hidden layers. However not all the neurons receive the entire error but they receive a part proportional to the contribution they made during the generation of the output. This process, in combination with gradient techniques explained in 3.4.2 is repeated in each layer allowing them to modify their parameters in order to reduce that error.

Learning by reinforcement: It is a slower type of learning than the previous one and bases its operation on the idea of not indicating exactly the expected output during training. In this kind of learning the function of the supervisor is limited to indicate if the output obtained is adjusted to the desired using a reinforcement signal (success = 1 and failure = -1). As a function of this, the weights are adjusted by a probability mechanism. The supervisor is more similar to a critic than a teacher.

This type of learning might seem to generate worse results, but is the basic concept of adversarial generative networks (GAN) where this supervisor is, at the same time, another neural network that does know the expected outputs and learns at the same time than its adversary. This competition between both networks increases the capacity for learning.

Stochastic learning: This learning consists basically of randomly changing the values of the weights of the network and evaluating their effect on the results. In this type of networks, an analogy is made between the network and a physical solid with energy state. By physical laws the solid would always tend towards a state of minimal energy. In a similar way, the weights of the network would tend to obtain the values that better output produce. The energy of the network is usually calculated by the function of Liapunov. If the energy after the change is smaller, the change is accepted, otherwise, is not accepted.

Within this type of learning, Boltzman's learning, which is based on thermodynamic considerations and information theory, stands out. The neurons continue to operate in a binary way as in the general case of stochastic learning where the active state will be repressed with a 1 and the inactive with a -1. In this case we also find an energy function whose value depends on the active or inactive states of the individual neurons. During the operation the machine selects a neuron in a random way and changes its status. Subsequently, evaluates the energy function and the pseudo temperature.

Competitive Learning: In this type of learning there is a competition between neurons in the output layer that compete to generate the best result. This type of learning is very useful in the detection of different patterns within the same data set because each neuron, due to competition, will get specialized in the detection of a specific pattern.

As it has been seen in the previous section, the process of learning, in a neural network, consist on modify its synaptic weights through an iterative process that allows it to improve the obtained results. There are many types of possible algorithms, however the most used is the error correction algorithm. The correction of the error is performed by a family of functions called optimizer functions. There are many different of them and we are going to explore them in the next section.

3.4.1 Cost Functions

The cost functions has the objective of giving a measure of how good is a neural network in the process of generating outputs in relation with the expected ones. It is computed as a single value because it works as a measure.

The general form of cost functions is as follows:

$$C(W, B, S^r, E^r) \tag{3.7}$$

where W represents the weights of the network, B the biases, S^r the input of a single training sample and E^r represents the desired output. The backpropagation algorithm will use this function to compute the error of the output layer:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma^{'}(z_j^i) \tag{3.8}$$

The most used cost function in neural networks and the one which will also be used in this project is the **cross-entropy function**:

$$C_{CE} = -\sum [E_j^r ln(a_j^L + (1 - E_j^r) ln(1 - a_j^L)] \tag{3.9}$$

There are also other important kinds of cost function as the **quadratic**, the **exponential** or the **Hellinger distance** for positive values.

3.4.2 Optimizer Functions

As it has been explained immediately above, this section is going to make a deep review of the different optimizer functions .

Gradient descent: It is the basic optimization algorithm. Its operation consists in the sequential search of the minimum of the error function. The search direction of the minimum, like in the mathematical gradient, will be indicated by the negative of the vector that result of applying the gradient to the objective function [42].

$$\theta = \theta - \mu \bigtriangledown_\theta J(\theta; x^i; y^i) \tag{3.10}$$

There are three variants of gradient descent, which differ in how much data we use to compute the gradient of the objective function:

- **Batch gradient descent:** Is the vanilla gradient descent and computes the gradient of the cost function with relation to the parameters θ. As it calculates the gradients for the whole dataset only to perform one update it becomes too slow for huge datasets. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces

- **Stochastic gradient descent (SGD):** This kind of gradient performs a parameter update for each training example x^i and label y^i. The formula written in the general explanation is the formula of this particular case 3.10.

- **Mini-batch gradient descent:** In this case, this gradient takes the best of both previous gradients and performs an update for every mini-batch of n training examples.

$$\theta = \theta - \mu \bigtriangledown_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{3.11}$$

Adaptive Gradient Algorithm (AdaGrad): Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. That maintains a per-parameter learning rate that improves performance on problems with sparse gradients (e.g. natural language and computer vision problems)[42].

Adadelta: is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w.

Root Mean Square Propagation (RMSProp): that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight (e.g. how quickly it is changing). This means the algorithm does well on online and non-stationary problems (e.g. noisy).

Adam: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. It computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t Adam also keeps an exponentially decaying average of past gradients m_t, similar to the momentum:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t vt = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{3.12}$$

Adam realizes the benefits of both AdaGrad and RMSProp. Instead of adapting the parameter learning rates based on the average first moment (the mean) as in RMSProp, Adam also makes use of the average of the second moments of the gradients (the uncentered variance). Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages [7].

AdaMax: The v_t factor in the Adam update rule scales the gradient inversely proportionally to the l_2 norm of the past gradients. If we generalize this update to the l_p norm we can observe that larger values of p generally become unstable and that is why l_1 and l_2 norms are commonly used. However $l_\forall$ exhibits stable behaviour This is the difference between Adam and AdaMax [42].

Nadam: This optimizer combines Adam and NAG (Nesterov accelerated gradient). The main difference with Adam is that NAG allows us to performe a more accurate step in the gradient direction by updating parameters with the momentum step before computing the gradient.

As it was introduced in the **Error correction learning** in 3.4, this optimizer functions commonly use the backpropagation algorithm to adjust the weight of the neurons after computing the gradient of the loss function.

3.4.3 Which optimizer should be used?

This is a common question at the beginning of a project and it does not have a sure answer until you use some of the above possibilities. At first, it is interesting to study the kind of data is going to be used in the experiment. If it is sparse data the best results will be reached by one of the adaptive learning rate methods, that also have the benefit that it is not necessary to tune the learning rate but likely achieve the best results with the default value.

RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. In [28] it is shown that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice. However, SGD usually achieves to find a minimun, but it might take too much longer than with some others optimizers.

In this project, some of them have been used and we will see the different results later.

3.4.4 Problems with the use of gradients in learning

There are two big problems when using gradient based methods for learning which are vanishing gradient and exploding gradient.

Vanishing gradient is a problem found in training artificial neural networks with gradient methods and back propagation. In such methods, each neural weight receive and update in each iteration of training. This update is proportional to the gradient of the error function. The problem appears because sometimes the gradient will be decreasing each iteration and it is possible that reaches some values that prevent the weight from changing its value. In the worst case, this may completely stop the network from training. We can find a common example in the application of the hyperbolic tangent with back propagation where the range of the tanh is (-1,1). The gradient will decrease exponentially and the training will become very slow [36].

Exploding gradient: In this case we find a different behaviour where the norm of the gradient starts to increase exponentially during the training due to the explosions of the long term components. It is a behavior contrary to the vanishing gradient.

3.5 LSTM Networks

3.5.1 Introduction

At this point, we know we are going to use recurrent neural networks with different optimizer functions in order to see which gives better results. There are other many variables that affect the quality of the results such the pre-processing of the data, the size of the neural network, the learning direction, etc.

In the special case of this project, where the main objective is to build a conversational bot, we find other important problem that appears when using recurrent neural networks. This problem is known as the problem of long-term dependencies and it had its influence in the decision of what kind of neurons will be used in the solution.

3.5.1.1 The Problem of Long-Term Dependencies

One of the appeals of RNNs is the idea that they are able to connect previous information with the current task due to its recurrent nature. This nature was the key to select them to develop the solution because taking in count previous information is a vital point in order to maintain a coherent conversation. The problem is that RNNs are not always capable of doing this.

Sometimes, we only need to look at recent information to perform the current task. As we are working with some kind of language modelling, we can imagine a language model trying to predict the next word based on the previous ones. In some cases the relevant information is going to be near to next prediction but sometimes it's not. For example imagine the sentence "the clouds are in the sky" and the algorithm is about to predict the word "sky". It's obvious that we don't need any further context more than the sentence by itself. In this cases, where the gap between the relevant information and the place that it is needed is small, this kind of network can perform the process of learning using the recent past information.

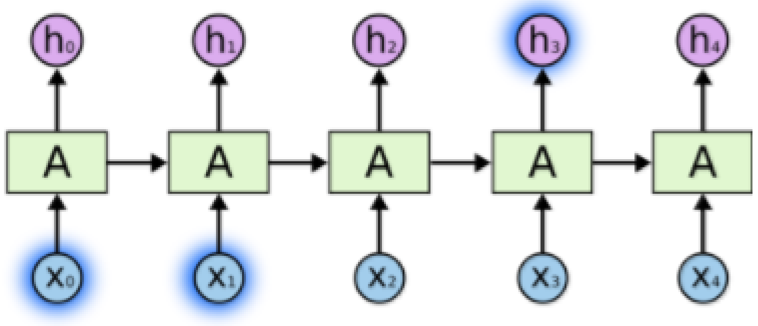


Figure 3.13: RNN long term dependencies A

But there are cases where is needed a further context, such as conversations where we use information given at the beginning like the genre or the nationality. Consider a similar example to the previous one where a person says "I grew up in Spain… I can speak fluent Spanish". If both sentence are near in the time is possible to suggest the word Spanish which is the name of the language spoken in Spain but the gap between both sentences can have much more information which is not relevant. Unfortunately, as the gap grows, RNNs become unable to learn to connect information [5].

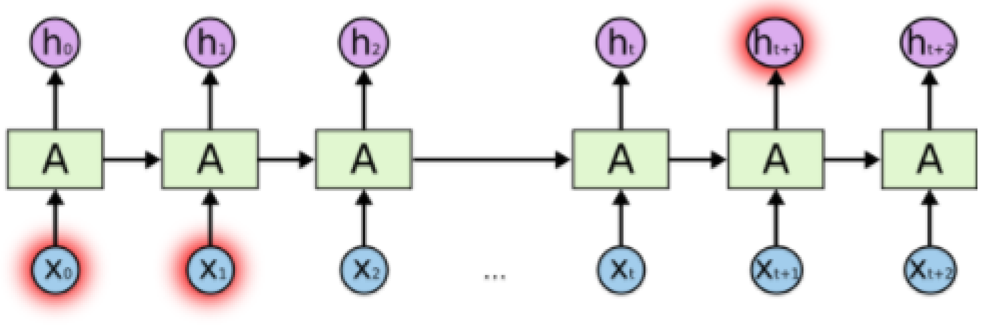


Figure 3.14: RNN long term dependencies B

A human could carefully pick some parameter in order to avoid this problem with simple examples but not for difficult ones. Thankfully we can use LSTMs to solve the problem.

3.5.2 Main ideas behind the LSTM architecture

Long Short Term Memory, usually known as LSTMs are a special kind of RNN which are capable of learning long-term dependencies. They were presented by Hochreiter and Schmidhuber in 1997 [24], and they were refined by many other authors and researchers.

Now they are widely used because they are specially designed to avoid the problem explained above.

As it has been explained in 3.3.4, all recurrent nets have the form of a chain composed by repeating modules as we can see below [39].

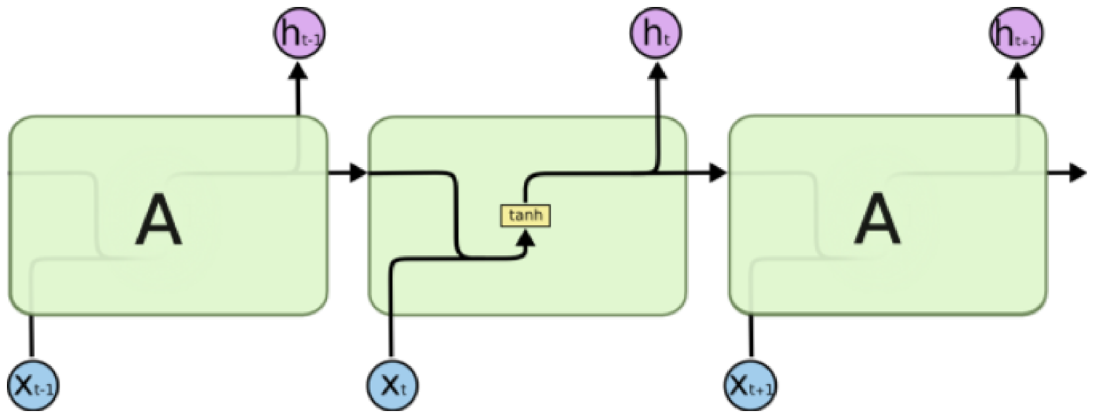


Figure 3.15: Detailed RNN unit

This image is a more detailed one than the presented on 3.11 In this example we have a simple RNN structure with a single layer where we have a tanh activation function. In the case of a LSTM, it has also the chain structure but with other special layers.

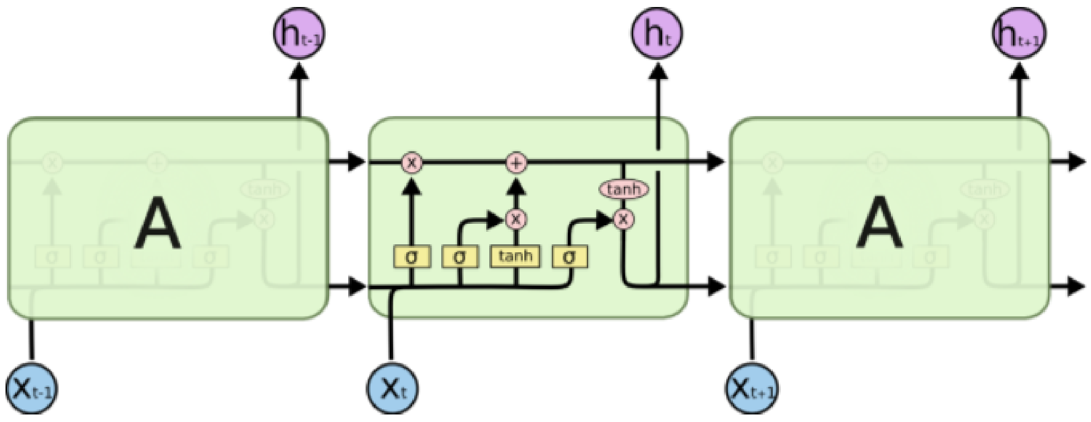


Figure 3.16: Detailed LSTM unit

This diagram is going to be explained in depth in the next epigraph, but it is important to know that each of the yellow boxes represents a layer of the neural network. In the next section there is going to be an step-by-step explanation of how a LSTM unit works.

The key in the LSTM functionality is the cell state which is the horizontal black line that goes through the top of the diagram. This state runs straight down the entire chain and it suffers some simple interactions as linear operators. It is important to mention that operations represented as pink circles are element-wise operations.

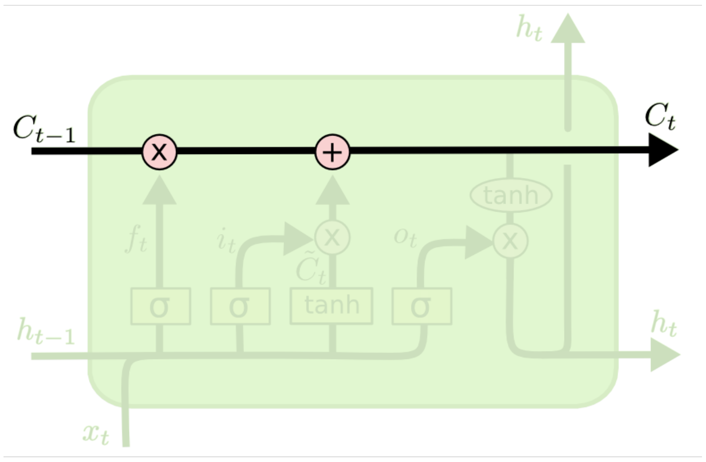


Figure 3.17: LSTM cell state

This line constitutes the main way for the information to flow in the RNN chain and the LSTM does have the ability to add or remove information using some kind of structure called gates. The information go through these gates depending on a sigmoid layer and a pointwise multiplication operation. We can observe this in the top left corner of the unit.

The sigmoid layer "decides" how its information affects the main state cell information flow with outputs between zero and one describing how much of each component should be let through. Obviously a value of zero means that nothing is going through and a value of one means everything goes through.

First of all, a LSTM unit is going to decide which information is going to be thrown away from the cell state. This is made by the sigmoid layer, also known as "forget gate layer". It takes h_{t-1} from the previous unit and x_t which constitutes the new information for the current task. The sigmoid layer outputs a number between zero and one for each number coming in C_{t-1}. If we perform a pointwise multiplication with all ones it will mean to preserve all the information and viceversa, if we have a vector of zeros, it will "forget" all the previous information. Continuing with the previous dialogue example, if we change the genre of the subject in the new input data x_t , the result of the sigmoid would be probably all zeros because it will have to decide answers without taking in count the previous genre information.

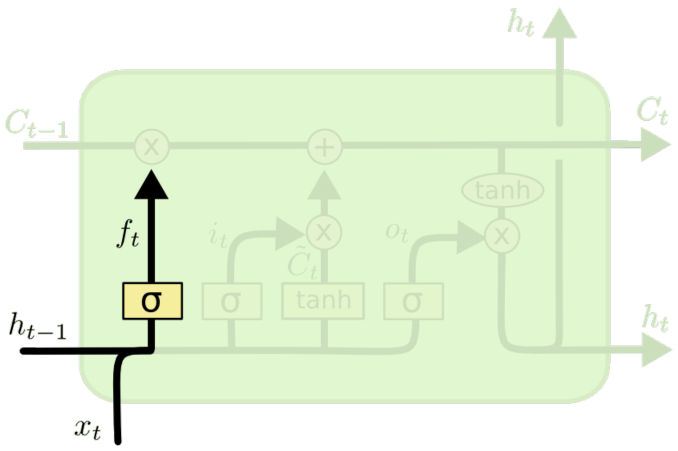


Figure 3.18: LSTM forget gate

So we have an output from the sigmoid layer which consists on the multiplication of a weight matrix W_f with the input data at current time step t and the hidden state of the previous time step coming from the previous unit.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.13}$$

The next step is about the decision of what new information will be stored in the cell state. One that will update information and other that will add new information. This is performed by two layers, the sigmoid one will decide which values will be updated and the tanh one will create a vector of new candidates C'_t. If we continue thinking on the genre example, in this step we would want the LSTM unit to add the new genre as new information and to forget the previous one.

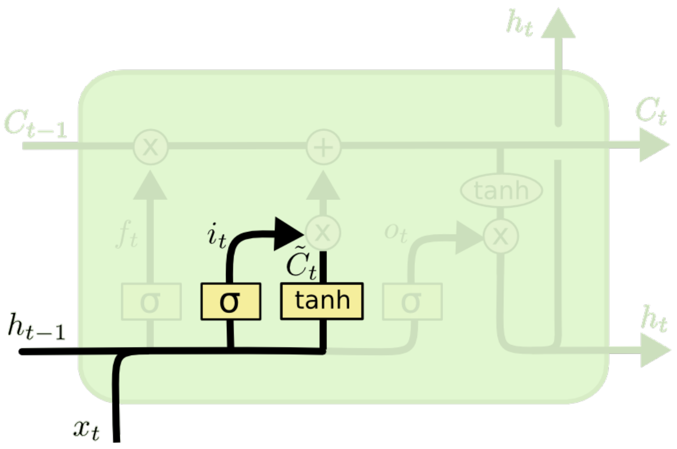


Figure 3.19: LSTM new candidates

In this case both layers will have two matrices of weights (W_i and W_c) which will be multiplied but its respective inputs.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.14}$$

$$C_t^{'} = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \tag{3.15}$$

So now we have the information we want to forget, the information we want to update and the information we want to add. These will update the cell state from C_{t-1} to C_t. In order to get this, as it has been said before, the LSTM multiplies C_{t-1} by f_t in order to forget some information. Then add $i_t \circ C_t^{'}$ which represents a vector of new candidates values.

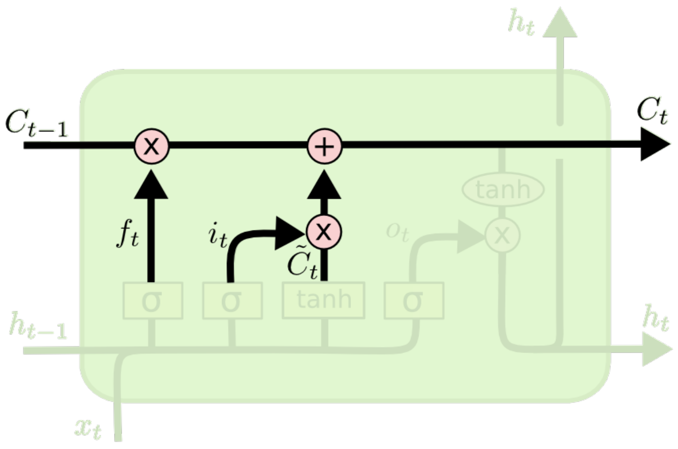


Figure 3.20: LSTM combination of two gates

$$C_t' = f_t \circ C_{t-1} + i_t \circ C_t' \tag{3.16}$$

Now the vector of information that represents the cell state is completely updated and is almost ready to be the output. But first, it suffers another transformation that filter the information that is going to be used in the next LSTM unit. It goes through a sigmoid layer that decides which parts of the cell state will constitute the output. Then a tanh layer will push values between -1 and 1. The output of this step is then multiplied by the output coming from the sigmoid. Doing this the LSTM will only output the parts we decide to. We can imagine the function of this last step if we imagine that the network performs changes in order to get relevant information of the subject to use the correct form of the verb.

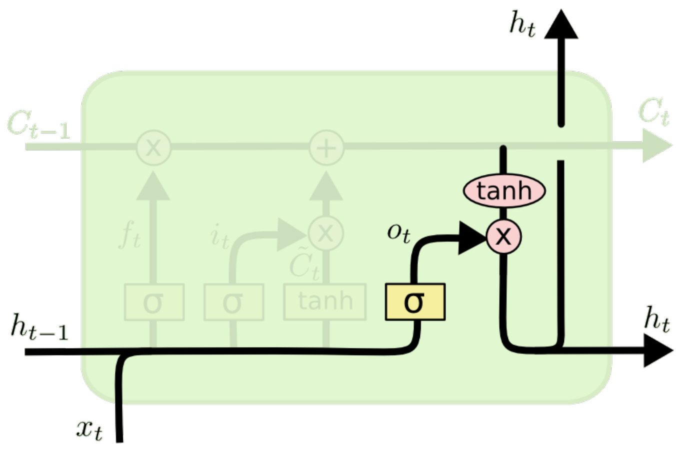


Figure 3.21: LSTM final step

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.17}$$

$$h_t = o_t \circ tanh(C_t) \tag{3.18}$$

As expected, this is the common implementation of a LSTM but there are many other possibilities. In fact, almost each new paper involving LSTMs uses a slightly different version. Because of this, it's worth to mention some of them.

One popular variant was introduced by Gers and Schmidhuber in 2000 [20] and it has the curiosity of adding peephole connections.

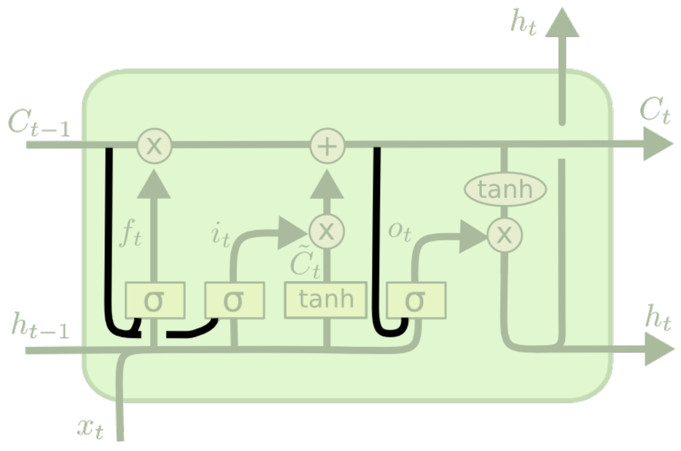


Figure 3.22: LSTM with peepholes

$$o_t = \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o) \tag{3.19}$$

$$i_t = \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \tag{3.20}$$

$$f_t = \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \tag{3.21}$$

The presence of the peepholes is not always the same. In fact, depending on the paper, some authors will select some peepholes and delete the others. The importance of this schema is that different layers are allowed to look at the cell state so the decisions of what to forget or what to add are taken together.

Another variations is to combine forget and input gates. In this case, these two task are not performed separated but together. The philosophy is to forget only when we are going to input new information in its place and vice versa. This kind of schema looks like the next figure.

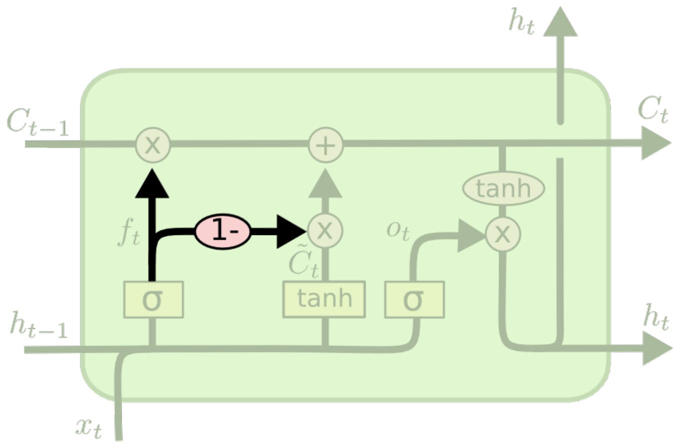


Figure 3.23: LSTM mixed gates

$$C_t = f_t \circ C_{t-1} + (1 - f_t) \circ C_t^{'} \tag{3.22}$$

At this moment we have explained step by step how LSTM works. All the matrices mentioned and presented on the formulas constitutes matrices of weights, that is, matrices of learnable parameters which will go changing with the backpropagation and they will go learning what to forget, add or update.

These are only a few LSTM variants which are common in different projects. There are lots of others like Depth gated RNNs by Yao,et al. (2015) [51].

Obviously, maybe the question of which of them to use come up but this is answered in Greff, et al. (2015) [22] . This paper, after analyzing many of them, reaches the conclusion that they are all about the same and have, more less, the same results in a variety of contexts. However, is not a bad idea to prove some of them in an experiment in order to know which has a better performance.

But this is not the only thing that LSTM allows. They also represent a solution to the vanishing gradient problem because they preserve the error that can be back-propagated through time and layers. By maintaining more constant error, LSTM allow recurrent networks to continue learning over many time steps.

In the next image we can appreciate the differences between a normal unit in a recurrent network (left) and a LSTM unit (right).

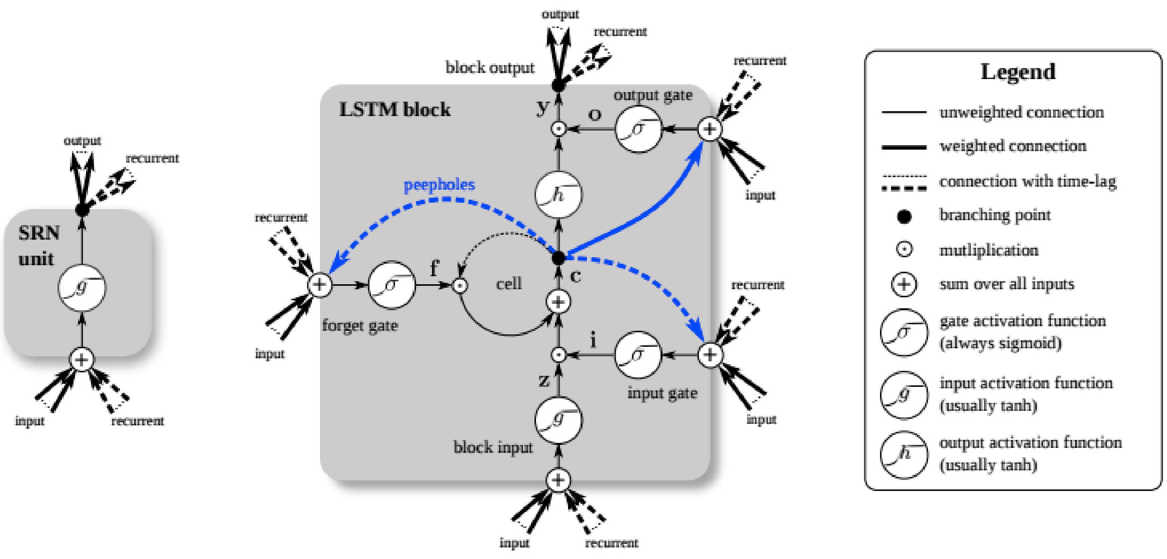


Figure 3.24: Difference between RNN unit and LSTM unit

As we can easily observe, the complexity is much higher and this has its effect in computational costs but it is worth using them because all the advantages mentioned before.

3.5.3 Gated Recurrent Unit

This kind of unit, also called GRU, was proposed by Cho et al. (2014) [11] to make possible the fact that any recurrent unit could capture dependencies of different time scales in a adaptively way. The architecture of a GRU unit is similar to the LSTM and also has gate units that control the how information flows in the unit. The activation h_t^j of the GRU unit can be defined at a time t as a linear interpolation between the previous activation h_{t-1}^j and the candidate activation $h_t^{'j}$:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j h_t^{'j} \tag{3.23}$$

The z_t^j represents an update gate which decides how much the unit updates its activation. Similar as it was previously explained, the update gate can be expressed as it follows:

$$z_t^j = \phi(W_z \cdot x_t + W_z \cdot h_{t-1})^j \tag{3.24}$$

We can appreciate that this linear sum is a similar procedure to compute the new state in LSTM unit. However, GRU unit does not have mechanisms to control the degree to which its state is exposed, but exposes the entire state each time.

The candidate $h_t^{'j}$ is computed in a similar way to that of the traditional recurrent unit:

$$h_t^{'j} = tanh(W \cdot x_t + U(r_t \circ h_{t-1}))^j \tag{3.25}$$

Where r_t represents a vector of reset gates and the operator is an element-wise multiplication. When some of the gates in the set r_t are close to 0, the reset gate makes the unit work as it was reading the first symbol in an input sequence, that is, as it was forgetting the previous state, like in an LSTM unit.

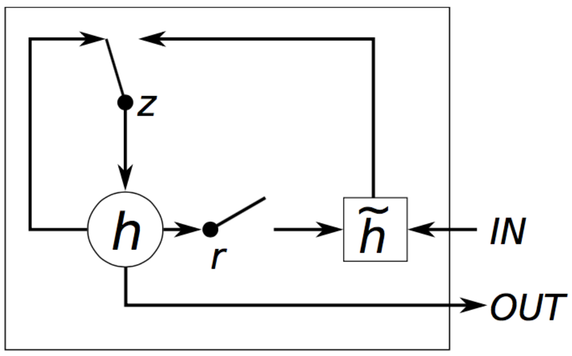


Figure 3.25: GRU unit

As we can see, the structure is very similar to the one of the LSTM unit, but it is simpler. A GRU has two gates, the update gate z and the reset one, r. This last gate determines how to combine new input with the previous information in the unit, and the update gate determines how much information to update, or in other words, how much previous memory to keep. The rest of the differences are:

- GRUs units have two gates and LSTMs have three.
- GRUs don't have and internal memory independent from the hidden state. They do not have the output gate presented in LSTM.
- Here, the responsibility of the reset gate which is present in LSTM is split up into both r and z.
- When computing the output is not necessary to compute a second non-linearity.

Obviously the benefits of using GRU instead of LSTM are the faster computing with GRU due to the smaller matrices U and W. Nevertheless, with big amounts of data LSTM may lead to better results.

3.5.4 Summary

So as a final summary of this section we can think about Recurrent Neural Networks as a kind of networks compose by units that is able to "remember" information by modelling the data they are exposed to.

We have seen that they produce dynamic models in order to yield accurate classifications dependent of the context of the examples. This is allowed by the hidden state that determines the previous classification in a serie. In each step, last hidden state is combined with the new one to generate a new hidden state and a new classification. Is in this way how they are context aware.

To get this propose in more complex context where the related information can be separated by a big gap of time, the network needs to recall different short-term memories at different times. Some of these "memories" will have been forged recently and maybe others were forged many steps before. Here the LSTM units have the aim of correctly associate this "memories" with new inputs.

Due to this, recurrent networks have the capacity to predict. They grasp the structure of data over time and they use it to predict the next element. These new elements might be next letter in a word or next words in a sentence. For all these reasons explained before, **recurrent networks with LSTM have been selected to develop the solution of this project**.

3.6 Technologies

Now that we have seen all the background context and the theory behind the neural networks and the kind of units, connections and layers, it is necessary to explain how they are going to be implemented.

Nowadays, there are some programming languages and some that allows anybody to implement neural networks. Let's make a short overview of the most representing libraries and languages.

Torch

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first. It is based on LUA which is a fast and efficient scripting languages what makes of it an efficient and easy to use option. Some of it features are:

- Powerful N-dimensional array
- Neural network models
- Fast and efficient GPU support

It was very popular but it has been losing presence in projects and papers due to the appearance of new libraries. Despite of this, some big companies continue using it in some projects as Facebook or Google.

Theano

Theano is a Python library that allows to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It is primarily developed by academics and has been powering large-scale computationally scientific investigations since 2007. Some of its features are:

- Despite of run on top of Python, it has its own data structure which are tightly integrated with Numpy, which is a fundamental Python package for scientific computing.
- Transparent use of the GPU.
- Efficient symbolic differentiation because of the capability of derivate functions with many inputs.

Unfortunately, Theano was used less and less and has been abandoned.

TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. It was develop by Google (specially Google Brain) and it was released on the last semester of 2015. As it has been said, TensorFlow uses graphs where nodes represent mathematical operations, while edges represent the multidimensional arrays (tensors) communicated between them. Some of its features are:

- Flexible architecture that allows to deploy computation to one or more CPUs or GPUs with a single API.
- Fast and efficient
- It can use TPU which are a kind of CPU specifically oriented to compute with tensor on tensorflow.

This library also offers a desktop visualizer of the graphs and some parameters during the process of training. It is call TensorBoard and it can help to see in “real time” how the algorithms are evolving with the information.

Nowadays it is probably one of the most used libraries for neural networks, maybe due to the fact that it have been designed specifically to solve this kind of problems. Google's backing also makes that the community of developers grow quickly.

Pytorch

Pytorch is deep learning framework based on Python. It offers two high-level features:

- Tensor computation with strong GPU acceleration.
- Deep Neural Networks built on a tape-based autograd system.

This package allows to reuse most common Python libraries as numpy, scipy or Cython. It also uses tensors and graphs that can be computed both on CPU or GPU.

One special characteristic that makes a difference with the previous libraries is that it uses a technique called Reverse-mode auto-differentiation, which allows to change the way the network behaves arbitrarily with zero lag. This is not unique in PyTorch but it's the fastest implementation.

Other special characteristic that improves the situation is that is deeply integrated into Python and only in Python and leaves the asynchronous execution. This allows a faster debugging in case it is necessary.

CNTK

The Microsoft Cognitive Toolkit - CNTK – is a unified open source deep-learning toolkit develop by Microsoft. It can be included as a library in your Python, C# or C++ programs, or used as a standalone machine learning tool through its own model description language (BrainScript). CNTK can be used both in 64x Windows or Linux and, nowadays, is the third most popular tool only behind Tensorflow and Caffe, which is another deep learning framework developed by Berkeley AI Research. Some of the features we can find in Microsoft's are:

- CNTK is in general much faster than Tensorflow.
- CNTK can be easily scaled over thousands of GPUs
- Inference: it has a C#/.NET/Java inference support that makes it easy to integrate into user applications

Sadly, despite of it is a good framework, the community behind other options is bigger and it makes easier to find documentation about some typical problems.

Keras

Keras is a high-level API for neural networks. It is based on Python and is capable of running on top of TensorFlow, CNTK or Theano. The idea behind Keras was to enable fast experimentation. This give the users some capabilities that we can explore below:

- User friendliness: The API was developed for humans not for machines and the user experience is in the center of this idea, minimizing the number of user actions required for common use cases.
- Modularity: Fully-configurable modules that can be combined with little restrictions. In particular, layers, cost functions, optimizer, activation functions, etc, are standalone modules ready to combine and create new models. In addition, is very simple to add new modules or extend the ones existing.
- Python: the modules are described in Python which makes them compact and easier to debug.

On the other hand, if there is not too much interest in how the layers are connected, how activation function works, etc, Keras is a good option because the connections between the modules are a kind of transparent for users. In addition, the process of setting up small and medium experiments is pretty simple, so is a good choice for proving.

In this case, it has been decided to use both TensorFlow and PyTorch. The decision came up because both technologies are booming and they have bigger communities than others, with many people contributing and developing examples and projects. There are also big forums where people discuss about typical problems in some kinds of experiments and they contribute with particular solutions.

Keras has been also used in the process of learning how coding neural networks is and to develop previous and simpler examples than the one that concerns this project.

CHAPTER 4

The Problem

This chapter describes the main characteristics of the problem that this project aims to resolve.

4.1 Background

As it has been explained in the initial abstract and in the introduction, the use of neural networks is rising in many scientific and non-scientific fields. Some of those fields are natural language processing (NLP) and machine translation.

Natural language processing is a field of computer science, artificial intelligence and linguistics that studies the interactions between machines and human language. It is focus on investigate efficient mechanisms to allow humans to communicate with machines using natural languages.

On the other hand, machine translation is an area of linguistic computation which investigates the use of software to translate both text and speech from one natural language to other.

With the advance of neural networks, specially on the image recognition field with convolutional networks, researches started to use them trying to solve some problems related with the previous explained areas. Hence, following this way, the first neural machine translation algorithm appeared and they are giving very good results currently. There are also other possibilities to perform translation process like rule-based or phrase-based translation but they do not reach as good results as neural translation is reaching. This encourages Google and other big companies to deploy its neural machine translation solutions, becoming the base of current online translators.

Despite of this is a project about a conversational bot, it is very interesting to know how neural machine translation (from now **NMT**) works because this kind of algorithms are easily extrapolated to the case of conversational bots with little changes. Hence, NMT algorithms are going to be explained deeply in the next epigraph.

4.2 Neural Machine Translation

This approach can be classified by a Vauquois Triangle which can be observed in the following image

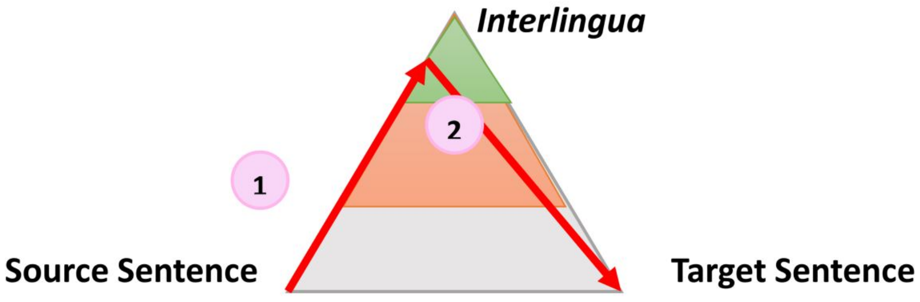


Figure 4.1: Vauquois Triangle

This is divided in two parts that are:

- Analysis: This part is commonly called encoding and generates a sequence of vectors with a representation of the words in the input. Later this vector will be explained in detail.
- Transfer: Also known as decoding, generates the target form.

These two parts conform two different modules in the architecture of the network. Both modules constitute the pair encoder-decoder and they are going to be explained now.

4.2.1 Seq2Seq

The Sequence to Sequence model is an architecture that have been introduced and explained in multiple articles [45, 10].

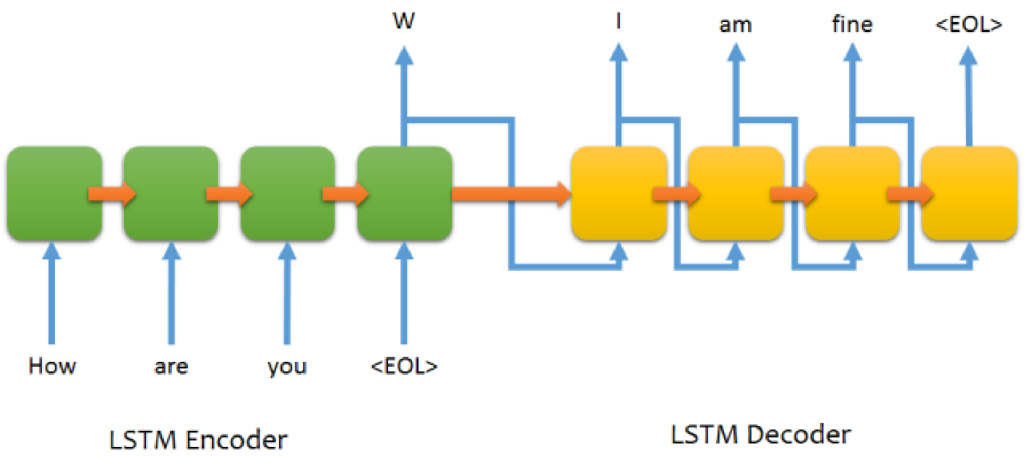


Figure 4.2: Seq2Seq Model

In this architecture it can be observed two different structure which are the encoder and the decoder. They are both two RNNs and, in this particular case, each unit is a LSTM, so we can imagine each of the coloured squares as a LSTM unit as it has been seen in 3.5.2. Despite of they have the same architecture, they behave totally different and in many cases they perform opposite task. In one hand, the enconder is going to take sequences of input information (sentences in this case) and will process one symbol (in this case they are words) at each timestep. It will translate this sequence of symbols into a vector of characteristics where the important information of the input is going to be encoded. In the next picture we can observe a simple representation of this phenomenon.

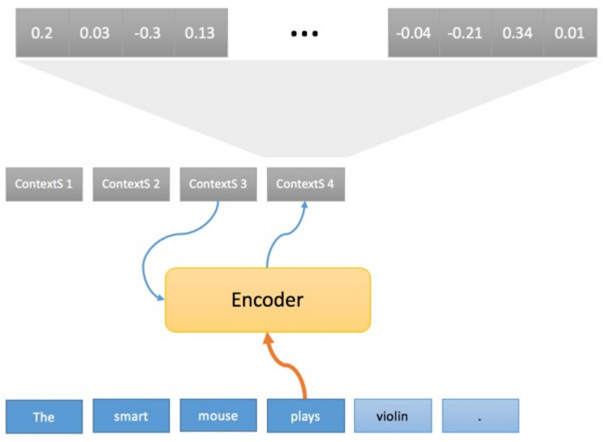


Figure 4.3: Encoder Generating Context

Each timestep a new word is processed so it is giving or removing information from the vector that represents the context. This process is the one which was explained step by step at 3.5.2.

Therefore, the last hidden state (in the last green square) will have the summary of the sequence's information. This constitute what is called thought vector and it is passed to the decoder when the tag "<eos>" (end of sequence) appears [47]. Then, from this thought vector, the decoder generates another sequence, producing a symbol (again on word) each timestep. In this case the generation of the symbols is influenced by previous context but also by the previous generated symbols and it will also finish with an <eos> tag [45].

The strength of this model lies in its simplicity and generality. Because of this, the seq2seq model can be used in many task as machine translation, question/answering and conversations by doing little changes in its architecture.

One problem in this model is the use of a dictionary of the most used words in a language. Probably this dictionary will have more than 50.000 entries, so the decoder will have to run the activation function over a large number of words for each word in the output, trying to map it to the most probable one. This may cause a slow down of the training process.

Despite of these problems, there are some techniques that try to reduce the impact and that will be explained below.

4.2.2 Padding

This is a technique used before learning and it converts the variable length sequences into fixed length sequences by padding. The padding process add as many <PAD> symbols to the original sentence as it is necessary to reach the fixed length. Other symbols are also used:

- EOS: end of sequence.
- GO: start encoding.
- UNK: unknown word. When the word is not in the vocabulary.

Those other symbols both fulfill their mission and influence in the padding. So if we have the pair of sentences of the image 4.2 [How are you?, I am fine] and a fixed length of 8, the sequences of symbols will be:

["How","are","you","?",PAD,PAD,PAD,PAD,EOS]

[GO,"I","am","fine",".",EOS,PAD,PAD]

By doing this, we can solve the problem for an experiment where we have short sentences, so the fixed length is not going to be very big. However, this solution is not enough for experiments where the lengths are sparse and exist a huge difference among them.

4.2.3 Bucketing

As it has been introduce above in section 4.2.2, it exist the possibility of facing problems where sentences have sparse lengths where the addition of many padding symbols may overshadow the real information in the sentence (short sentences with tens of padding symbols).

In order to solve this, the bucketing process form groups of different lengths and put them into the same bucket, building an array of buckets each of them having a pair of lengths. Imagine next array of buckets:

(5,10),(10,15),(15,20),(20,25)

If the input is compose by a question of length 4 and a 5 symbols length answer, the bucketing process will pad the question with only a symbol and the answer with 5 symbols (5,10). This make the model change in each iteration to be compatible with the sequence length, independently of it is training or predicting. Despite of this changes, models share the same parameters and, thanks to that, its functionality does not change. However, this technique is not mandatory when using RNN due to the fact that they can deal with variable length sequences.

4.3 Word Embedding

In section 4.2.1, and specially in figure 4.3, it has been a slightly introduction to the embedding concept, but in this section we will continue the explanation in more depth.

In essence, word embeddings are a type of word representation that allows to represent words with similar meanings in a similar way. Despite of the idea seems to be easy and clear, the technique behind it may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems [21]. Generating word embeddings with a very deep architecture is too computationally expensive for a large vocabulary. This is probably the main reason why it took until 2013 for word embeddings to appear in the NLP stage.

In fact, word embeddings are a group of techniques that allows to represent words as vectors in a vector space. The values of each vector where each word is mapped, are learned as it would be in a neural network.

Each word is mapped to a vector of real values that have often tens or hundreds of dimensions [4]. This dimensions will represent different aspects of the word, making possible that words that are used in similar ways will have similar representations. By doing this, we will get almost all the dictionary words represented with only a combination of hundred of features, which is much smaller than a 50.000 entries dictionary.

We can imagine that with colour, shapes and sizes constituting a 3 dimensions world.

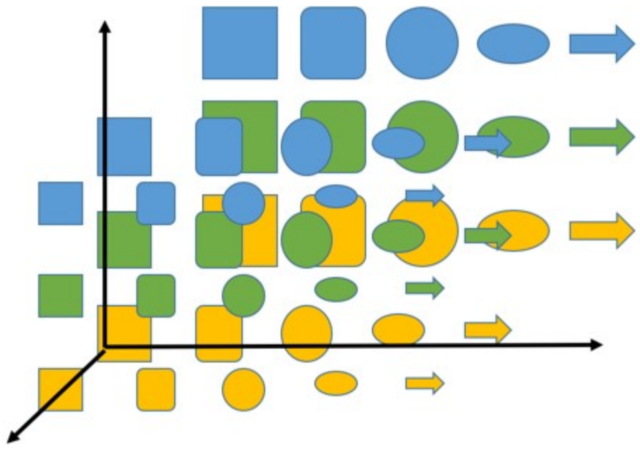


Figure 4.4: Embeddings Simple Representation

This idea comes from linguistic studies showing that words with similar context will have similar meanings [23].

To go on deeply in the explanation, it is necessary to assume some notation. We assume that the training corpus has sequences of T training words $w_1, w_2..., w_T$. With all the words (or only a determined number of them) of the corpus we create a dictionary V which is a vector of size $|V|$. Then, each word is associate with its input embedding (v_w) which will have d dimensions. The output embedding will be (v'_w). Finally we have also a objective function J_θ related with parameters θ. Word embedding models are often evaluated using perplexity, which is a cross-entropy measure taken from language modelling [2, 8].

It exist a strong relation between word embeddings models and language models. Therefore, it is important to know that language models computes the probability of a word w_i by applying the chain rule with the $n-1$ previous words. Applying this rule with the Markov assumption, the product of a entire sentence can be approximated by the product of the probabilities of each word when given the n previous:

$$p(w_1, w_2..., w_T) = \prod_i p(w_i | w_{i-1}, ..., w_{i-n+1}) \tag{4.1}$$

If it is used a neural network to compute that probability, with a softmax layer, we obtain:

$$p(w_1, w_2..., w_T) = \frac{e^{h^\top v'_{w_t}}}{\sum_{w_i \in V} e^{h^\top v'_{w_i}}} \tag{4.2}$$

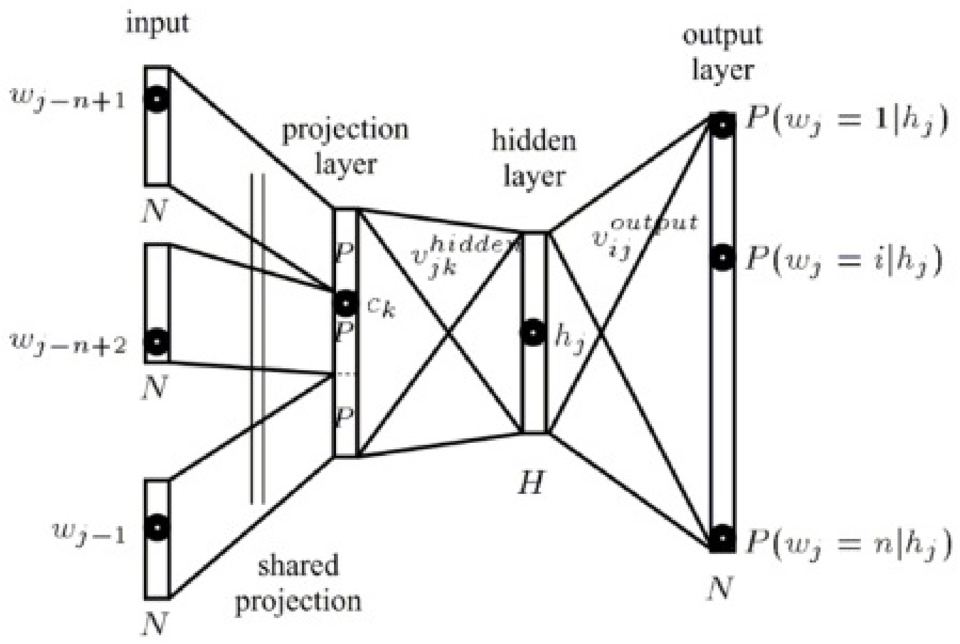


Figure 4.5: Word Embeddings NN

We can observe in 4.5 that h is the output vector of the last hidden layer. Hence, the term $h^\top v'_{w_t}$ represents the probability of the word w_t to be selected normalized by the sum of the probabilities of all the words in the dictionary 4.2. This is because v'_w is a weight matrix whose rows are the v'_w representation of all words w. Therefore, v'_{w_t} is the row-vector that compose the representation of w_t. By doing this we obtain an adaptation of the output vector to a probability distribution over the words in the dictionary. This last step conform which is known as softmax-layer.

Using this layer, the model tries to maximize the probability of predicting the correct word in each time step. So, in an ideal model it would try to maximize the average logarithmic probability of the whole corpus, but in real models it is done with the n previous words:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-1}, w_{t-2}..., w_{t-n+1}) \tag{4.3}$$

There exists different techniques to perform what have just been explained. Some of them are:

- Classic neural language model
- C&W model
- Word2Vec
- GloVe

It is going to be explained Word2Vec because is the technique used by TensorFlow and PyTorch and Classic Neural Language Model because is necessary to understand which are the main problems of using deep learning in word embedding.

4.3.1 Classic Neural Language Model

This model was proposed by Bengio et al. [4] in 2003 and consists of a feed forward neural network with only one hidden layer as we can observe in the next figure.

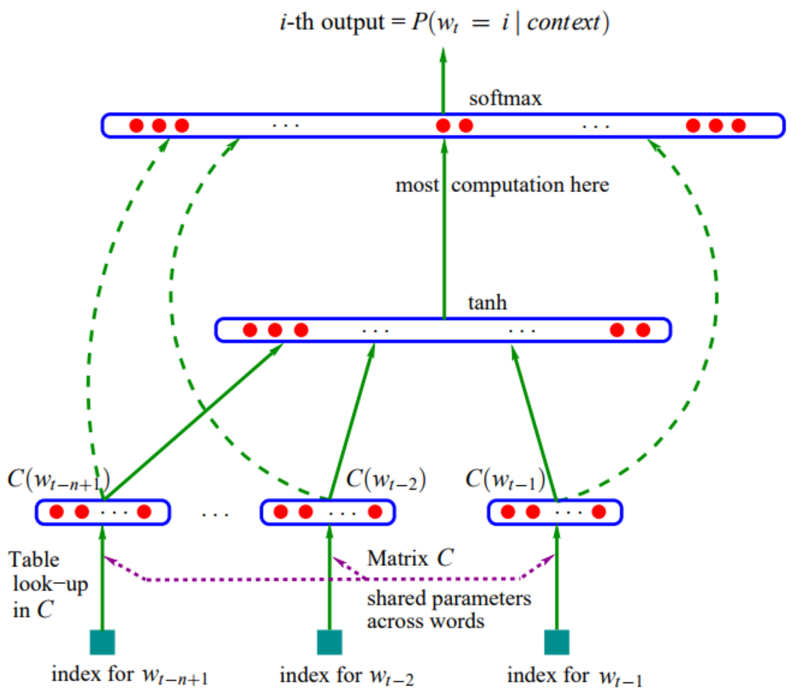


Figure 4.6: Classic neural language model. One hidden layer NN

As it has been explained just before the objective function of this model is a generalization of 4.3:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log f(w_t|w_{t-1}, w_{t-2}..., w_{t-n+1}) \tag{4.4}$$

In order to get that, Bengio et al. propose the next blocks for his model:

- Embedding Layer: This layer generates word embeddings by multiplying an index with a word embedding matrix (bottom part of 4.6)
- Intermediate Layer: It can be one or more producing an internal representation of the input.

- Softmax Layer: final layer that produces a probability distribution over all the words in V.

We can add two observations to those parts of the model. One of them refers to the fact that the intermediate layers can be replace by LSTMs as it is mentioned in [25] and [27]. However, in [25], it is even propose to use CNN instead of LSTM because it generates better results.

Finally, it is important to mention that the softmax layer acts as the bottleneck of the network because the computational cost of computing softmax is proportional to the size of V, that is, proportional to the number of words. Obviously, in experiments as machine translation or conversational models, the number of words has to be huge to increase the coverage of possible answers and translations.

4.3.2 Word2Vec

This is probably the most popular method nowadays. It was proposed by Mikolov et al. [35] at Google in 2013 to make the process of embedding more efficient by eliminating the expensive hidden layer. This is due to two architectures that can be observed in 4.7

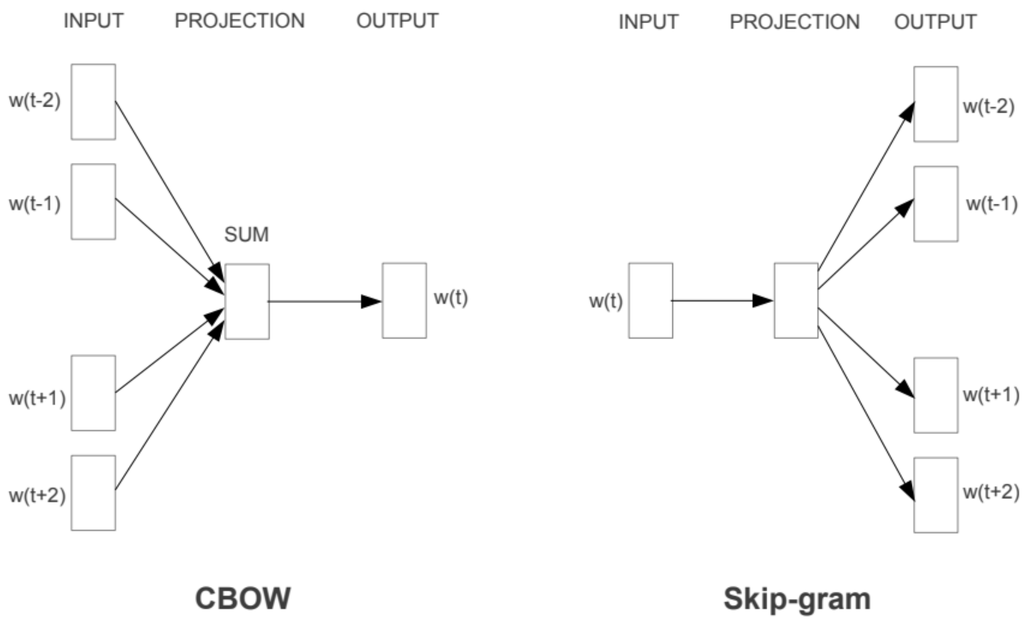


Figure 4.7: Word2Vec architecture

Continuous bag of words(CBOW): This architecture allows the algorithm to predict next word by using the previous n words but also the next n. This nature give them the idea of calling it continuous because its order is not very important. The objective function in this case is slightly different to 4.3 due to the fact that it is taking in count the next n words:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t|w_{t-n}, w_{t-n-1}..., w_{t-1}, w_{t+1}, ...w_{t+n}) \tag{4.5}$$

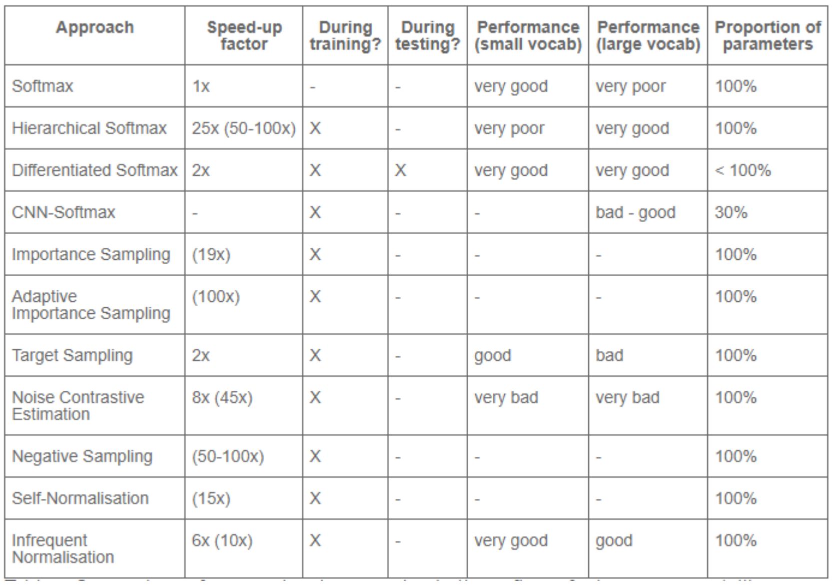
Skip-gram: In this part, instead of using the surrounding words to predict the centre word (CBOW), the algorithm uses the centre word to predict the surrounding words as we can see in 4.7. If we recall 4.2, we can simply modify it to apply it to the surrounding words instead to the previous words. For this purpose, and to follow Mikolov's paper notation, we will rename the centre word as w_I and the surrounding ones as w_O:

$$p(w_O|w_I) = \frac{exp(h^\top v'_{w_O})}{\sum_{w_i \in V} exp(h^\top v'_{w_i})} \tag{4.6}$$

Next step is to eliminate the influence of h, because in this architecture it does not exist hidden layers, and to change it by the embedding in the input:

$$p(w_O|w_I) = \frac{exp(v_{w_t}^\top v'_{w_O})}{\sum_{w_i \in V} exp(v_{w\,I}^\top v'_{w_i})} \tag{4.7}$$

At this point, with this architecture, we have reduce the computational load of the hidden layer. Nevertheless, we continue performing a complete softmax function which is an slow option with a big size of V. There exists many softmax approximations that can be applied and they are group in two categories which are Softmax-based approaches and Sampling-based approaches. Next figure list them and its pros and cons.



Approach	Speed-up factor	During training?	During testing?	Performance (small vocab)	Performance (large vocab)	Proportion of parameters
Softmax	1x	-	-	very good	very poor	100%
Hierarchical Softmax	25x (50-100x)	X	-	very poor	very good	100%
Differentiated Softmax	2x	X	X	very good	very good	< 100%
CNN-Softmax	-	X	-	-	bad - good	30%
Importance Sampling	(19x)	X	-	-	-	100%
Adaptive Importance Sampling	(100x)	X	-	-	-	100%
Target Sampling	2x	X	-	good	bad	100%
Noise Contrastive Estimation	8x (45x)	X	-	very bad	very bad	100%
Negative Sampling	(50-100x)	X	-	-	-	100%
Self-Normalisation	(15x)	X	-	-	-	100%
Infrequent Normalisation	6x (10x)	X	-	very good	good	100%

Figure 4.8: Softmax approximations table

As it can be observed standard softmax gives the worst speed results. However, the complexity of these techniques and the great variety of them are beyond the scope of this project.

4.4 Attention Mechanism

In this point we know that recurrent Neural networks using LSTMs have the ability to work with longer sequences. However, since the LSTM networks appeared, there have been many attempts to augment this networks with new properties, many of them trying to improve its performance with long sequences. Nowadays four directions stand out as particularly exciting:

- Neural Programmers
- Adaptive Computation Time
- Neural Turing Machines
- Attentional Interfaces

This four approaches are very interesting, but in this case only Attentional Interfaces are going to be explained because is the technique used in this project.

The main idea behind this approach is to simulate human and some animals behaviour. When an animal is processing an image it focuses on specific parts of the visual to choose the adequate response. Humans have the same behaviour, even when we are translating a word or an audio recording. We pay more attention to a segment of the audio or to some of the words that we are trying to translate. Therefore, the idea is to get the same on neural networks, putting them to focus on a subset of all the information they are managing.

This process is taken by the attention model:

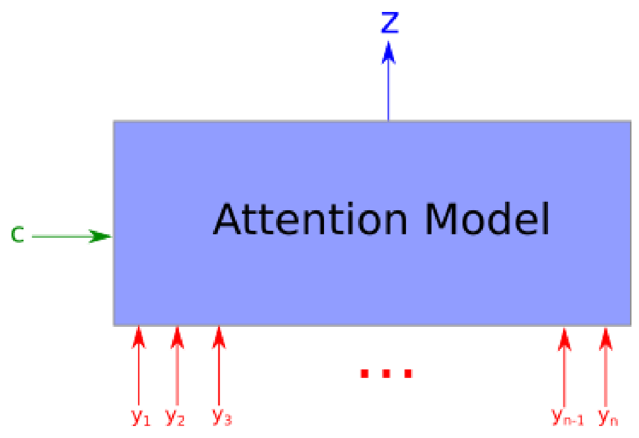


Figure 4.9: Attention Model

It has n arguments as input $(y_1, ..., y_n)$ and a context C. Its output is Z which is an arithmetic mean of the y_i whose weights are chosen according to the relevance of each y_i given the context C.

In a translation example, the inputs would be the h_i hidden states of the different LSTM units processing the input sentence.

Internally, the attention model can be represented as it follows:

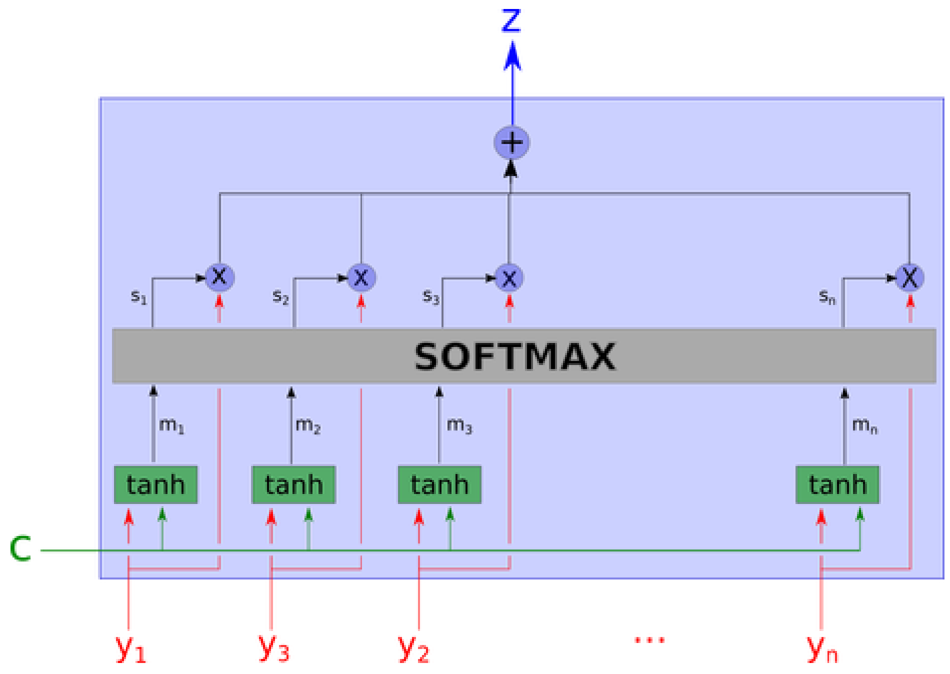


Figure 4.10: Detailed Attention Model

Firs of all we have C which is the context and the inputs y_i (that in our case they are h_i). Each LSTM unit is processing a word of the input sentence and it is generating a hidden state which are those h_i. The context will come from the previous hiddent state in the output layer h'_i (see 4.11)

Then, the model computes m_i using a tanh layer that performs an aggregation of the values y_i with C in an independent way, this is, y_j does not influence in any other of the inputs y_i:

$$m_i = tanh(W_{cm}C + W_{ym}y_i) \tag{4.8}$$

The it computes each the weight of each of the inputs by using a softmax function. The softmax will produce a mapping between the weights and a probability density function. If one of the m_i is much bigger than the rest, the ouput of the softmax will be similar to the output that an argmax function would give, this is an array of 0s whit a 1 in the position of the bigger m. So the softmax can be thought as the max of the relevance of the words according to the context.

Finally, Z is computed as a weighted arithmetic mean of all the y_i [3]:

$$z = \sum_i s_i y_i \tag{4.9}$$

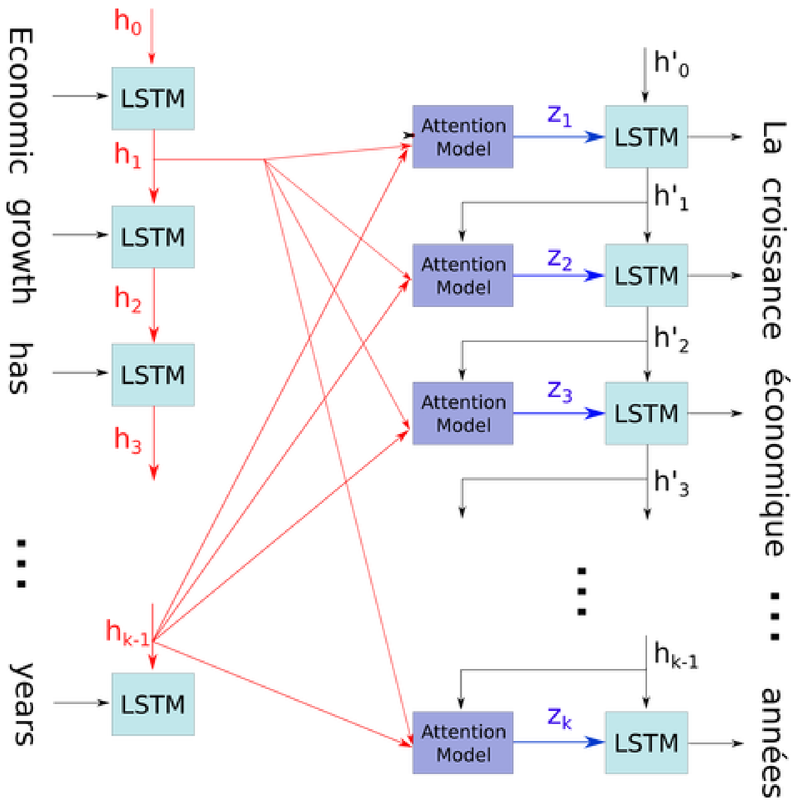


Figure 4.11: LSTM network with attention model

As we can observe in the figure 4.11, each time the decoder produces a word, it determines the contribution of each hidden state 4.9. Usually only the hidden state generated from a single word is the one that is going to influence in the translation.

Finally, this process can be seen as an alignment [3], because the network usually focus in a single word to produce an output word each time step. The image 4.12 shows how the attention focuses in different words with a colour map. The whit squares represent the biggest weights obtained during the generation of the output word.

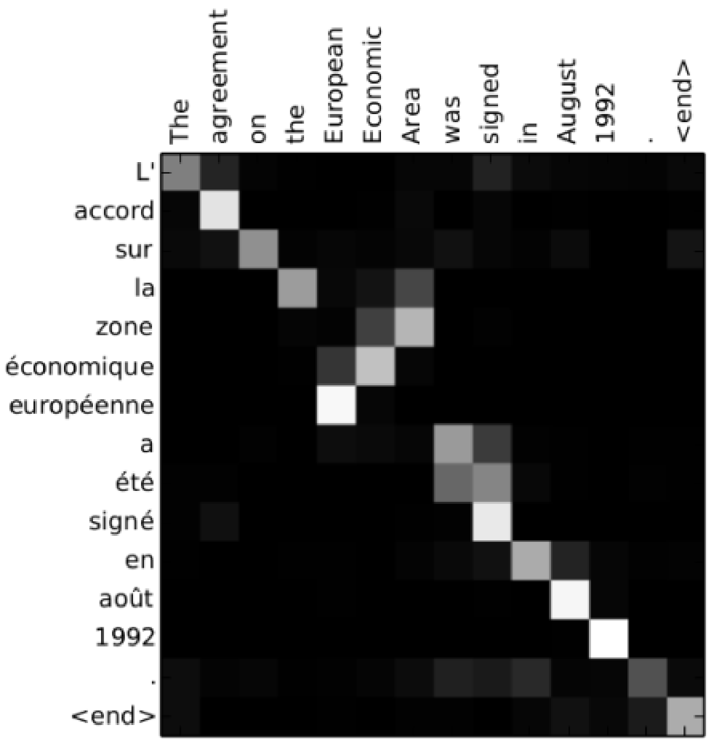


Figure 4.12: Alignment in translation

4.5 From NMT to Conversational Bot

Until this point, in this chapter, we have introduce how neural machine translation algorithm works, explaining the models behind the process and the different techniques uses both for reducing computational costs and improving the results.

It does not exist any model that performs the process of learning to talk as a human. It is clear that, if a neural network has to learn to talk in a language understandable to humans, it has to use NLP. In addition, when neural networks perform machine translation they are not aware they are doing that. This is, they learn how to translate by setting relationships between words and semantic constructions using learnable parameters.

So if we give the appropriate information to the neural network it could learn this "relations" between sentences in the same language, imitating the process of translation between a question and a response instead of between two sentence in different languages.

Therefore, the main idea would consist of feeding the encoder with sentences and provide the network with the responses for this sentences, in order to provide the network with the information it should produce.

CHAPTER 5

The Solution: A conversational bot

5.1 The Data

As it has been introduced in 4.5 the idea is to reuse the NMT algorithm to build a conversational bot. For that purpose, it is necessary a big amount of human conversations to provide the neural network with the enough information to learn.

There are some possible options where to get conversational information between humans, for example Twitter or forums. However, in both cases we have the problem of the orthography of the users. This could seem not very important but abbreviations or malformed words are very common in both environments and it can influence directly in dictionaries and in the process of learning. In addition, is often difficult to follow a conversation in Twitter due to the participation of many people and the mentions. On the other hand, conversations in forums have big user interventions and, many times, talking about so specif topics which would affect the objective of getting open domain conversations.

The third option evaluated was the use of films subtitles. This can constitute a enormous source of conversations with many topics involved and it does not have orthography problems and usually not very extensive interventions. In addition, it is relatively easy to find free subtitles datasets with enough information to train an experiments with these characteristics. All these reasons were decisive to choose this kind of information as base data.

Once it was decided, it was the turn of what kind of subtitles choose and in what language. The biggest free corpus found was in English and apart from the language, there were three options.

- The firs option was OpenSubtitles corpus (Jörg Tiedemann, 2009 [46]). This is an English free subtitles corpus in XML with more than 2 million sentences. It can be found here[1]. It exist a 338 millions english sentences corpus but it is not free.
- The second option was Cornell Movie Dialogs Corpus [18]. It contains 220.579 conversational exchanges between 10.292 pairs of movie characters. It can be found here [17].
- The third option is a variation from the 338 million OpenSubtitles dataset. The variation consist in a subset where following properties were extracted:

  1. First sentence ends with question mark.
  2. Second sentence has no question mark.
  3. Second sentence follows in the movie the first sentence by less than 20 seconds.

  This generates a 14 million sentences corpus with a structure of question and answer. The problem is that there is no certitude that the answers are actually answering the questions, due to those 20 seconds of possible distance between them.

5.2 Preprocessing Data

Once the kind of data has been selected, it must be preprocess in order to get the neural network producing accurate forecasts. This task can be separated in two big process which are transformation and normalization.

Transformation involves manipulating raw data inputs to get single type of input and **normalization** is a transformation of a single data input to distribute the data evenly and scale it into an acceptable range for the network.

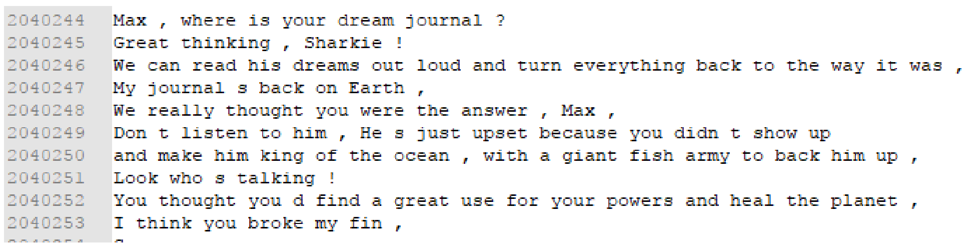
The knowledge of the domain is important in order to know what we want to get for choosing the better way of preprocessing. If it is correctly done, preprocessing can highlight underlying features which could increase the ability of learning associations among input and output data.

In this particular case, the process of transformation consists in deleting all the strange characters from the datasets [14]. When writing, many punctuation symbols are together with words and they give such a subtle nuance of information that they are not useful from a neural network point of view. In addition, many of this characters are completely stuck with words and they difficult the process of creating a vocabulary (*quickly!* and *quickly* are not the same word for the algorithm). Therefore, parentheses, semicolons, brackets, coding characters, hyphens, etc, have to be deleted from the corpus.

In the case of Cornell Dataset and the 14 million sentences OpenSubtilte corpus they are already preprocessed. However is not the case of the 2 million sentences dataset.

This particular dataset comes separated in years and films genre and in XML.The first was to put all the different files together in a file using a simple script.

As it is obvious, due to the XML tags, it was necessary to preprocess all the subtitles to get clear data. For that purpose, it was used a Python script specially coded for deleting all the strange chars. After preprocessing the result was the next:

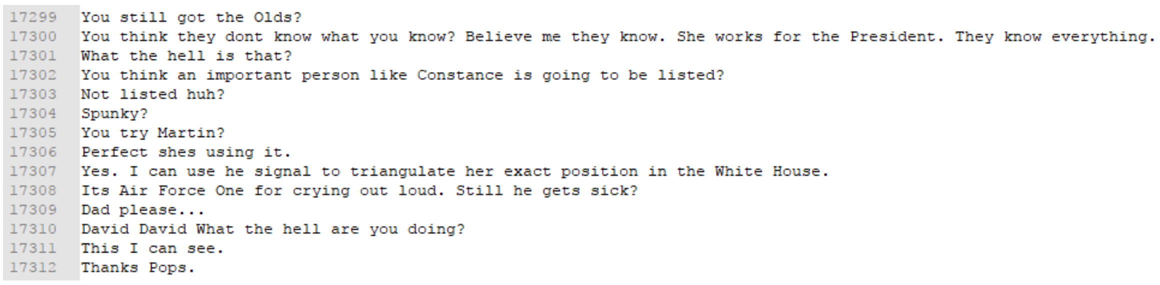


```
Max , where is your dream journal ?
Great thinking , Sharkie !
We can read his dreams out loud and turn everything back to the way it was ,
My journal s back on Earth ,
We really thought you were the answer , Max ,
Don t listen to him , He s just upset because you didn t show up
and make him king of the ocean , with a giant fish army to back him up ,
Look who s talking !
You thought you d find a great use for your powers and heal the planet ,
I think you broke my fin ,
```

Figure 5.1: Preprocessed OpenSubtitles

It can be observed that basic punctuation symbols as commas, stay in the data. This is because basic punctuation, as question marks, are useful to get well formed responses from the neural network. However, these symbols have been separated from the words in order to not affect in the dictionary creation process.

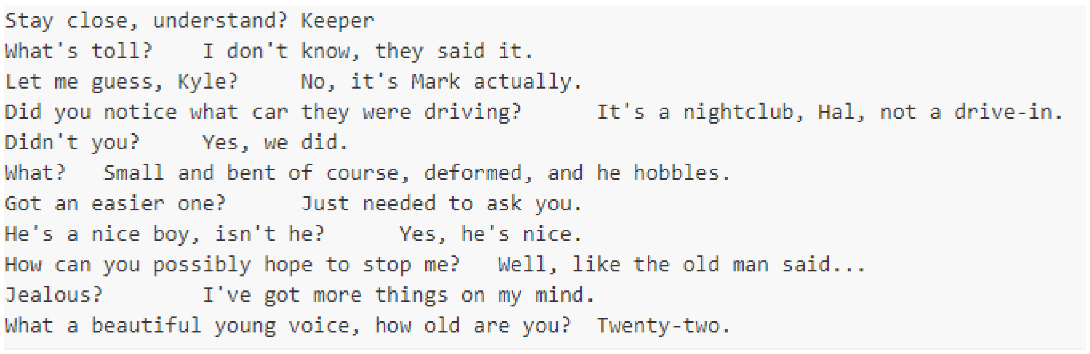
In the case of Cornell, we can appreciate in figure 5.2 that is already preprocessed.



```
You still got the Olds?
You think they dont know what you know? Believe me they know. She works for the President. They know everything.
What the hell is that?
You think an important person like Constance is going to be listed?
Not listed huh?
Spunky?
You try Martin?
Perfect shes using it.
Yes. I can use he signal to triangulate her exact position in the White House.
Its Air Force One for crying out loud. Still he gets sick?
Dad please...
David David What the hell are you doing?
This I can see.
Thanks Pops.
```

Figure 5.2: Cornell Dataset

Finally, in the 14 million sentences dataset, we find the data structured in questions and possible answers on the same line:



```
Stay close, understand? Keeper
What's toll?    I don't know, they said it.
Let me guess, Kyle?     No, it's Mark actually.
Did you notice what car they were driving?      It's a nightclub, Hal, not a drive-in.
Didn't you?     Yes, we did.
What?   Small and bent of course, deformed, and he hobbles.
Got an easier one?      Just needed to ask you.
He's a nice boy, isn't he?      Yes, he's nice.
How can you possibly hope to stop me?   Well, like the old man said...
Jealous?        I've got more things on my mind.
What a beautiful young voice, how old are you?  Twenty-two.
```

Figure 5.3: 14 million OpenSubtitles

With respect to the normalization, in this case that we are dealing with sentences, the process of padding, mentioned on 4.2.2, constitute a normalization process by itself because makes the sentences fit some standard length ranges.

5.2.1 Structuring the data

The NMT algorithm uses 2 data-sets. The one called source and the one called target. Source data-set contains sentences in one language and the target data-set the same but in a language different from the source one. The translation process occurs from the source data-set to the target data-set.

The structure is simple, each line (sentence) of the target data-set is the translation of the same line in the source data-set. So in a case between English and French, if line 1 in the source data-set is "Hello, I am Diego", the line 1 in the target data-set would be "Salut, Je suis Diego".

However, in the case of a conversation is not as clear as in the translation. In this case it have been decide to suppose that there are only 2 interlocutors (A and B) and that the information has a dialogue structure, this is,

$$A \rightarrow B \rightarrow A \rightarrow B \rightarrow ...B$$

. The problem is that there are not certitude that the information of the data-set follows that structure. In film dialogues may participate many interlocutors and they can be answering different things at the same time or maybe one actor has to say more than one sentence in a row. This may influence in the learning process because if the mentioned cases often appear they will generate fake relations between sentences.

One possible solution is to add the name of the character who says each sentence, making the process of identifying the interlocutor easier. Nevertheless, we do not have this information on data-sets so ABA structure has been assumed to be true.

With the aim of getting that structure, 2 million OpenSubtitles and Cornell datasets have been copied and separated in source dataset and target dataset. Then, the first sentence of the target dataset has been deleted in both cases, obtaining with this method the structure mentioned above:

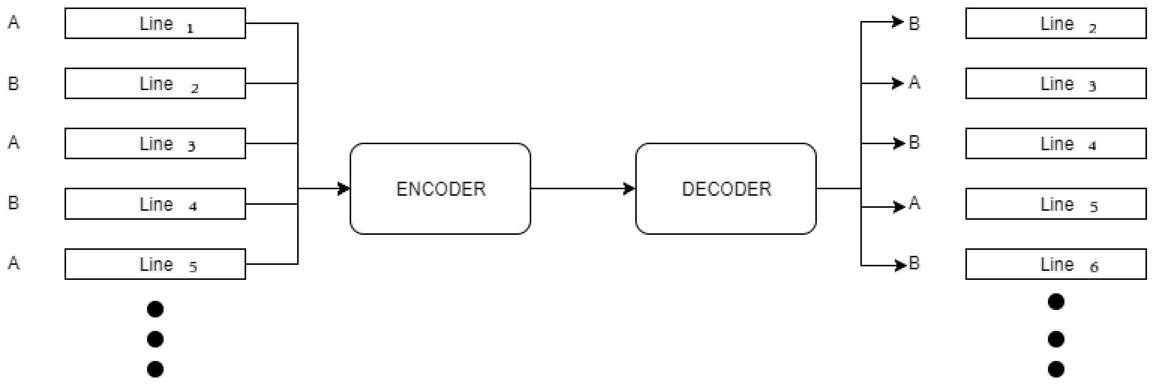


Figure 5.4: Process Scheme

In the case of the 14 million datasets is not necessary to move a row down the target dataset because in the same line we have the question (up to the punctuation mark) and the answers, so is enough to separate the questions in the source dataset and the answers in the target dataset. Because of this reason, this dataset constitutes the candidate option to get the better results. However, the structure of questions-answer has a limitation which is that, after training, is practically impossible that the network could answer one human question with another question. This is due to the fact that the network, changing its learnable parameters is unconsciously establishing relations between inputs and tagged outputs. Therefore, this network will perform a good job of answering questions but maybe not as good job as other possibilities in an open domain conversation. The results of the different datasets will be evaluated later.

After doing this preprocessing we have to divide the sets in three groups as it follows:

- Training Dataset: The sample of data used to fit the model

- Validation Dataset: Smaller sample of data used to provide an evaluation of model fit on the training dataset while tuning the model hyperparameters.

- Test Dataset: Sample of data which is used to provide an ubiased evaluation of model fit on the training dataset.

  The datasets mentioned before (OpenSubtitles, Cornell, etc) constitute the training datasets by itself. But after preprocessing them we can obtain the other datasets by dividing the previous ones into smaller sample sets.

Obviously, it is necessary to maintain the ABAB structure on the question/answer structure. Therefore, the easiest way to obtain those data-sets is by dividing first into source and target, as has been explained before, and then extract two subsets from each of them.

It is important to mention that both the source validation set and the target validation set has to be complementary, this means that there have to be always an "answer" B in target for a "question" A in source, and they have to be of the same size.

5.3 NMT Tensorflow

This is a TensorFlow implementation of a neural machine translation system. It was developed by Thang Luong and Eugene Brevdo, both researchers scientists at Google Brain, and it is include as a part of a bigger project that has the aim of bringing people closer to the neural networks and specially to the NMT systems.

For this purpose they have developed a TensorFlow solution based on seq2seq model 4.2.1 with a well explained code with the latest research ideas and with important notes about the background of NMT. They achieve this goal by:

- Using an attention wrapper API [6] developed in TensorFLow 1.2.
- Providing tips and tricks for building a good NMT model by replicating the Google's NMT system [50]

With the aim of teaching people, they provide two datasets:

- Small example: English-Vietnamese
- Large example: German-English

Once we have done an approximation to the project we are going to explain this implementation more deeply.

5.3.1 Structure

The general project can be divided some modules that have a well defined functionality:

Embedding: This module constitute a layer that needs a vocabulary V to work. It is designed to work with two vocabularies (the source language vocabulary and the target language vocabulary). For that purpose we need to generate two vocabularies with the .src and .tgt extensions to allow the code to distinguish between source and target. In the case of conversational bot, both files have to be identical.

This fact generates a computational problem which is that, despite we have the same vocabulary for source and target, the code is designed for translation so it will generate two words embeddings of the same vocabulary, one for the source and one for the target. This produces an unnecessary increase in the use of memory. The embedding weights 4.3 are usually learned during the training.

```
# Embedding
embedding_encoder = variable_scope.get_variable(
    ‘‘embedding_encoder’’, [src_vocab_size, embedding_size], ...)
# Look up embedding:
#   encoder_inputs: [max_time, batch_size]
#   encoder_emb_inp: [max_time, batch_size, embedding_size]
encoder_emb_inp = embedding_ops.embedding_lookup(
    embedding_encoder, encoder_inputs)
```

We can observe that making little changes we can develop an embedding-decoder. The code gives the option to use pretrained word representations, that can be collect in some forums, by the activation of two flags.

By using the –sos and –eos flags we can specified which are the start of sequence and end of sequence symbols. If we do not specify them <s> and </s> will be used.

Encoder: The embeddings obtained in the previous module are used as inputs in this module which consists of a multi-layer RNN. This RNN can share the weights with the one that conform the decoder, reducing computational costs, but if they have different weights the algorithm will obtain better results.

```
# Build RNN cell
encoder_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)

# Run Dynamic RNN
#   encoder_outputs: [max_time, batch_size, num_units]
#   encoder_state: [batch_size, num_units]
encoder_outputs, encoder_state = tf.nn.dynamic_rnn(
    encoder_cell, encoder_emb_inp,
    sequence_length=source_sequence_length, time_major=True)
```

This example is using a basic LSTM cell but we can change the type by using the flag –unit_type and select other types as GRU. num_units parameter will determine the number of neurons in the encoder layer and will constitute an argument in the creation of the encoder. The batch-size parameter defines the number of samples that are going to be propagated through the network allowing to use mini-batch gradient descent. This means that the network will train with n sentences from the total dataset each iteration.

On the other hand, with the sequence-length parameter we fix a length and with the time-major a shape for the input data.

Decoder: This module is also form by a multi-layer RNN and also have access to the embeddings obtained at the beginning. The encoder will be initialized with the last hidden state of the encoder as it follows:

```
# Build RNN cell
decoder_cell = tf.nn.rnn_cell.BasicLSTMCell(num_units)

helper = tf.contrib.seq2seq.TrainingHelper(
    decoder_emb_inp, decoder_lengths, time_major=True)
# Decoder
decoder = tf.contrib.seq2seq.BasicDecoder(
    decoder_cell, helper, encoder_state,
    output_layer=projection_layer)
# Dynamic decoding
outputs, _ = tf.contrib.seq2seq.dynamic_decode(decoder, ...)
logits = outputs.rnn_output
```

First we specified the kind of the decoder cell and how many of them will constitute the layer. Later, in the decoder initialization we put as argument these units and in the encoder_state the last hidden state of the encoder.

Finally the output layer is the projection layer. This layer is formed by a dense matrix that performs the transformation from the hidden states to logit vectors of dimension V. In these logit vectors each position represents a word of the vocabulary and in this projection layer is where the sofmax- layer explained in 4.5 is computed.

Loss: At this point, the code computes the training loss by using the softmax mentioned above:

```
crossent = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=decoder_outputs, logits=logits)
train_loss = (tf.reduce_sum(crossent * target_weights) /
    batch_size)
```

It uses the cross-entropy method to compute the loss. This is, the softmax classifier is hence minimizing the cross-entropy between the estimated probabilities and the "true" distribution.

The target-weights argument is a zero-one matrix of the same size than the decoder-outputs and has the mission of padding positions with 0. This positions are outside of the fixed length.

Gradient and optimization: This module is the one that computes the backpropagation using for that the training loss computed just before:

```
params = tf.trainable_variables()
gradients = tf.gradients(train_loss, params)
clipped_gradients, _ = tf.clip_by_global_norm(
    gradients, max_gradient_norm)
```

As we can observe, the function “gradients” will try to reduce the value of the parameter “train_loss” by modifying the “params”. By clipping the gradients we are limiting the values they can obtain and preventing them to get too large. With this technique we avoid the problem of exploding gradients mentioned in 3.4.4

Finally we have to choose which optimizer to use. As it was explained in 3.4.2 there are many options but a common choice is the Adam optimizer:

```
optimizer = tf.train.AdamOptimizer(learning_rate)
update_step = optimizer.apply_gradients(
    zip(clipped_gradients, params))
```

Usually the values of the “learning_rate” parameter are between 0.0001 and 0.001. This parameter, as an intuitive explanation, determines how quickly the network abandons old beliefs for new ones. If it is very small the network will think it is an outlier when a strange sample comes and if it is big the network will change its “mind” quickly and it can star thinking that strange samples are the right ones.

Attention Wrapper: This wrapper is based on the work on memory networks by Weston et al.,2015 [48]. In this particular case, the wrapper use the current target hidden state as a “query” to decide which is the part of the memory it should read. For this reason this is considered as a read-only memory:

```
# attention_states: [batch_size, max_time, num_units]
attention_states = tf.transpose(encoder_outputs, [1, 0, 2])

attention_mechanism = tf.contrib.seq2seq.LuongAttention(
    num_units, attention_states,
    memory_sequence_length=source_sequence_length)
```

As we can observe, in the attention_states we introduce the encoder_outputs which are the set of all source hidden states at the top layer. Defining the attention_mechanism we pass the attention_states and the source_sequence_length. This last parameter ensures that the attention weights will be properly normalized. This wrapper will be used to define de kind of the decoder cell as we have seen above:

```
decoder_cell = tf.contrib.seq2seq.AttentionWrapper(
    decoder_cell, attention_mechanism,
    attention_layer_size=num_units)
```

By doing this we have not add any new module but we have add a the attention functionality to the decoder cells.

5.3.2 Training TF-NMT

This NMT system needs to be feed with different datasets depending on which is its destination, the source part or for the target part.

We have to divide the datasets mentioned in 5.1 as it was explained on 5.2.1:

- Traininig Dataset
- Validation Dataset
- Test Dataset

In order to make the NMT train it is necessary to select a minimum number of parameters as we can see below:

```
python -m nmt.nmt \
    --attention=scaled_luong \
    --src=vi --tgt=en \
    --vocab_prefix=/tmp/nmt_data/vocab  \
    --train_prefix=/tmp/nmt_data/train \
    --dev_prefix=/tmp/nmt_data/tst2012  \
    --test_prefix=/tmp/nmt_data/tst2013 \
    --out_dir=/tmp/nmt_attention_model \
    --num_train_steps=12000 \
    --steps_per_stats=100 \
    --num_layers=2 \
    --num_units=128
```

As we can see there is a difference between the source datasets and the target ones. They can be called in the same way but they have to have a different suffix (src and tgt).

There is also another flag to indicate the path of the vocab, which can be obtained by using multiple methods. In this case, there has been coded a solution using one list and a dictionary, the list for checking the appearance of words and the dictionary for counting the number of appearances. So the code go through every word in the text checking if it is in the checking list. If it is, the code adds a unit to the value of that word that is working as key at the same time. After finishing the text, the dictionary is sorted and the X words which more appearance values are selected to compose the dictionary.

In this case the validation data is include in the dev_prefix flag.

The number of layer of both the encoder and decoder is in the flag num_layers and the number of neurons by layer in the num_units flag. It is necessary taking in count the available resources we have to perform an experiment in order to select the number of this two flags. These kind of networks are fully connected and increasing too much the number of layers or units may produce a huge usage of memory.

The training command showed in the box above is only one combination of the multiples commands that can be used. In the class **nmt.py** we can find a huge amount of parameters that can be used allowing to choose from the kind of optimizer (–optimizer) to the number of buckets for bucketing (–num_buckets). This big amount of parameters gives a huge flexibility to the implementation allowing us to reproduces nearly infinite experiments only limited by hardware and time.

5.3.3 Inference. Generating conversations

Once the NMT model have been trained, we can obtain answers to previously unseen source sentences. For that nature, this process is known as inference (testing) and it is not the same that training. In this case the code only has access to the source sentence (encoder_inputs)[34]:

1. The source sentence is encoded as it would be encoded during the training process. This produce a hidden state that in the last part of the encoder it is known as encoder state and it is used as input of the decoder part.

2. The process of decoding starts when the decoder receives the symbol <s> which means the end of the sequence. This symbol, as explained before, can be selected with the use of some configuration parameters (–tgt_sos_id)

3. Then, as in the process of training, the decoder output is treated as a set of logits, which allows to decide which word is more probable (the one with the higher logit value in each step).

4. This process continues until the decoder generates the </s> which would be the one with the higher logit value as an answer to <s> as we can see in the following process in the image below:

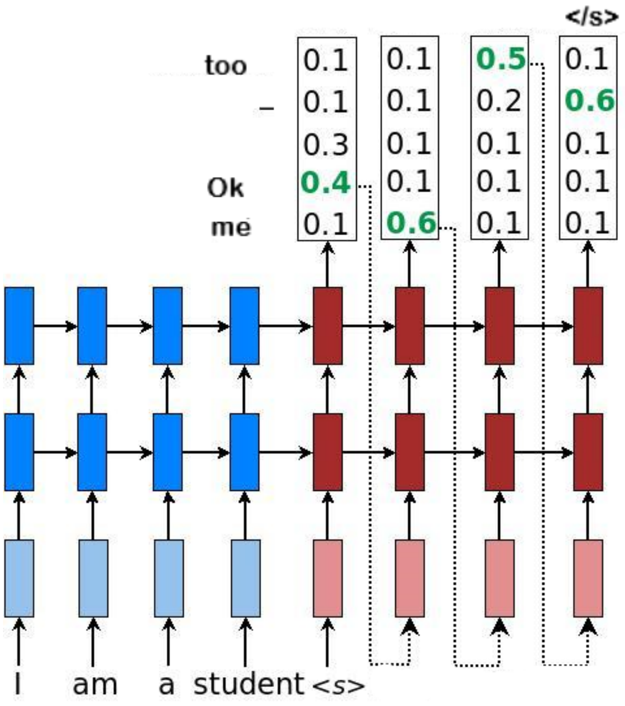


Figure 5.5: Greedy Decoder

We can observe that the generated response to the input "I am a student" will be "Ok me too".

The main difference with the training process occurs during step 3 because the decoder is not fed with the correct target word in each step but with the previous generated word predicted by the model.

In the case of infer process the code of the decoder will be slightly different to the training decoder:

```
helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(
    embedding_decoder,
    tf.fill([batch_size], tgt_sos_id), tgt_eos_id)

decoder = tf.contrib.seq2seq.BasicDecoder(
    decoder_cell, helper, encoder_state,
    output_layer=projection_layer)

outputs, _ = tf.contrib.seq2seq.dynamic_decode(
    decoder, maximum_iterations=maximum_iterations)
translations = outputs.sample_id
```

The main difference is in the helper, which is a GreedyEmbeddingHelper instead of a TrainingHelper. This is becase, as it has been just mentioned, the decoder in this process does not have the target word as input in each time step.

Once the model is trained, if we want to test how good the answers are we can obtain answer by passing the code an inference file and a path where to save the outputs:

```
python -m nmt.nmt \
    --out_dir=/tmp/nmt_model \
    --inference_input_file=/tmp/my_infer_file.vi \
    --inference_output_file=/tmp/nmt_model/output_infer
```

The content of the infer file can be copied from one of the test datasets. It is important to mention that the code saves checkpoints when it finishes an epoch of training. Each checkpoint has a copy of the learnable parameters at that point allowing to perform inference processes during training.

5.4 Open NMT

Open NMT is an open source project for neural machine translation and neural sequence modeling. It was launched in December 2016 and has become a collection of many implementations oriented both in academical aspects and industry aspects. It was originally developed by Yoon Kim and harvardnlp team [29].

There are different implementations of the seq2seq model in different languages as Lua, Torch or Pytorch. In this case PyTorch have been finally selected due to a problem with the TensorFlow implementation that will be explained in 6.2.1

5.4.1 Structure

The structure of this framework is slightly different with respect to the TensorFlow one.

Core Modules: This general module works as an interface and provide tools for generate embedding for the encoder and the decoder. This tools also include some features that allow to improve the process of learning as the ones we can find on the article "Linguistic Input features Improve Neural Machine Translation" [44].

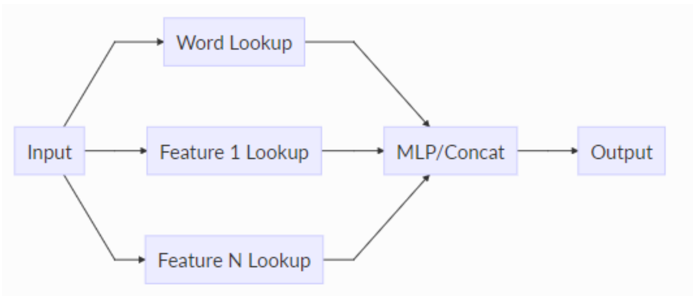


Figure 5.6: Embeddings Open NMT

It allows to set many configuration parameters as the size of the dictionary of embeddings, the size of the vocabulary, the use of pretrained vectors, etc.

In this implementation the code is able to generate its own vocabulary from the datasets thanks to the class preprocess.py

```
def build_save_vocab(train_dataset, opt):
    fields = onmt.io.build_vocab(train_dataset, opt.data_type,
                                 opt.share_vocab,
                                 opt.src_vocab_size,
                                 opt.src_words_min_frequency,
                                 opt.tgt_vocab_size,
                                 opt.tgt_words_min_frequency)
```

This class also makes a preprocess to the datasets, so with the objective of standardizing the task of preprocessing it has been code an small bash script that must be execute before the training. This script is going to automatize the process of giving the paths of the different datasets to the shell.

```
python preprocess.py \
  -train_src /workspace/data/utf_train.enc \
  -train_tgt /workspace/data/utf_train.dec \
  -valid_src /workspace/data/utf_dev.enc \
  -valid_tgt /workspace/data/utf_dev.dec \
  -save_data /workspace/data/bot
```

Model: It is a trainable object that implements a trainable interface to declare both the encoder and decoder. Specifically, it can be found in the class **onm.Models.NMTModel** together with the class **DecoderState** which an interface that groups the decoder states (hidden states) which is necessary for using beam search.

In the NMTModel class it can be found different schemes of encoders and decoder, include recurrent and no recurrent models. In the case of encoders interface is the **EncoderBase** that provides the next common scheme:

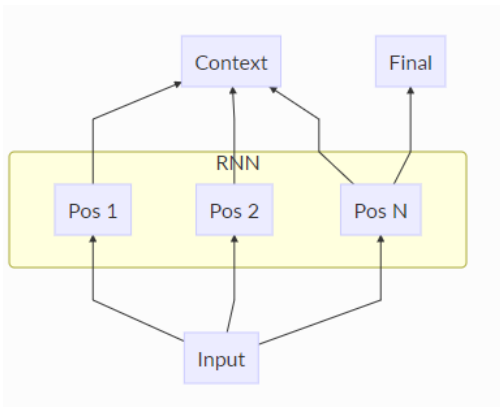


Figure 5.7: Encoder Open NMT

The RNN encoder using RNN units can be found in a class call **RNNEncoder** and it gives the possibility of modifying parameters as the type of unit (LSTM, GRU, SRU), the bidirectionally of the encoder, the number of layers, etc.

```
class RNNEncoder(EncoderBase):
def __init__(self, rnn_type, bidirectional, num_layers,
                 hidden_size, dropout=0.0, embeddings=None):
        super(RNNEncoder, self).__init__()

        ...

self.rnn = getattr(nn, rnn_type)(
                        input_size=embeddings.embedding_size,
                        hidden_size=hidden_size,
                        num_layers=num_layers,
                        dropout=dropout,
                        bidirectional=bidirectional)
```

In the case of the decoder the structure is similar to the encoder, but the interface gives more complexity as can be observed in the next figure:

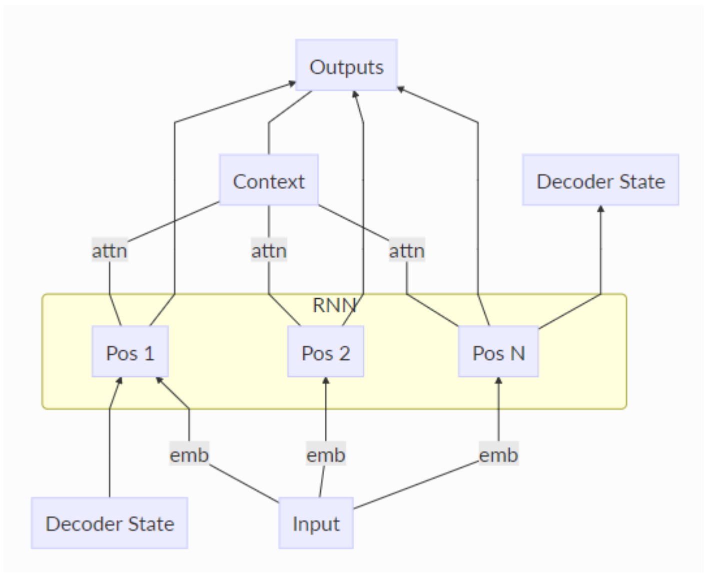


Figure 5.8: Decoder Open NMT

There are two RNN implementations but the useful one in this experiment is the **StdRNNDecoder** that is a standar fully batched RNN decoder with attention. It uses CuDNN and is based on the approach from "Neural Machine Translation By Jointly Learning To Align and Translate" [3].

```
class StdRNNDecoder(RNNDecoderBase):
def _run_forward_pass(self, input, context, state, context_lengths=None):
        ...
    rnn_output, hidden = self.rnn(emb, state.hidden)
    # Result Check
    input_len, input_batch, _ = input.size()
    output_len, output_batch, _ = rnn_output.size()
    aeq(input_len, output_len)
    aeq(input_batch, output_batch)
    # END Result Check
    ...
    # Calculate the attention.
    attn_outputs, attn_scores = self.attn(
        rnn_output.transpose(0, 1).contiguous(),
        context.transpose(0, 1),
        context_lengths=context_lengths
        )
```

In the case of the **Attention** it is a global attention by default and it uses a query vector. It is performed in a similar way to what was explained in 4.4, this is, it constitutes a unit mapping a query q of size dim and a source matrix H of size $n * dim$, to an output of size dim

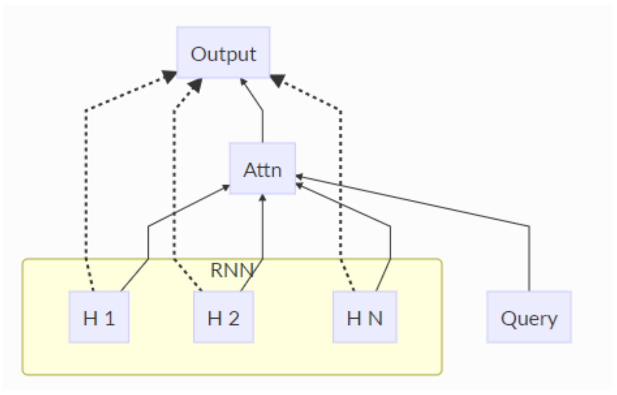


Figure 5.9: Attention Open NMT Scheme

Independently of the model used, all of them computes the output as:

$$c = \sum_{j=1}^{SeqLength} a_j H_j \tag{5.1}$$

The term a_j represents the softmax of a score function and then it is applied a projection layer to $[q, c]$. However, the models can compute the attention in a different way by changing some configuration parameters (–global_attention). Selecting dot/general or mlp will produce a different way of computing the attention score:

- **Luong Attention**
  - dot: $score(H_j, q) = H_j^T q$
  - general:$score(H_j, q) = H_j^T W_a q$
- Bahdanau Attention
  - mlp: $score(H_j, q) = v_a^T tanh(W_a q + U_a h_j)$

In addition, it also incorporates the **structured attention** which consist of an implementation of the matrix-tree theorem for computing marginals of non-projective dependency parsing. This is explore in the paper "Learning Structured Text Representations" [33]. In the next figure we can observe a non-projective dependency:

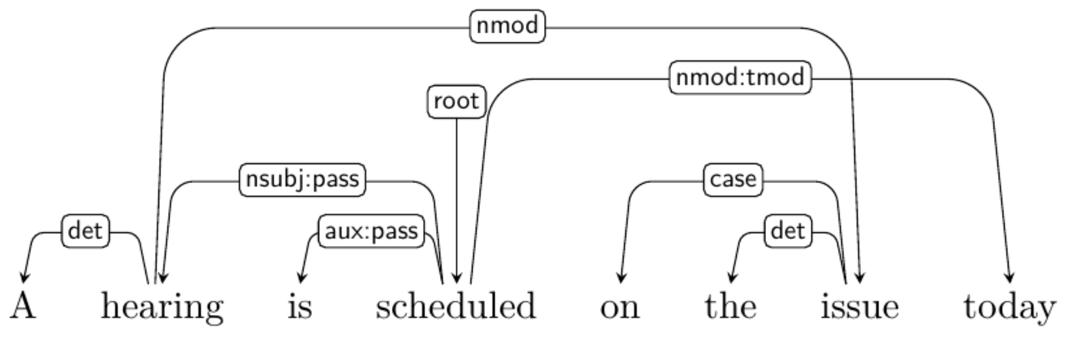


Figure 5.10: Non-projective Example [38]

This kind of relations allows the attention mechanism to focus more on some parts, allowing to improve the learning of the relations between words.

Loss: Similarly as in the TensorFlow implementation, this class gives an interface to compute the loss function by computing multiple loss computations as the monolithic loss (the forward loss for the batch) and the sharded loss which uses a technique of making shards of the state dictionary a relieve memory required for generation buffers. Users can also implement their own loss computation strategies by making subclasses.

```
...
def sharded_compute_loss(self, batch, output, attns,
                             cur_trunc, trunc_size, shard_size):

     batch_stats = onmt.Statistics()
        range_ = (cur_trunc, cur_trunc + trunc_size)
        shard_state = self._make_shard_state(batch, output, range_, attns
           )

        for shard in shards(shard_state, shard_size):
            loss, stats = self._compute_loss(batch, **shard)
            loss.div(batch.batch_size).backward()
            batch_stats.update(stats)

        return batch_stats
...
```

Optim: This is the controller class for optimization. It constitutes a thin wrapper that implements the necessary methods for training RNNs such as grad manipulations. By modifying some parameters, this class allows to change the optimization method (–optim) and the update of learning rate.

```
def update_learning_rate(self, ppl, epoch):
...

 if self.start_decay_at is not None and epoch >= self.start_decay_at:
            self.start_decay = True
        if self.last_ppl is not None and ppl > self.last_ppl:
            self.start_decay = True

        if self.start_decay:
            self.lr = self.lr * self.lr_decay
            print(''Decaying learning rate to %g'' % self.lr)

        self.last_ppl = ppl
        self.optimizer.param_groups[0]['lr'] = self.lr
```

5.4.2 Training Open NMT

The commands necessary to start the training are very similar to the example of TensorFlow. In fact, a big part of this implementation served as inspiration for the devlopment of the TensorFlow's one.

In this case, with this implementation have been trained many more examples, so it has been coded a bash script [19] in order to simplify the task of start the training process:

```
python train.py \
  -data ../data/bot \
  -save_model ../data/model/bot-model \
  -layers 2 \
  -rnn_size 256 \
  -word_vec_size 256 \
  -gpuid 0
```

These are the basic commands to start to train and their names give enough information to understand their meaning. This implementation, by using the preprocess.sh script mentioned before, generates some PyTorch arrays and a vocabulary and save them into save_data directory. Later, with the train.sh script it uses this preprocessed data. Is in this script where the different options (otimizers, attention mechanisims, number of layers, etc) have been selected.

On the other hand, the -gpuid parameters allows to indicate which gpu it can use and it is useful in the case of multiple gpus.

5.4.3 Inference. Generating Conversations

In this case has been also coded a script for testing as in the previous section 5.4.2. In this case the name of the script is infer.sh and its content is as follows:

```
python translate.py \
  -model $1 \
  -src ../data/utf_test.enc \
  -output ../data/preds.txt \
  -replace_unk \
  -verbose
```

During the training process, the NMT generates models that are saved into the path selected in -saved_model. These models are saved when each epoch finishes and they have all the learnable parameters as they were in that moment. They have in their names the perplexity value, so we select the best one as the argument for the train.sh script. We also need to feed the inference method with some test sentences.

After executing this script the NMT will generate a response for each of those test questions replacing the <unk> tags.

5.5 Metrics

Both implementations uses the same metrics in order to calculate how good are their predictions in comparison with the expected results. There are different kinds of metrics but two of them are the most popular in conversational generators world. These are perplexity and bleu.

5.5.1 Perplexity

The perplexity term, in information theory, refers to a measurement of how good is a probability model predicting a new sample.

The perplexity is defined as $2^{H(p)}$ where $H(p)$ is the entropy of the distribution $p(X)$ over all $x \in X$:

$$H(p) = -\sum_{x} p(x) log_2 p(x) \tag{5.2}$$

In the case of determining the perplexity of a model it is used $B(q)$ instead of $H(p)$, which is an estimate of the average power of the model $q(X)$ to produce a representation of a set of $x \in X$. In this case it would be as follows:

$$B(q) = -\frac{1}{N}\sum_{i=1}^{N} log_2 q(x_i) \tag{5.3}$$

In this particular case, N represents the number of samples which is the total number of words as the implementation works with unigrams. Intuitively, assuming $2^{B(q)}$ (it could be also $eB(q)$ depending on the example),we can observe the next extreme cases:

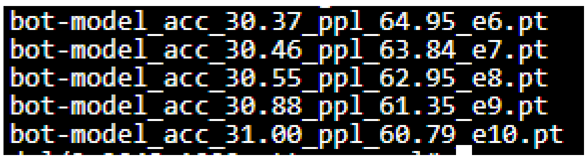
- The minimum value would be when $q(x_i = 1$ for all i. This means that each sample is predicted with the highest confidence and this make the perplexity equals to 1 due to 2^0.
- The maximum value would be when $q(x_i = 0$ for all i. This means that each sample is predicted with the lowest confidence, making the perplexity $2^{+\infty} = +\infty$

However, the last formula assumes that each sample appears only one time in the N test samples considered. Taking in count that in our case a word could appear in sentence more than one time it is necessary to adapt this expression:

$$B(q) = -\sum_{i=1}^{N} \tilde{p}(x) log_2 q(x_i) \tag{5.4}$$

$$\tilde{p}(x) = \frac{n}{N} \tag{5.5}$$

The next figure correspond to checkpoints of a training and we can observe how the score of perplexity is decreasing each epoch:



```
bot-model_acc_30.37_ppl_64.95_e6.pt
bot-model_acc_30.46_ppl_63.84_e7.pt
bot-model_acc_30.55_ppl_62.95_e8.pt
bot-model_acc_30.88_ppl_61.35_e9.pt
bot-model_acc_31.00_ppl_60.79_e10.pt
```

Figure 5.11: Perplexity Obtained by Epoch

The decreasing of the perplexity also decrease with each epoch. This is due to the fact that in the first epochs of the training is when the RNN is improving faster.

5.5.2 Bleu

Bleu is an algorithm specially designed to evaluate the quality of a text which it has been generated by machine translation between two languages ("the closer a machine translation is to a professional human translation, the better it is" [40])

For each word in the candidate translation, bleu takes the maximum number of appearances in any of the reference translations (m_{max}). Then, it counts m_w which is the number of times that the questioned word appears in the candidate sentence. This value is clipped to m_{max} and this will be the new value of m_w. Later, these clipped counts are summed over all distinct words in the candidate. This sum is then divided by the total number of words in the candidate translation.

In practice, instead of words, n-grams are used because they obtain better results when human evaluates them. The higher correlation was found to be four-gram [13].

However, in the case of generating conversational models, bleu seems to work worse than perplexity and almost all experiments and papers about this topic use perplexity as the main metric.

5.6 Results: Conversations

5.6.1 Different Trainings

It have been performed many kind of trainings trying always to reach the best results possible with the available resources.

We can classify them depending on the source data used:

- Cornell trainings: 3 small trainings with small topologies.
- Open Subtitles trainings: 2 medium trainings with medium size topologies.
- 5 million OS question/answer trainings: 7 big trainings with medium a large size topologies.
- 14 million OS question/answer training: Biggest training with large size topologies.

All the prediction files can be found in [15]

5.6.1.1 Cornell Trainings

In this processes of training it has been used the Cornell dataset whose characteristics were mentioned on 5.1.

The main parameters modification were:

1. Training 1: 1 layer encoder, 1 layer decoder, 128 units per layer and 128 word embeddings.
2. Training 2: 1 layer encoder, 1 layer decoder, 512 units per layer and 128 word embeddigns.
3. Training 3: 1 layer encoder, 1 layer decoder, 512 units per layer and 256 word embeddings.

These trainings were the first and they have the objective of show how an increase in the number of units and in the number of word embeddings leads to better results.

Obviously, the best perplexity, was achieved in traing 3 in epoch 13 with a value of 119.9. In the first training it was 150.69, and in the second 123.84, both also in epoch 13. We can observe how the increase in the number of units achieves significant better score. On the other hand, the increase in the number of word embeddings also leads to a better result but the difference is not as significant as between traning 1 and 2.

However, a perplexity of 119.9 is not very good and responses have many <unk> tags and many “I don’t know” responses. In this case, the <unk> tags are due more to the fact that the topologies of the networks are small and they have a lower learning capacity.

Below we can observe some of the best answers of each of the trainings:

Training 1:

```
SENT 21: Nothing. Good night.
PRED 21: Good night.

SENT 175:  Wait wait wait. Hold up. Hold up. Look.
PRED 175:  What?

SENT 75:  Hi honey.
PRED 75:  Hi.
```

Training 2:

```
SENT 23:  Doing what?
PRED 23:  I don't know.

SENT 79:  Who is she Daddy?
PRED 79:  I don't know.

SENT 113:  You ok?
PRED 113:  Yeah.
```

Training 3:

```
SENT 6: Where did he keep his money?
PRED 6: I don't know.

SENT 28: Can you give me one good reason why I should?
PRED 28: No.

SENT 46: Really?
PRED 46: Yeah.
```

As we can observe, responses (PRED) are very poor. Sometimes, they are good answers for the particular question but in general the “I don’t know problem” mentioned in 6.2.2 is very common due to the small topology of the network.

On the other hand, despite the perplexity is better in training 3, the responses are not much better from a human point of view.

5.6.1.2 Open Subtitles trainings

In these particular case, the objective of these 2 trainings were to show the same intuition than in 5.6.1.1 but using a bigger dataset in order to obtain better results. The characteristics of this dataset have been also mentioned in 5.1

The main parameters modification were:

1. Training 1: 2 layers encoder/decoder, 500 units per layer and 500 word embeddings.
2. Training 2: 2 layers encoder/decoder, 1024 units per layer and 1000 word embeddigns.

The perplexity obtained by the first training was 58.17 in the epoch 13. The second training achieved a perplexity of 34.89 in the 13 epoch. We can observe that these perplexity scores are much better than in 5.6.1.1. In these case it is because both dataset and topologies are much larger so the networks have the capability of learning more and better. Here, we can also observe that increase number of units and number of word embeddings leads to better results, but it this case the difference between both perplexities is bigger than in the Cornell cases. The explanation is that we are increasing the number of units by a factor of 2 but there are also the double of layer so the connections in a fully connected topology are more than the double. In addition, the combinations of word embeddings are exponential so a difference between 500 and 1000 offers much more possibilities than between 128 and 256.

Below we can observe some of the best answers of each of the trainings:

Training 1:

```
SENT 1:  Jesus .
PRED 1:  What ?

SENT 25:  Perhaps under the circumstances , I should be calling you
    Captain .
PRED 25:  Yes , sir .

SENT 195:  The extra bedrooms were for the children , which were never
    used .
PRED 195:  It was a long time ago .
```

```
SENT 2151:  After three months of trying , Pam said it wasn't working and
     they were gonna get a donor from a sperm bank .
PRED 2151:  She didn't even know that .

SENT 2193:  Go play dominoes .
PRED 2193:  Go !

SENT 35457:  is the little death that brings total obliteration . Flesh
    dropping off . Fear is the little death ! I must not fear .
PRED 35457:  I will kill you .

SENT 50739:  I il come later
PRED 50739:  I il see you later
```

In the next block we can observe two continue conversatiosn that have some sense.

```
SENT 2205:  He s happy , they re happy , we re happy .
PRED 2205:  We re happy .

SENT 2207:  Just leave it alone , all right ?
PRED 2207:  Okay .

SENT 2209:  Just leave it alone .
PRED 2209:  I m sorry .

SENT 2211:  I m serious .
PRED 2211:  I m sorry .
____________________________________________________________


SENT 35309:  who have long held a prophecy that a man would come ,
PRED 35309:  and a man who would not be a man .

SENT 35311:  a messiah ,
PRED 35311:  a leader of the world .
```

Training 2:

```
SENT 4001:  See you next week , Mrs. <unk> .
PRED 4001:  Thank you .

SENT 4033:  Wanna get a drink ?
PRED 4033:  Sure .

SENT 4057:  Door Closes
PRED 4057:  Door Opens

SENT 4269:  I merely believe in the law .
```

```
PRED 4269:  I believe in you .

SENT 4411:  Why don t you hold that on
PRED 4411:  Cause I love you

SENT 4505:  He says San Francisco is like an anthill at the foot of a
    bridge .
PRED 4505:  He says there is no way out of here .

SENT 4551:  Like jumping into a fountain in Rome .
PRED 4551:  That's why we re here .

SENT 6091:  Three boys , a girl , four will go .
PRED 6091:  One , two , three .
```

In example "See you next week , Mrs. <unk>." , we can observe the problem of not having a word in the dictionary. Probably the name did not appear enough times to be in the dictionary and the network does not know it.

We can also observe how it has learnt the relations in the example of the numbers. It receives numbers and "thinks" that an answer with numbers could be ok.

In general we can observe that using a bigger dataset with a larger topologies produces better answers.

5.6.1.3 5 million OS question/answer trainings

This dataset has been the base of the biggest number of trainings. They have been performed 7 types of training:

1. Training 1: 2 layers encoder/decoder, 512 units per layer and 500 word embeddings. Best perplexity = 88.63

2. Training 2: 2 layers encoder/decoder, 2048 units per layer, 1000 word embeddigns and general attention . Best perplexity = 60.41

3. Training 3: 2 layers encoder/decoder, 2048 units per layer, 1000 word embeddigns and dot attention. Best perplexity = 60.45

4. Training 4: 2 layers encoder/decoder, 2048 units per layer, 1000 word embeddigns and mlp attention. Best perplexity = 60.46

5. Training 5: 3 layers encoder/decoder, 512 units per layer, 1000 word embeddigns. Best perplexity = 63.68

6. Trainign 6: 3 layers encoder/decoder, 512 units per layer, 1000 word embeddigns and bidirectional. Best perplexity = 63.68

7. Training 7: 3 layers encoder/decoder, 1024 units per layer, 1000 word embeddigns and bidirectional. Best perplexity = 57.57

In this case, the general size of the trainings is bigger than in 5.6.1.2. These trainings have the objective of showing the influence of other training parameter as attention or bidirectionality.

In general we can observe that the kind of attention does not influence almost anything because using any of the three options leads to almost same results.

In the case of 3 layers examples we can confirm the intuition mentioned before which is that the parameters whit more influence in the result are the number of layers and units and the relation between them.

With respect to the values of the perplexities, they are bigger than in 5.6.1.2 due to two reasons:

- The main one is that this dataset is preprocessed in order to have only questions and answers. In addition many of "I don't know" type answers have been deleted. Hence, the network is not going to learn those kind of answers as easy as in the previous experiments. That produces a more variety of responses but this makes the perplexity to increase because the answers are going to be more different between them. On the other hand, deleting those kind of easy answers, produces that the network has some tendency to answer using part of the question.

- As the second reason, in those trainings the learning rate has been changed in each of them trying to avoid "I don't know" answers and answers using parts of the question.

One possible explanation to the problem of answering with part of the question could be the next:

If we imagine a simple question as "how are you?<eos>", in the moment of generate the first word the information that the network uses is the tag of end of sequence, the internal representation of the question and the attention values. We can consider that the tag is not giving any information so the result will be generated using the internal representation and the attention values. So if the answer begins with the word "how" this could be because the attention mechanism is focusing in the first word trying to "understand" what the question is about. It has also the information coming from the internal representation but maybe it is dominated by the attention mechanism. So, perhaps, this problem comes from a bad combination of the internal information and the attention information. But it is just a guess and to prove it could take another entire project.

In this case we are not going to see any example because they are going to be shown in the last training which is based on a larger version of the same dataset.

5.6.1.4 14 million OS question/answer training

In this particular case only one training has been done due to the huge amount of time necessary for training (more than 3 weeks).

The parameters used to train were: 2 layers encoder/decoder, 2048 units, 1000 word embeddings, bidirectional,

Obviously this is the example with the smallest perplexity and the one that gives better answers. Let's see some of them:

```
SENT 5:  Can you just let me out, man?
PRED 5:  I need to talk to you.

SENT 89:  And why don't you think I can find a woman for myself?
PRED 89:  Because you're a woman.

SENT 343:  What does that mean?
PRED 343:  It means I don't want to go to jail.

SENT 451:  So you know who I am?
PRED 451:  I know who you are.

SENT 841:  Nikki, what happened to my relaxing Eastern Bell music?
PRED 841:  I got it right here.

SENT 981:  What was your duty as criminal investigator?
PRED 981:  To arrest criminals

SENT 1085:  Why are you scolding him unnecessarily?
PRED 1085:  He is lying to you.

SENT 1175:  You're afraid of that sissy?
PRED 1175:  I'm not afraid of anything.

SENT 1365:  Well, how would they uncover what happened to Nancy Sloan,
    unless someone from the <unk> told them?
PRED 1365:  We need to know what they know.

SENT 2087:  What does the Prophet Mohammed say?
PRED 2087:  He says that he is a murderer

SENT 2277:  Can I tell you something?
PRED 2277:  You can tell me anything.
```

```
SENT 2315:  What do you want, girl?
PRED 2315:  I want you to tell me the truth.

SENT 2451:  Uh, Long Beach, why?
PRED 2451:  Well, I wanted to talk to you about something.

SENT 2835:  Pardon, sir?
PRED 2835:  I thought I heard something.

SENT 2881:  Now why would you nominate us for an award without even
    mentioning it first?
PRED 2881:  Because I thought it would make you feel better.

SENT 3175:  Why not?
PRED 3175:  Because I don't want you to get hurt.

SENT 4891:  Why didn't you tell us that he is an advocate?
PRED 4891:  He is a advocate?

SENT 5069:  So, Jack, if you didn't come here to kill me, why are you
    here?
PRED 5069:  I want to talk to you.

SENT 5447:  Why would you do that?
PRED 5447:  Because I love you.

SENT 5791:  But if space is like a fabric that can stretch and bend?
PRED 5791:  That's exactly what it is.

SENT 7489:  What time was this, Mr Harris?
PRED 7489:  Quarter past six.

SENT 10913:  You're calling him a liar?
PRED 10913:  I'm not calling him a liar.

SENT 11503:  WHO DOESN'T LIKE PIZZA?
PRED 11503:  I DON'T LIKE PIZZA
```

As we can observe, these answers are much more complex that in any other of the previous examples and in some cases they are even ingenious answers. We can observe in the question of the time how the network generates an answer with a time. This kind of answer is the only possible one for such a question and it shows how the networks has learnt the relation between questions asking for time and answer giving a particular time.

We can also appreciate how the network uses capital letter when the question was also in capital letters. It is interesting because it can be used to let a network know that this could mean that the interlocutor is shouting.

However, these answers are not the most common, and the simple answers (I don't know, what do you mean, I'm sorry, etc) have too much presence in many answers.

So after performing such a training and watching the results, we can imagine how difficult is to get a conversational neural network having a fluid and open domain conversation.

CHAPTER 6

Conclusions

6.1 General Difficulties

As general difficulties encountered during the development of the project, we can mention the following:

- **Structure of the data:** Data , as has been explained on 5.1 and 5.2, has a vital importance in the task of reaching good results in any kind of neural network experiment. In this particular case, the data has been preprocessed and structured in order to get good results. Nevertheless, there are always strange characters that have strange codifications and are hard to clean. This characters can appear many times in the data and can influence the results (they can be selected as candidates for vocabulary for example). Cleaning and adapting the data to the experiment can take long time. In addition, many of these possible problems are usually detected after the training process when observing strange results. This produce a lot of waste of time.

- **CPU and GPU:** These kind of experiments can be executed both in CPU and GPU. However, the capabilities of the GPU in these task far exceed those of the CPU. The problem is that not all the GPUs are compatible with the different frameworks. Nowadays, only some versions of Nvidia GPUs are able to work with. In my particular case, I started working with my CPU due to the fact that my PC has an AMD GPU and this caused that small trainings took ages. Fortunately, thanks to my tutor, I got a connection with a GSI server where I was able to use a Nvidia Titan X with Pascal. This reduced in several orders the time invested in each training.

- **GPU memory:** Despite the use of a powerful GPU, there are many trainings that are impossible to perform due to the lack of GPU memory. This particular GPU has a memory of 12GB GDDR5X but it can be not enough. Some research experiments use tens and even hundreds of millions of sentences with big network architectures. With a Titan X, an experiment with 2 layers (both in encoder and decoder), 4096 units per layer and the dataset of 14 million sentences will produce an "out-of-memory" error. So those huge experiments are performed by using multi-GPU where the work can be divided between them.

- **Debugging:** TensorFlow and PyTorch are frameworks formed by wrappers over other programming languages. This may cause a lot of problems when debugging some errors, complicating the fact of finding the solution.

- **Time:** Without a doubt, time is the most limiting factor in this type of experiments. It is directly affected by all the aspects mentioned above, and in turn, it especially affects the training process. The smallest trainings performed have taken nearly 3 days of calculations. On the other hand, the biggest one took more than 3 weeks and, despite of this, its results have not been as good as I thought. The aspects that produce more increase of calculation time are the size of the dataset and the architecture of the network. Due to the batch nature of the processes, an increase in the size of the dataset, maintaining the network size, will produce a proportional increase in the time. Nevertheless, this is no the case of the architecture. If the size of the dataset is maintained constant and we increase the number of layers, the number of units, the size of the vocabulary or all at the same time, the time will grow in a non proportional way. That is because all of this aspects are related (for example the fully connected nature) and it is difficult to estimate the time impact of the changes before starting to train. As it has been explained, the use of multi-gpus could help by significantly reducing time. This fact acts as an important limitation because is not very productive to wait such a time until see the results. In addition, sometimes results are not very good and you can not predict it a priori, so some experiments become a huge waste of time.

6.2 Particular Difficulties

In this case, these difficulties does not have to be common with other neural network experiments and are more related with the fact of using NMT for generating conversations.

6.2.1 TensorFlow Implementation

The first idea was to use TensorFlow NMT implementation to build the conversational bot. At firs it was used the Open Subtitle corpus of 2 million sentences and after waiting days between trainings all of them produce bad results.

The responses generated by the NMT were plenty of <unk> tags regardless of the size of the vocabulary that was used. Usually, when the <unk> tags appear is because the source word which influence in the word that should appear instead of the tag is not in the vocabulary, so the NMT unknowns the word and this produces that the <unk> reaches more significance than other possible words.

It was thought that it could be due to strange characters that appear in the dataset even when they were preprocessed. All of them were preprocessed many times, deleting all kind of characters include common chars as dots or commas. It was thought that maybe dots or commas were encoded with strange characters and they were influencing in the results because they appear in almost all the sentences. But the responses continued with lots of <unk> tags.

Other option was to leave dots and commas and let them to be in the first positions in the vocabulary. This was done with the idea of given the NMT the possibility of recognize this characters but nothing changed. In the meantime, the trainings performed were getting more and more bigger, with more layers and more units trying to solve this problem, but nothing changed.

The datased was changed and the Cornell dataset was selected because it was already preprocessed. However, the responses continued plenty of <unk> and it was impossible to debugging, in part due to the wrapper architecture of TensorFlow.

After investigate the problem, many topics had been opened on forums in those days asking to solve similar problems and it seemed to be due to changes in the code added to improve the translation process. In that moment, more than half of the time available for the project had passed, so after were discussing it with my tutor, we decide to change to the PyTorch implementation offered by Open NMT.

With this implementations, same experiments gave much better results. If the trainings are big enough, only some <unk> appear which is normal because all the words in the dataset are not in the vocabulary. So that was the main reason of changing to this implementation.

In essence, changing technology did not involve major changes because TensorFlow and PyTorch are two very similar libraries with an operation that , in practice, is identical.

6.2.2 “I don’t know” problem

This is a common problem and it particularly affects conversational bots in open domain conversations.

After training a model, many of the responses to diverse questions tend to be responses of the style of “I don’t know” or “What do you mean?”. There are more repetitive structures like “I’m going to...” or “Yes” and so on. This problem appear because this kind of responses are very common in the training data and they give an answer for multiple possible questions [31].

Therefore, when the neural network is learning the relations between questions/answer, it learns that common responses lead it to good error results. That is because an “I don’t know” generated response compared with the possible labeled reference answers is going to give a low error. So, once the network reaches this point it is difficult that a change in its parameter leads it to responses that generate a smaller error, even if we tune the learning rate during the training. This behavior can be compare to the fact of reaching a local minimum in a function. If the network reaches this state, the gradient functions will indicate to always return to the same point because it is the best point around the multidimensional function.

This responses are apparently not a good action to take, since i closes the conversation down. They can be good particular answers but they do not allow to continue a conversation. In addition, this responses in the case of facing two trained models will become stuck in an infinite loop of repetitive responses.

There are some papers that face this problem by adding techniques of reinforcement learning that have been applied in MDP and POMDP dialogue systems. They introduce a neural reinforcement learning generation method, which which can optimize long-term rewards designed by developers. Therefore, some rewards are given to good conversations and, defining heuristic approximations, the conversational agents will optimize these long-term rewards, behavior that leads them not to always use “I don’t know answers”. So, by defining a reward policy the agent learns to optimize it by using policy gradient methods [32].

Unfortunately these improvements are very recent and have not yet been incorporated into open source implementations, so the problem of “I do not know” answers is still difficult to solve.

6.3 Future lines of work

As future work, the main option is the use of several GPU. The frameworks and libraries used during the project are prepared to be used with several GPUs by using the multi-gpu system. This allows to divide the workload by numerating (-gpuid) the GPUs that are able to work with.

At present, the GSI server has three Nvidia Titan X that can be used in parallel. This could allow to considerably reduce training time and, at the same time, it will allow to perform larger experiments which will probably reach better results.

On the other hand, it is also possible to explore the exact reason of why the network tend to answer with part of the question when using question/answer structured dataset. Finding out the reason will allow the network to be nearer of maintaining a ”good” conversation.

6.4 Final Conclusions

The generation and understanding of natural language is still an objective that has a long way to advance to get a state where a machine is able to maintain a perfect open domain conversation with a human.

Nowadays there are some projects that have reached good results, getting models able to have general conversations with a medium level of complexity. But the absolutely understanding between machines and human using natural language is still far.

However, advances in electronics are leading to the production of increasingly powerful GPUs. This, at the same time, allows to perform larger experiments in less time so these advances are directly increasing the speed at which advances in the NLP with neural networks occur. New models and techniques that improve current systems are constantly appearing, therefore, we are moving exponentially towards a complete solution of this problem.

All of this, without any doubt, makes this deep learning field one of the most exciting and with higher expectations of progress in the coming years, because Artificial Intelligence is not only a matter of the future but is also a matter of the present.

Bibliography

[1] OpenSubtitles corpus, the open parallel corpus. `http://opus.nlpl.eu/OpenSubtitles.php`.

[2] Pytorch tutorials pytorch. `http://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html`.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[6] Eugene Brevdo. Attention/Decoder Wrapper. `https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/seq2seq/python/ops`.

[7] Dr. Jason Brownlee. Machine Learning Mastery. `https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/`.

[8] Dr. Jason Brownlee. Machine Learning Mastery machine learning blog. `https://machinelearningmastery.com/what-are-word-embeddings/`.

[9] Edgar Nelson Sánchez Camperos and Alma Yolanda Alanís García. *Redes neuronales: conceptos fundamentales y aplicaciones a control automático*. Pearson Educación, 2006.

[10] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[12] Paul R Cohen. *Empirical methods for artificial intelligence*, volume 139. MIT press Cambridge, MA, 1995.

[13] Deborah Coughlin. Correlating automated and human assessments of machine translation quality. In *Proceedings of MT summit IX*, pages 63–70, 2003.

[14] Diego San Cristóbal. Clases Auxiliares. `https://github.com/DiegoSCB/Clases-auxiliares`.

[15] Diego San Cristóbal. Prediction Results. `https://github.com/DiegoSCB/predicciones`.

[16] Pedro Ponce Cruz and Alejandro Herrera. *Inteligencia artificial con aplicaciones a la ingeniería*. Marcombo, 2011.

[17] Cristian Danescu-Niculescu-Mizil. Cornell Movie Dialogs Corpus. `http://opus.nlpl.eu/OpenSubtitles.php`.

[18] Cristian Danescu-Niculescu-Mizil and Lillian Lee. Chameleons in imagined conversations: A new approach to understanding coordination of linguistic style in dialogs. In *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics, ACL 2011*, 2011.

[19] Óscar Araque Diego San Cristóbal. Training Scripts. `https://github.com/DiegoSCB/initial-bot-OpenNMT`.

[20] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[21] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[22] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 2017.

[23] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[24] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

[26] Andrej Karpathyl. Andrej Karpathy blog. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

[27] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *AAAI*, pages 2741–2749, 2016.

[28] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017.

[30] Raymond Kosala and Hendrik Blockeel. Web mining research: A survey. *ACM Sigkdd Explorations Newsletter*, 2(1):1–15, 2000.

[31] Satwik Kottur, Xiaoyu Wang, and Vitor R Carvalho. Exploring personalized neural conversational models.

[32] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*, 2016.

[33] Yang Liu and Mirella Lapata. Learning structured text representations. *arXiv preprint arXiv:1705.09207*, 2017.

[34] Minh-Thang Luong, Eugene Brevdo, and Rui Zhao. Neural machine translation (seq2seq) tutorial. *https://github.com/tensorflow/nmt*, 2017.

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

[36] Michael Nielsenl. Neural Networks and Deep Learning. `http://neuralnetworksanddeeplearning.com/chap5.html`.

[37] Michael Nielsenl. Neural Networks and Deep Learning Book. `http://neuralnetworksanddeeplearning.com/chap1.html`.

[38] Joakim Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*, pages 351–359. Association for Computational Linguistics, 2009.

[39] Christopher Olahl. colah's blog. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[40] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[41] Elaine Rich and Kevin Knight. Artificial intelligence. *McGraw-Hill, New*, 1991.

[42] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[43] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[44] Rico Sennrich and Barry Haddow. Linguistic input features improve neural machine translation. *arXiv preprint arXiv:1606.02892*, 2016.

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[46] Jörg Tiedemann. News from OPUS - A collection of multilingual parallel corpora with tools and interfaces. In N. Nicolov, K. Bontcheva, G. Angelova, and R. Mitkov, editors, *Recent Advances in Natural Language Processing*, volume V, pages 237–248. John Benjamins, Amsterdam/Philadelphia, Borovets, Bulgaria, 2009.

[47] Oriol Vinyals and Quoc Le. A neural conversational model. *arXiv preprint arXiv:1506.05869*, 2015.

[48] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

[49] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.

[50] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[51] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated recurrent neural networks. *arXiv preprint*, 2015.