UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Text translation using Machine Learning

Diploma Thesis

Kyriaki Chiona

Supervisor: Tsalapata Hariklia

Volos 2021

UNIVERSITY OF THESSALY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Text translation using Machine Learning

Diploma Thesis

Kyriaki Chiona

Supervisor: Tsalapata Hariklia

Volos 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Μετάφραση κειμένου με χρήση Μηχανικής Μάθησης

Διπλωματική Εργασία

Κυριακή Χιονά

Επιβλέπων/πουσα: Τσαλαπάτα Χαρίκλεια

Βόλος 2021

Approved by the Examination Committee:

Supervisor	**Tsalapata Hariklia** E.DI.P., Department of Electrical and Computer Engineering, University of Thessaly
Member	**Stamoulis Georgios** Professor, Department of Electrical and Computer Engineering, University of Thessaly
Member	**Daskalopoulou Aspasia** Assistant Professor, Department of Electrical and Computer Engineering, University of Thessaly

Date of approval: 24-2-2021

Acknowledgements

I would like to thank my family and friends for helping me to achieve my goals and especially my friend Stylianos Gkountouvas.

DISCLAIMER ON ACADEMIC ETHICS AND INTELLECTUAL PROPERTY RIGHTS

Abstract

The main purpose of this thesis is to study machine translation. Specifically, neural machine translation using neural networks to achieve our goal. The model we used is based on the Encoder-Decoder architecture, commonly called Sequence to Sequence. This model can produce a sequence given as input another sequence using a variety of recurrent neural networks. To create a neural machine translation model a parallel text corpus is required and the given sentences must undergo some pre-processing, which includes tokenization and encoding the tokens. We tested two different methods of tokenization, character level and word level, and two methods of encoding the tokens, using one-hot vectors and embedding vectors. A variety of tests have been performed, using different parameters, in order to achieve the best translation results according to bilingual evaluation understudy score, a score designed for translation tasks. Finally, after evaluating the performance of many models, we conclude to the advantages and disadvantages of each model.

Περίληψη

Ο κύριος σκοπός αυτής της διπλωματικής είναι η μελέτη συστημάτων μηχανικής μετάφρασης. Συγκεκριμένα, χρησιμοποιήσαμε συστήματα νευρωνικής μηχανικής μετάφρασης για να επιτεύξουμε τον στόχο μας. Το μοντέλο που χρησιμοποιήσαμε βασίζεται στην αρχιτεκτονική Encoder-Decoder, γνωστό και σαν Sequence to Sequence. Αυτό το μοντέλο μπορεί να δεχθεί σαν είσοδο μια ακολουθία και να δημιουργήσει μια άλλη ακολουθία χρησιμοποιώντας διάφορα αναδρομικά δίκτυα. Για την δημιουργία ενός μοντέλου μηχανικής μάθησης είναι απαραίτητη η χρήση μια συλλογής παράλληλων προτάσεων και η προ-επεξεργασία τους. Δοκιμάσαμε διάφορες μεθόδους προ-επεξεργασίας όπως τον χωρισμό κάθε πρότασης σε λέξεις ή σε χαρακτήρες. Στην συνεχεία κωδικοποιήσαμε αυτές τις λέξεις ή τους χαρακτήρες χρησιμοποιώντας διανύσματα one-hot ή διανύσματα embedding. Πραγματοποιηθήκαν διάφορες δοκιμές, αλλάζοντας κάθε φορά παραμέτρους, ώστε να επιτύχουμε το καλύτερο αποτέλεσμα στις μεταφράσεις σύμφωνα με ένα κριτήριο δίγλωσσης αξιολόγησης που ονομάζεται BLEU. Στο τέλος, αξιολογήσαμε τα μοντέλα μας και αναφέραμε τα πλεονεκτήματα και μειονεκτήματα κάθε μοντέλου.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Nowadays, more and more people want to communicate each other but a big barrier exists. This barrier is the variety of languages that makes communication difficult and requires translations to be created by hand. This mundane task of translating documents to other languages must be automated in order to create the required translations quickly and efficiently.

1.1 Motivation

In this thesis we will investigate how we can solve the task of automated translation using machine learning and all the required steps to do this. There have been many attempts to create models for this purpose, following many different approaches like Example-based machine translation (EBMT), Rule-based machine translation (RBMT), Statistical machine translation (SMT) and finally Neural machine translation (NMT).

The first models developed was statistical models [14], [15] that are based on information theory. These models attempt to translate documents according to a probability distribution function where a string from the target language is the translation of a string from the source language. Another approach, commonly called the classical approach, is creating translations using a rule-based system [16]. A rule-based system creates translations according to linguistic information extracted from the grammars and dictionaries of the two languages and converts the source sentences to the targets based on morphological, syntactic, and semantic analysis of both languages. The complete opposite of a rule-based system is an example-based system [17]. Such systems are not based on deep linguist analysis but instead splits the source sentence in parts, translates each part and then combines these translated parts to create the target sentence. Finally, neural machine translation is performed by systems that are based in artificial neural networks [18].

We will attempt to explain and analyse all the steps involved in creating a complete model, from pre-processing the data to evaluating the performance of a model.

1.1.1 Contribution

The model we will develop is based on the Sequence to Sequence architecture. This architecture is using two separate models, one that processes the input sequence and another one that produces the output sequence. In neural machine translation a bilingual, or parallel, text corpus is required and some pre-processing is applied to it. The pre-processing involves many steps, like cleaning the text and remove any unnecessary character, and splitting each sentence in tokens. Two different tokenization methods will be used. First of all, we will split each sentence into characters, which means our tokens will consist of all unique characters, and the model will produce the predicted sentence character by character. Secondly, each sentence will be split into words and each unique word will be a token. Moreover, each token, in a sequence, must be encoded in a format suitable for the model. We decided to use two different encoding formats, creating one-hot vectors or embedding vectors for each token. In the end, we will compare the different tokenization methods and encoding formats. Finally, a metric is necessary in order to help us evaluate each model. The classic metrics, like accuracy, are not suitable for the task of neural machine translation. This lead us to use BLEU as a metric, which is a metric specifically designed for scoring translation tasks, to evaluate our models.

1.2 Organization of the thesis

This thesis consist of five chapters. In the second chapter (2) we give a general overview of machine learning and we present the basic concepts required to understand how a machine learning model works. Next (3), we present the dataset used and the pre-processing steps required along with the two different encoding method we use. After this section, we introduce a method of evaluating machine translation results called bilingual evaluation understudy (BLEU). Afterwards (4), we analyse how the Sequence to Sequence architecture works and it’s components and we develop a model based on this architecture. Moreover, we present the results of our models, after extensive testing, using different parameters and network types. Finally (5), we summarize our results and suggest some future actions to potentially improve the results.

Chapter 2

Introduction to machine learning

2.1 Introduction

Machine learning (ML) is a branch of artificial intelligence (AI) (Fig. 2.1) that studies algorithms that act without being explicitly programmed. Machine learning can be considered a subset of artificial intelligence. Machine learning algorithms learn from experience and build models based on a sample of data called ”training data”. Those models have the capacity to constantly improve by exposing the model to new data, whereas rule-based algorithms will perform always the same. Nowadays machine learning usage is constantly increasing and we can find more and more fields where it is used like self-driving cars, speech recognition, neural machine translation, effective web search, etc.

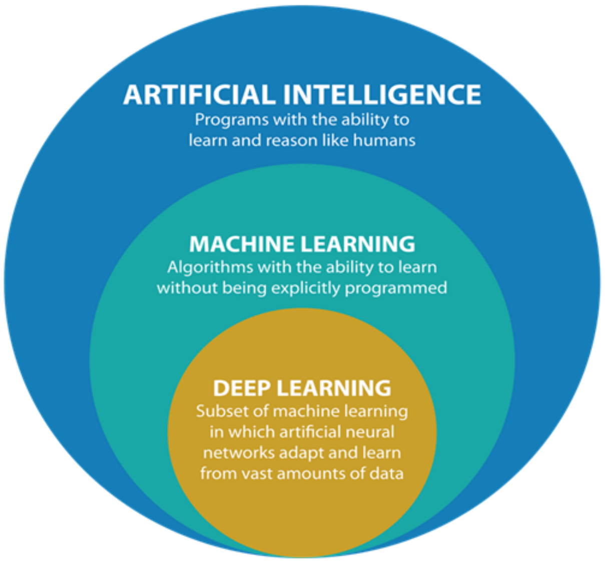

ARTIFICIAL INTELLIGENCE
Programs with the ability to learn and reason like humans
MACHINE LEARNING
Algorithms with the ability to learn without being explicitly programmed
DEEP LEARNING
Subset of machine learning in which artificial neural networks adapt and learn from vast amounts of data

Figure 2.1: Relation between AI, ML and deep learning. [1]

Machine learning problems are often divided in many categories, but the biggest are supervised and unsupervised. In supervised problems,that include classification and regression, we have both the training data and the expected answer. In contrast in unsupervised problems we only have the training data and we expect algorithm to give us an answer. Unsupervised problems include clustering, dimensionality reduction, finding relations between word vectors (word embeddings), etc.

2.1.1 Machine learning models

Performing machine learning involves creating a model which is trained on some data and then can process more data to make predictions.

There are many different models and algorithms depending on the task at hand, but the most commonly used are the following.

Regression analysis

Regression analysis contains many statistical methods and algorithms with the purpose of finding a relation between the input variables and their features. The most common form of regression analysis is linear regression (Fig. 2.2), which only applies if the relation is linear, but more advanced forms can be used like ridge regression, logistic regression, polynomial regression.

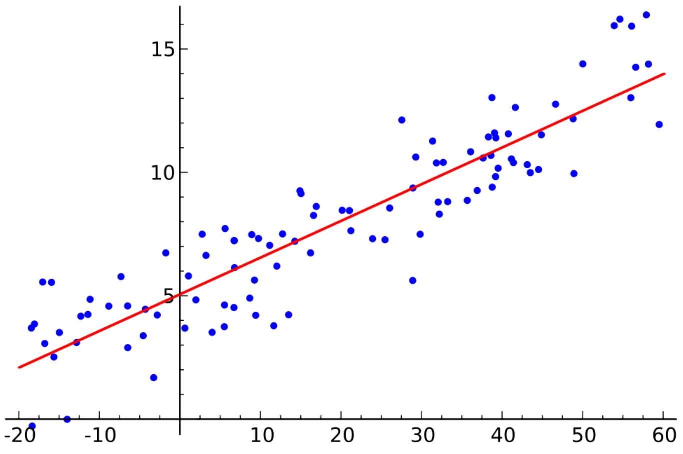


Figure 2.2: Linear regression. [2]

Decision trees

Decision tree learning uses ”trees” as a model to make predictions (Fig. 2.3). Each example of the training data has its features analysed and based on this analysis the internal nodes of the tree are created. These nodes act as a filter which guide new data presented to the model a specific leaf, which is the prediction of the model.

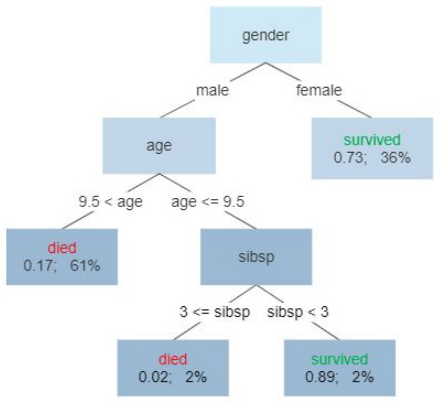


Figure 2.3: Decision trees showing the survival of the passengers on the Titanic. [3]

Clustering algorithms

Clustering analysis is a set of algorithms with their main propose to create groups (clusters) from the input data (Fig. 2.4). Cluster analysis itself is not one specific algorithm but the problem to be solved and this can be achieved with many different algorithms like k-nearest neighbors algorithm (k-NN), Nearest-neighbor chain algorithm, Quantum clustering, etc.

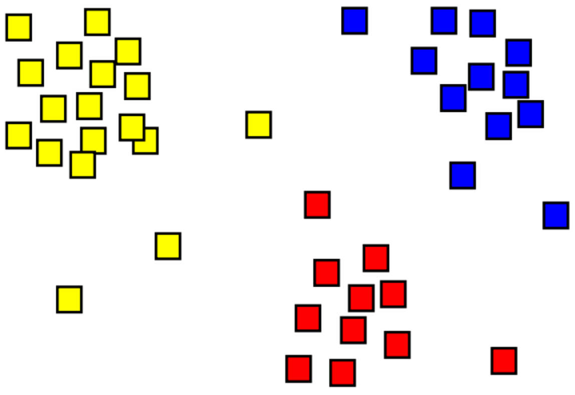


Figure 2.4: The result of clustering. [4]

Support vector machines (SVMs)

Support vector machines (SVMs) are a group of related supervised learning algorithms that are used for classification or regression (Fig. 2.5). These algorithms are very powerful binary linear classifier due to their non-probabilistic nature. Modifications to the algorithm exists for non-linear problems. Support vector machines are also called maximum-margin classifier due to the fact that they maximize the distance of the hyper-plane nearest data point on each side.

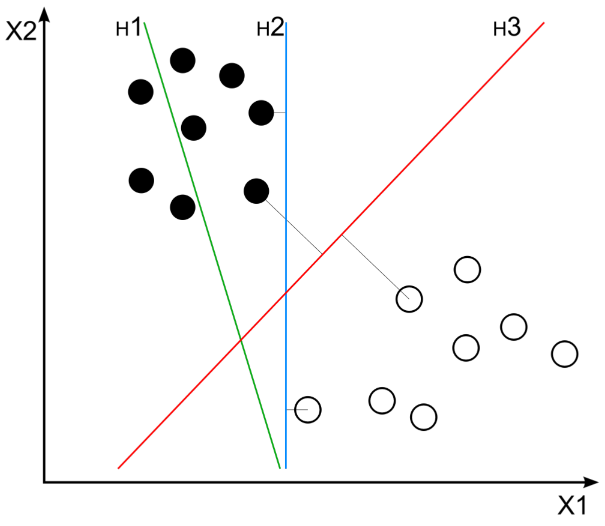


Figure 2.5: SVM maximum-margin classifier. [5]

Genetic algorithms

A Genetic algorithm is a algorithm that mimics the biological evolution. It is a search algorithm and heuristic technique that uses methods such as mutation and crossover to generate models that improve predictions.

Artificial neural networks

Artificial neural networks are systems inspired by the biological neurons that constitutes animal brains. A artificial neural network is s system composed by many connected units called neurons. Each connection has some inputs and one output so it can transmit information to each other. The output is a function of it's neuron inputs with some weights. During training these weights are adjusted using algorithms like gradient decent. Many neurons are grouped into layers that may perform specific tasks and many layers are grouped into a model

(Fig.2.6). The signal propagates through the first layer to the last, possibly after traversing the layers multiple times.

There are many different categories of neural network layer like a dense layer, convolution layers and recurrent layers. Some of those layers will be analysed in the following section.

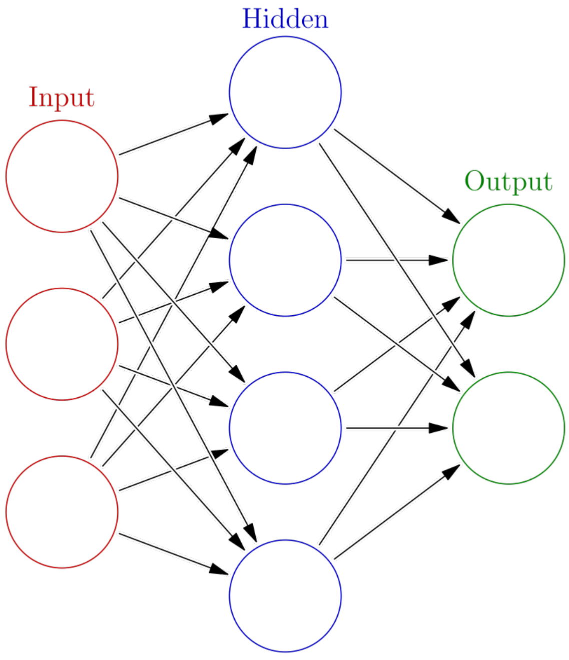


Figure 2.6: Artificial neural network with one hidden layer. [6]

2.1.2 Loss function and gradient decent

Almost all machine learning models use some sort of a function, called loss function, to estimate the parameters used by the model. This function maps an action to a real number or calculates the difference between the real value and the prediction. The purpose of the model is to minimize the loss function. However this function dose not have an explicit form or may depend on a huge number of variables and calculating the minimum using analytical mathematics may be impossible.

To solve this problem we use an algorithm called gradient descent. This algorithms uses an iterative method to find the minimum of a function. The algorithm uses the first order derivative of the function we try to minimize and a initial point. At each iteration we attempt to move in the opposite direction of the derivative (eq. 2.1). To avoid overtaking the minimum we introduce a term, λ, called learning rate to reduce the size of the step.

$$x_{n+1} = x_n - \lambda \nabla f(x_n) \tag{2.1}$$

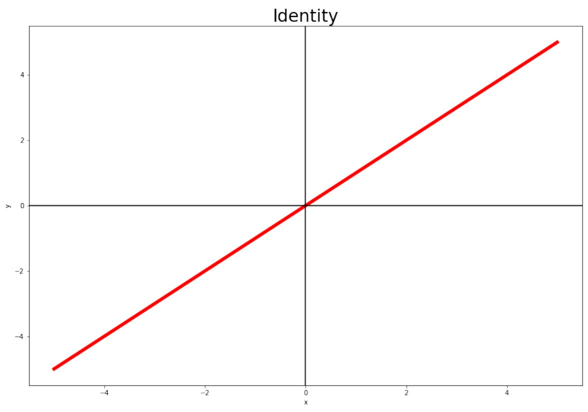
By calculating constantly consecutive points we will eventually reach a minimum, local or global. Most machine learning model use a variation of gradient descent, with either changing how the learning rate works or how each variable of the loss function affects the new point.

2.1.3 Activation functions

The output of each neuron passes though an activation function. The purpose of the activation function is to introduce non-linearity into the output of a neuron. Some of the most used activation functions are presented bellow.

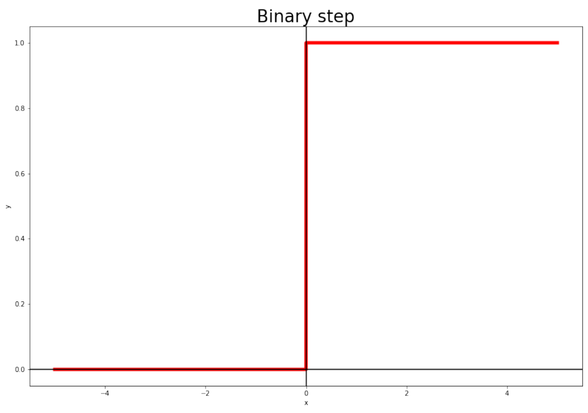
Identity

$$f(x) = x$$



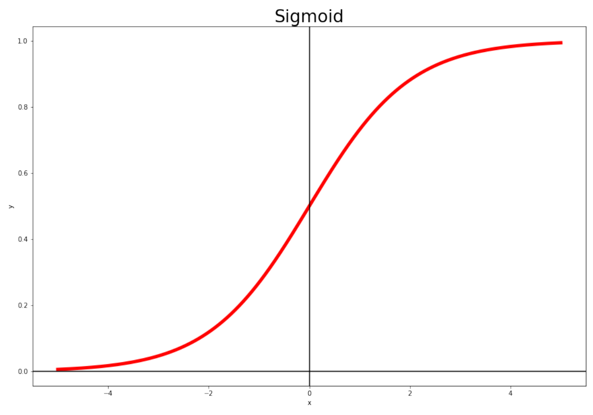
Binary step

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$



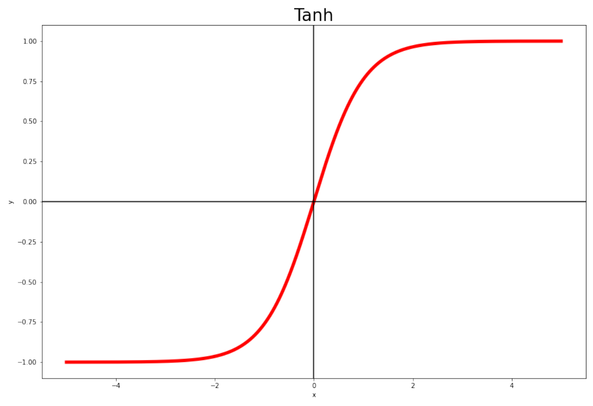
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



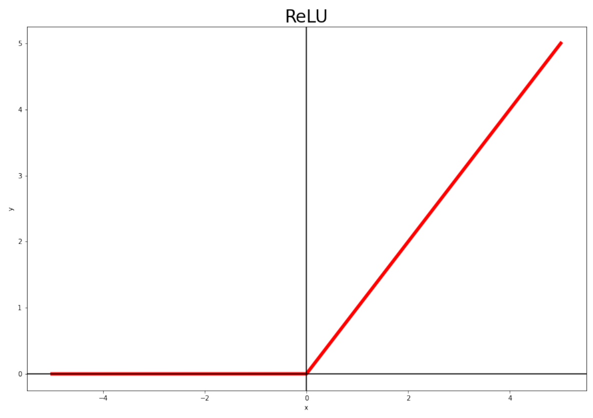
Tanh

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$



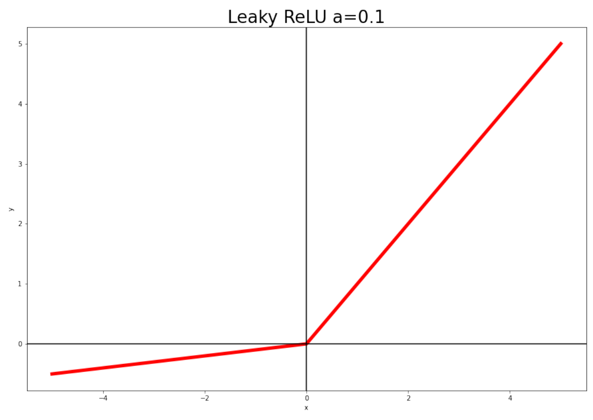
Rectified linear unit

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



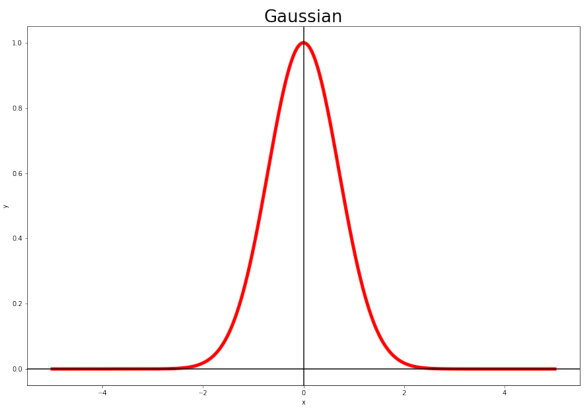
Leaky rectified linear unit

$$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$



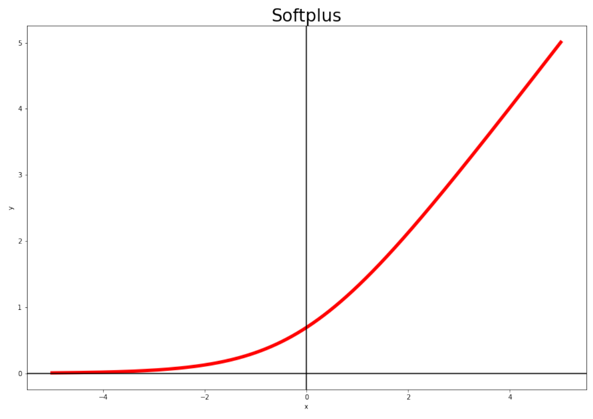
Gaussian

$$f(x) = e^{-x^2}$$



Softplus

$$f(x) = \ln\left(1 + e^x\right)$$



Softmax

The Softmax function is an activation function applied to a whole layer instead of a single neuron. This function takes as argument the outputs of all the neurons, usually of the last layer, for which calculates a probability in order to determine the final result. For this purpose a vector of probabilities is created, one for each potential outcome, which has a sum one.

The output of each neuron is calculated using the following equation (eq. 2.2).

$$s(y)_i = \frac{e^{y_i}}{\sum_{j=1}^{N} e^{y_j}} \tag{2.2}$$

2.2 Perceptron

The Perceptron [19], [20] is a simple algorithm to classify data in two classes. It can be considered the simplest neural network that consist of only one neuron and is used to train a binary classifier using supervised learning. Perceptron is a linear classifier and will always reach a final state if our data are linear separable. However, if the training data are not linearly separable, which means a hyperplane cannot separate them, the algorithm will not converge and will fail completely. A layer that consists of many perceptron units is called a dense layer.

It has a number of inputs and one output, either 1 or 0, which is a mathematical relation between the inputs and some weights (eq. 2.3).

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

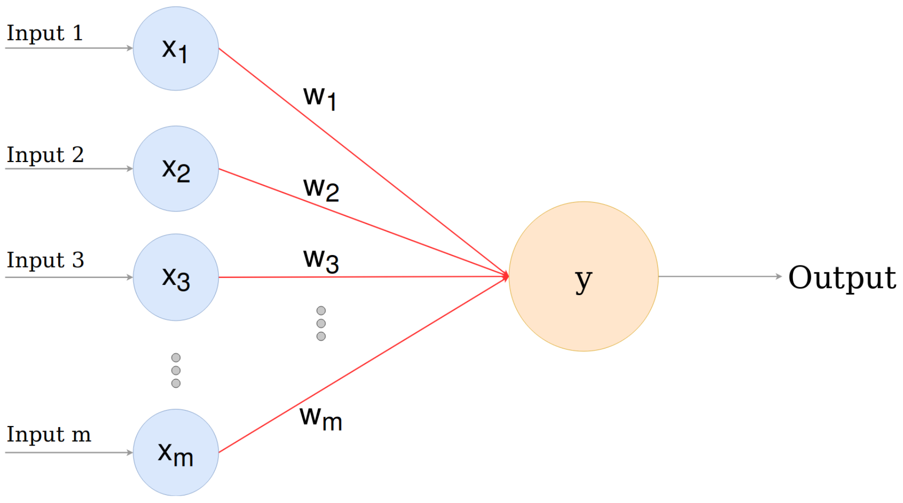


Figure 2.7: Perceptron. [7]

2.3 Multilayer perceptron

A multilayer perceptron neural networks is a feedforward multilayer network based on the perceptron. It contains at least three layers one input, one output and one or more hidden layers (fig. 2.8). Feedforward networks are the most common type of networks. In these networks information flows from the input layer to the output without loop. The goal of feedforward network is to find appropriate weights to approximate a function. It is proven, in the universal approximation theorem [], that a arbitrary large multilayer perceptron with one hidden layer can approximate every function. However it is not viable for a layer to have infinite neurons so in practice we add additional layers to achieve this.

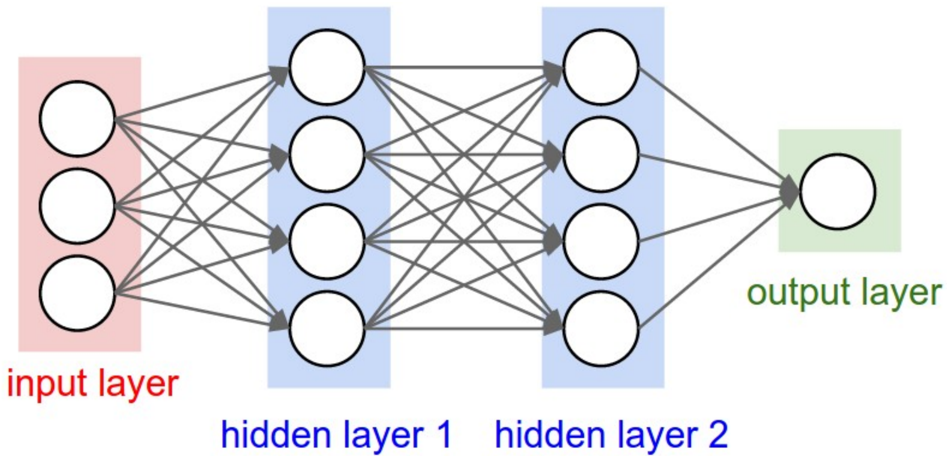


Figure 2.8: A four layer perceptron network. [8]

The multilayer perceptron networks train in a similar manner to the simple perceptron, but with an important difference. Due to the multilayer nature of the network, the output is a function of all the wights in every layer, so the gradient updates have to move from the output to the previous layers. This algorithm is called backpropagation.

2.3.1 Back-propagation

In multilayer neural networks the output of the network depends on the weights of all the layers (eq. 2.4). Due to this relation we cannot use normal gradient descent to calculate the gradient and adjust the weights.

$$g(x) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 x))) \tag{2.4}$$

With x being the input vector, y the target vector, f^l the activation function at layer l, W^l the weights at layer l and L the number of layers.

By defining the cost function as $C(y_i, g(x_i))$ we attempt to find the partial derivative of C with respect to each weight w^l_{jk}. By computing $\partial C/\partial w^l_{jk}$ we can calculate the change we must do to each weight in order to approach the target value y_i

Back-propagation [21], [22] is an algorithm that calculates all those gradients efficiently and without duplicate calculations. These gradients can be used with an optimization algorithm to update the weights of the network.

2.4 Recurrent neural networks

The idea behind recurrent neural networks (RNN) is to create a network that can use sequences as inputs. These sequences may have different lengths and their values may be related. The main difference of recurrent neural networks to feedforward networks is that they contain loops (fig. 2.9) [23]. This difference facilitates the training of neurons to handle long term dependencies in the input sequences by allowing them to use further information. In this way, the neurons have an internal memory created from their input, which depends on the previous values.

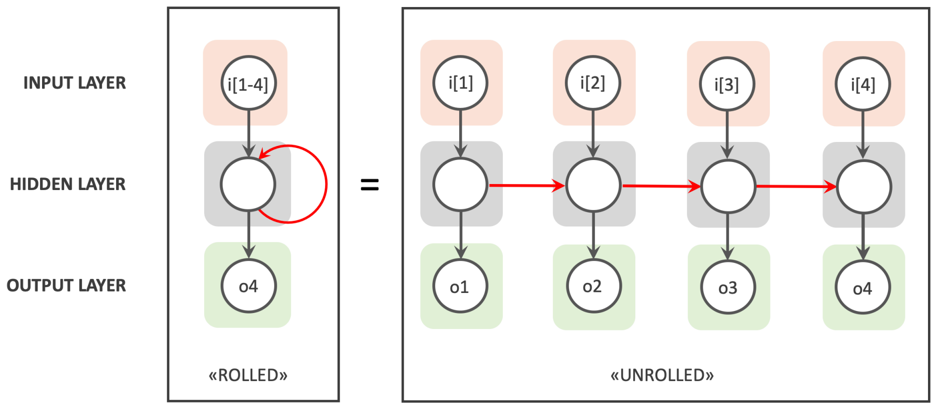


Figure 2.9: Unfolded recurrent neural network. [9]

However, the problem of vanishing gradients appears in ordinary recurrent neural networks during training. In a very long sequence, it's values may have long term dependencies. When an ordinary recurrent neural network have to memorize this sequences it is possible that these dependencies cannot be learned due to the nature of the training algorithm. A solution to this problem is to create variations of the ordinary recurrent neural networks that control how these gradients are calculated.

2.4.1 Long short-term memory

Long short-term memory [24] is a variation of ordinary recurrent neural networks that tries to solve the vanishing gradients problem. The long short-term memory cell is composed of many gates that control the gradients (fig. 2.10). These gates are the input gate (eq. 2.5), the output gate (eq. 2.7) and the forget gate (eq. 2.6). Specifically the input gate controls the information that enters the cell, the forget gate control the information that creates the cell memory and the output gate that controls the output of the cell. The equations bellow describe how these gates work.

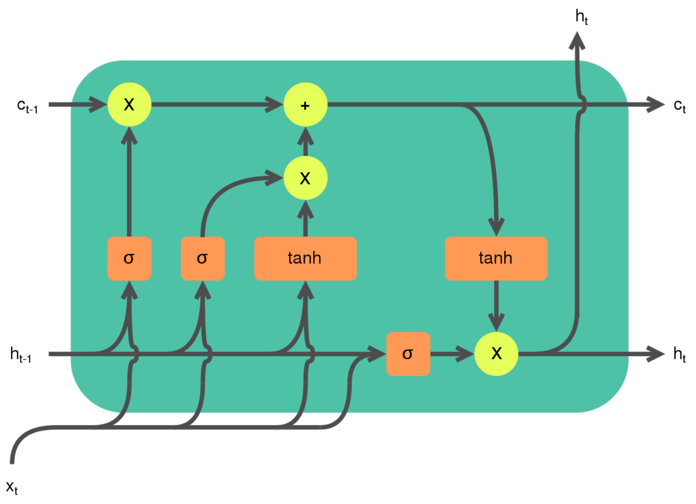


Figure 2.10: Long short-term memory architecture. [10]

In the following equations [10] the x_t is the input vector, W and U are weight matrices and b is the bias. The equations use the sigmoid function σ and the hyperbolic tangent function $\tanh$ and the operator $\circ$ denotes element-wise multiplication.

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.5}$$

Where i_t is the activation of the input gate.

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.6}$$

Where f_t is the activation of the forget gate.

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.7}$$

Where o_t is the activation of the output gate.

$$\begin{aligned} c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ h_t &= o_t \circ \tanh(c_t) \\ \tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \end{aligned} \tag{2.8}$$

Where c_t and h_t is the cell state vector and output vector (also called hidden state) vector respectively and the $\tilde{c}_t$ the cell input activation vector.

2.4.2 Gated recurrent unit

The gated recurrent unit (GRU) [25] is a variation of LSTM designed to be simpler to compute and implement (fig. 2.11). The main difference of gated recurrent units is that they have only two gates instead of three. The two gates of the gated recurrent unit is the update gate (eq. 2.9) and the reset gate (eq. 2.10). However, although recurrent neural networks with infinite precision in the states and unbounded computation time are turing complete, it is stated that LSTMs are strictly stronger than GRUs ([26], [27]) and in many problems, like neural machine translation, they underperform.

In the following equations [11] the x_t is the input vector, W and U are weight matrices and b is the bias. The equations use the sigmoid function σ and the hyperbolic tangent function $\tanh$ and the operator $\circ$ denotes element-wise multiplication.

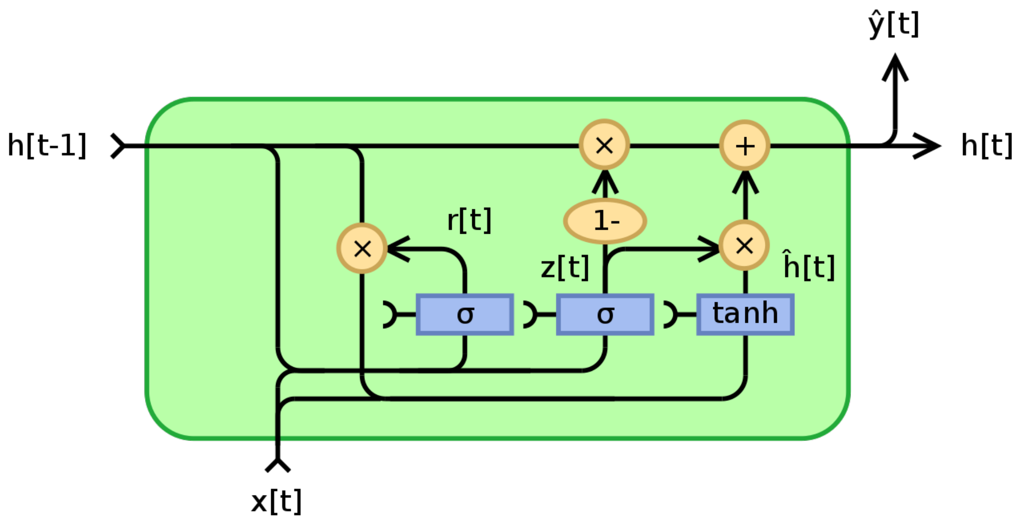


Figure 2.11: Gated recurrent unit architecture. [11]

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{2.9}$$

Where z_t is the activation of the update gate.

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{2.10}$$

Where t_t is the activation of the reset gate.

$$\begin{aligned} \hat{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t \end{aligned} \tag{2.11}$$

Where h_t is the output vector and $\hat{h}_t$ is the candidate activation vector.

Chapter 3

Text prepossessing and machine translation metrics

3.1 Prepossessing raw text

The dataset used in this thesis is a parallel corpus that consists of two languages and that contains sentences in the original language alongside their translation. Specifically experiments were done using the English - Italian ANKI dataset [28] that contains 343813 pair sentences.

The pair of sentences first must be converted to a format that is ”understood” by the machine learning algorithm. There are multiple steps in processing raw text. The first step is to keep only the characters and the punctuation that belongs to our alphabet. We may also remove ascents that we don’t want and convert everything to lowercase.

After this process we have to tokenize our sentences and then we can proceed in two different ways. The first way is known as one-hot encoding, while the second one is to create embeddings.

3.2 Tokenization

In natural language processing (NLP) tokenization is the process of splitting text in small chunks (tokens). These tokens can either be words or characters. After creating these tokens we assign a unique number to each unique tokens, as shown below.

```
text_data =  ['good morning', 'the weather today is good']
```

```
tok_text =    [['good', 'morning'],
               ['the', 'weather', 'today', 'is', 'good']]
tok_result = [[0, 1],[2, 3, 4, 5, 0]]
```

This conversion from chunks to numbers allows us to represent text in a format which enables us to train machine learning models using text as input. From this process we produce categorical data, however machine learning model cannot use directly these data and these sequences of numbers must be processed further.

3.3 Categorical data encoding

3.3.1 One-hot encoding

One way to process categorical data is to encode them to vectors using one-hot encoding. In one-hot encoding we produce binary sparse vector, specifically vectors that only one of their values is one and all others are zero. This conversion allows machine learning model to extract more meaningful information from categorical variables, because their initial values does not represent any relations and does not have any specific meaning.

The process to encode categorical data to one-hot vectors is very simple. First for every token, we have to create a vector with its size being equal to the number of tokens we have. Then we set all of its values being zero, except one, where the position of of this value depends on the number assigned to this token. This is demonstrated bellow.

```
[0, 1] = [[1, 0, 0, 0, 0, 0],[0, 1, 0, 0, 0, 0]]
[2, 3, 4, 5, 0] = [[0, 0, 1, 0, 0, 0],
                   [0, 0, 0, 1, 0, 0],
                   [0, 0, 0, 0, 1, 0],
                   [0, 0, 0, 0, 0, 1],
                   [1, 0, 0, 0, 0, 0]]
```

However this encoding although very expressive can generate huge vectors with all but one of their values zeros and this can cause problems with the training of the model or the storage required for the dataset.

3.3.2 Token embeddings encoding

Another way to prepossess categorical data and specifically text data is to create an embedding for each token. An embedding is a vector with a relative low dimensionality that is either created beforehand or during training [29]. With embeddings instead of representing a token with one big binary sparse vector we attempt to map it with a smaller but denser vector. Moreover, embeddings try to capture more information (fig. 3.1) from the token like the meaning, the interpretation of the word and even grammatical and lexicological information.

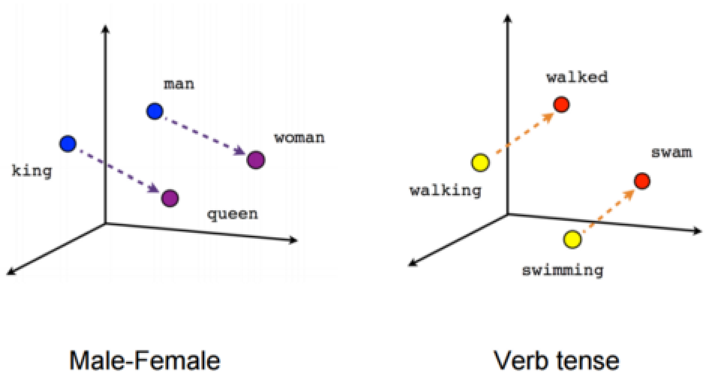


Figure 3.1: Embeddings capturing word semantics. [12]

The embeddings generation process can either be a layer in our machine learning model or we can create a new model specifically to generate those embeddings. An example of the embedding vectors, with the output dimension being three, is presented bellow.

```
[0, 1] = [[0.01075683, -0.01871429, -0.01149996],
          [ 0.03609798, -0.04325888,  0.04786615]]
[2, 3, 4, 5, 0] = [[ 0.01533573, -0.01497992,  0.04166322],
                   [-0.00210105, -0.01850808, -0.04897275],
                   [ 0.0454616 ,  0.0019915 , -0.03638037],
                   [-0.00881171,  0.01351931, -0.04072988],
                   [ 0.01075683, -0.01871429, -0.01149996]]
```

3.4 Machine translation metrics

In most machine learning problems, like regression or classification, it is quite easy to evaluate the performance of a model, due to the fact that it can be calculated using a cost function. For instance, in a regression problem, although the fact that the output can take any value, we can calculate the difference of this value to the expected output.

However, in machine translation, due to the complexity of the task we cannot simply calculate such a metric without limiting our model. For example, when translating a sentence, there are several ways to express the meaning of the sentence in the target language.

For this reason, new metric that evaluate the performance of a machine translation model was developed and the most used of them is called Bilingual Evaluation Understudy or BLEU.

3.4.1 Bilingual evaluation understudy

Bilingual Evaluation Understudy (BLEU) is an algorithm developed to evaluate the translations of one language to another [30]. The popularity of the algorithm is high due to the fact the the scores it produces are similar with how humans would evaluate the translation.

The final score, which is always between 0 and 1, is calculated by averaging each translated sentence's score. Each translated sentence's score is calculated by comparing it with a list of reference translations, using a form of modified precision calculating what percentage of n-grams can be found in the reference translations. The most common version of the algorithm BLEU uses n-grams from 1 to 4 and it is called BLEU4.

Bellow is presented an example that demonstrates how does BLEU2 works.

```
Reference 1: the cat jumps high
Reference 2: this cat can jump high
Translated: the cat high
```

First we calculate all the 1-grams of the translated sentence that exist in the reference sentences.

1-grams counts	
the	1
cat	1
high	1

Table 3.1: Number of 1-grams counts

As we can see the total number of 1-grams is 3 and the total number of 1-grams counts is 3, so the 1-gram precision is $P_1 = \frac{3}{3} = 1$

Then we calculate all the 2-grams of the translated sentence that exist in the reference sentences.

2-grams counts	
the cat	1
cat high	0

Table 3.2: Number of 2-grams counts

As we can see the total number of 2-grams is 2 and the total number of 2-grams counts is 1, so the 2-gram precision is $P_2 = \frac{1}{2} = 0.5$

Finally the BLUE2 score of this sentence is $\frac{P_1+P_2}{2} = \frac{1+0.5}{2} = 0.75$

The BLEU algorithm is often applied to a collection of translated sentences and achieving a score 1 is almost impossible and many time undesirable. Human translations usually achieve a score of $0.6 - 0.7$ ([]) due to the fact that each human can translate a sentence using different words but with the same meaning. Usually a BLEU score of $0.3 - 0.4$ indicates that the translations are understandable but the quality is not very high and a score of $0.4 - 0.5$ indicates high quality translations [31].

Chapter 4

Sequence to Sequence architecture

4.1 Sequence to Sequence

Sequence to Sequence (Seq2Seq) is a architecture designed to handle variable-length sequences as input and output. This architecture has two major components, the encoder and the decoder, hence it is also called Encoder-Decoder Network.

This architecture was initially developed by Google [32] to be used in machine translation, however it proved useful in many other tasks like summarization, conversational modeling and image captioning, due to the fact that it is not necessary for the output and the input to be the same format. Moreover, Facebook [33] managed to use a Seq2seq model for symbolic integration and solving differential equations. We used this model for neural machine translation and bellow we will explain how the encoder and the decoder works in this task.

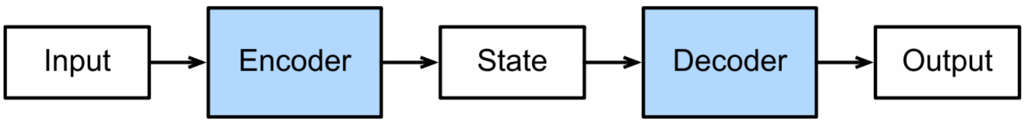


Figure 4.1: The Seq2Seq architecture

The encoder takes as input a variable-length sequence and then transforms it into a fixed length vector (hidden state), which contains all the information of the input encoded. The decoder takes this fixed length vector and produces a variable-length sequence based on this encoded vector [34].

4.1.1 Encoder

The encoder is one of the two basic components of Seq2Seq. Specifically the encoder transforms the input sequence to a fixed length vector containing all the necessary information. In neural machine translation the input of the encoder will be the tokenized sentence either in one-hot encoding or it will be the output of the embedding layer. Suppose we have an input sequence $x_1, x_2, ..., x_N$, where x_n is the n^{th} token of the sequence. A time step n the encoders transforms the input x_n and the hidden state h_{n-1} from the previous time step into a new hidden state h_n. We can describe this with the following function (eq. 4.1).

$$h_n = f(x_n, h_{n-1}) \tag{4.1}$$

In essence the encoder transforms the hidden states for all time steps into a context variable **c** (eq. 4.2).

$$\mathbf{c} = q(h_1, h_2, ..., h_N) \tag{4.2}$$

In most Seq2Seq models the context variable is the last hidden state ($\mathbf{c} = h_N$). This hidden state is then passed to the decoder (fig. 4.2).

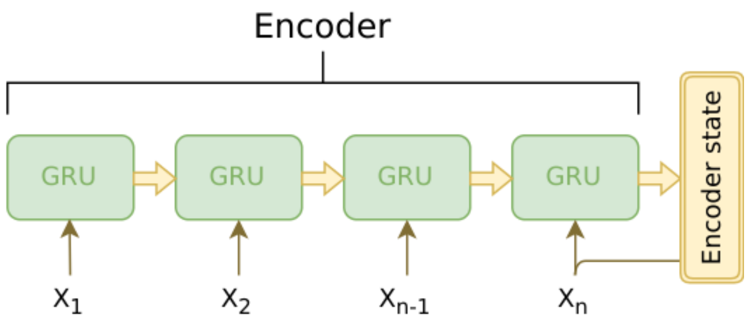


Figure 4.2: The encoder in a Seq2Seq model. [13]

4.1.2 Decoder

As mentioned before, the decoder uses the context variable **c** to produce the output sequence (fig. 4.3). Given the output sequence $y_1, y_2, ..., y_M$, where y_m is the m^{th} token of the sequence, the probability of the decoded output y_m results from the sequence $y_1, y_2, ..., y_{m-1}$ and the context variable **c** (eq. 4.3).

$$P(y_m \mid y_1, \ldots, y_{m-1}, \mathbf{c}) \tag{4.3}$$

This conditional probability can be modeled using a recurrent neural network. At each time step m of the output sequence the recurrent neural network takes y_{m-1} as input and its hidden state is initialized using the context variable $\mathbf{c}$. At each subsequent time step we use the previous output to make the next prediction until we reach the end of the sequence.

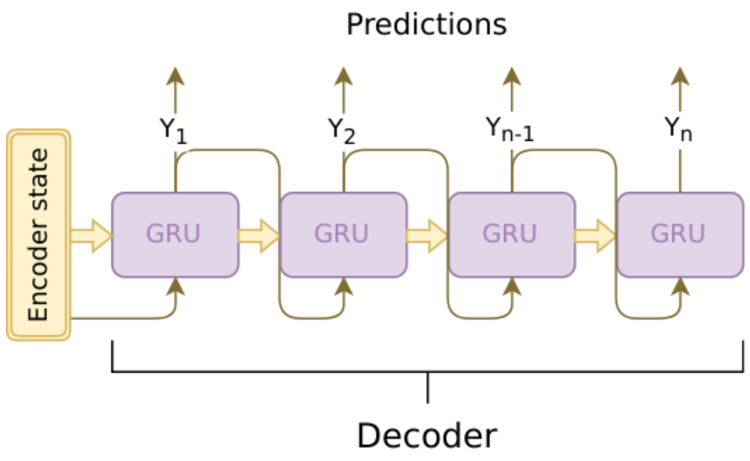


Figure 4.3: The decoder in a Seq2Seq model. [13]

Moreover, the decoder contains one more layer, a single Dense layer, with the softmax activation function to produce this conditional probability.

4.2 Attention mechanisms

Humans do not process all the information that is available in their environment but instead only focus in a fraction of this information. Using attention mechanisms we try to introduce this concept in machine learning algorithms and specifically in deep learning. In machine learning attention can be seen as a mechanism to give more weight to some inputs or in other words to introduce bias over some inputs.

In a Encoder-Decoder network used for neural machine translation from English to Italian the sentence ”Good life” is translated to ”Bella vita”. The network maps the word ”Good” to ”Bella” and ”life” to ”vita” implicitly, but when we use attention this mapping becomes more explicit.

The attention mechanism can be introduced in a network as an attention layer. The input in an attention layer is called a query and for every query the attention returns an output.

This output is based on the memory of the attention layer. This memory is encoded in the attention layer and contains some keys k and values v in pairs. The equation bellow, called score function, (eq. 4.4) is used to calculate the similarity between the query and the key, for each key $k_1, k_2, ..., k_n$.

$$a_i = \alpha(\mathbf{q}, \mathbf{k}_i). \tag{4.4}$$

The final attention output is calculated using the following equations (eq. 4.5,4.6)

$$\mathbf{b} = softmax(a) \tag{4.5}$$

$$\mathbf{o} = \sum_{i=1}^{n} b_i \mathbf{v}_i \tag{4.6}$$

The important part of the attention layer is the scoring function and there are many ways to implement. The two most common attentions are the dot-product attention [35], which uses the equation (eq. 4.7) as a scoring function and the additive attention [36], which uses the equation (eq. 4.8).

$$\alpha(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{K}^\top/\sqrt{d} \tag{4.7}$$

Where $\mathbf{Q} \in \mathbb{R}^{m\times d}$ contains m queries and $\mathbf{K} \in \mathbb{R}^{n\times d}$ has all the n keys.

$$\alpha(\mathbf{k}, \mathbf{q}) = \mathbf{v}^\top \text{tanh}(\mathbf{W}_k\mathbf{k} + \mathbf{W}_q\mathbf{q}). \tag{4.8}$$

Where $\mathbf{W}_k \in \mathbb{R}^{h\times d_k}$, $\mathbf{W}_q \in \mathbb{R}^{h\times d_q}$, $\mathbf{v} \in \mathbb{R}^h$ are learnable weights.

4.3 Our models and experiments

In this thesis we developed a Seq2Seq model (fig. 4.4) from scratch with the ability to define the type of recurrent layer used for the encoder and the decoder. The implementation of the model can be found in the following repository `https://github.com/chionkyr/seq2seq_nmt`. The type of recurrent layers supported are GRU layers, LSTM layers and bidirectional layers either GRU or LSTM. A bidirectional layer is a layer that consists of two recurrent layers working together, with the first layer processing the input tokens in order, while the second layer processing the input tokens in reverse order. This enables the layer to use information from both future and past to generate the current state.

In the experiments done, we create character level models and word level models, where each token is a character or a word respectively. Furthermore, we tried to encode the tokens using two different ways, one-hot vectors or embedding vectors.

All model have been tested with the same batch size of 512 and initial learning rate of 0.01, but we varied the size of the encoder and decoder between three values $128, 512, 1024$. Furthermore, the training of the model stopped if for the duration of 15 consecutive epochs there was no improvements and the learning rate was automatically reduced, to a minimum of 0.0001, if there was no improvements during the last 5 consecutive epochs.

An optimizer is an algorithm we use to minimize or maximize a cost function. In multi-layer neural networks an optimization algorithm is used in conjunction with backpropagation to update the weights properly. We used RMSprop [37] as an optimizer for our model, which is an adaptive learning rate optimizer, and usually used when training recurrent neural networks.

The models demonstrated in this thesis was made with the help of TensorFlow [38], an open-source Python library for machine learning.

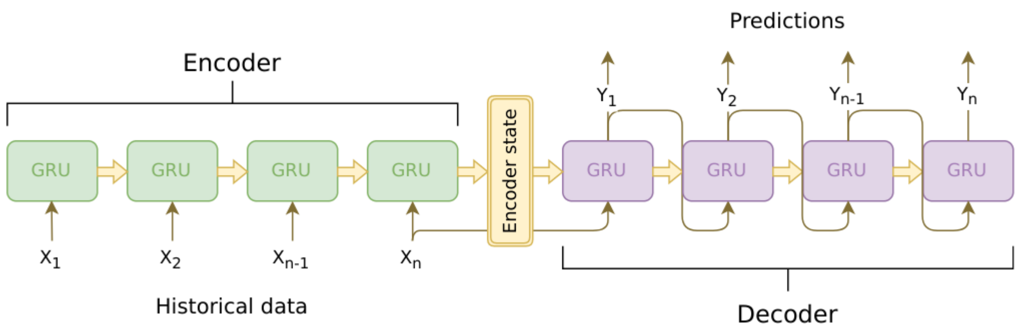


Figure 4.4: The Seq2Seq model. [13]

4.4 Character level tokenization

The first models we tested are character level models, where each token is a character. During the tokenization the source sentences underwent minimal prepossessing, keeping any numbers or punctuation that existed, but the characters were converted to ASCII.

4.4.1 One-hot encoding

GRU

As we can see (table 4.1) increasing the size of the network the BLEU score increased, which means the quality of the translations improved linearly. However the network with size 128 have BLEU scores below 0.3 and the translation quality will be low.

Seq2Seq using GRU and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.4760	0.8562	0.8116	0.7555	0.6135	0.8143	0.2507
512	0.2696	0.9182	0.7333	0.7968	0.4809	0.8589	0.3430
1024	0.1272	0.9644	0.8646	0.7935	0.4832	0.8700	0.3952

Table 4.1: Metrics of a Seq2Seq model using GRU and one-hot encoding.

LSTM

All LSTM networks showed poor performance in this dataset (table 4.2). Regardless of the size used all three of our networks achieved the same BLUE score of 0.27 and could not produce good translations.

Seq2Seq using LSTM and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.4718	0.8578	0.8392	0.7471	0.6255	0.8099	0.2729
512	0.3244	0.9015	0.7929	0.7718	0.5098	0.8470	0.2755
1024	0.2415	0.9274	0.7274	0.8033	0.5315	0.8499	0.2796

Table 4.2: Metrics of a Seq2Seq model using LSTM and one-hot encoding.

Bidirectional GRU

Using a bidirectional GRU network for our encoder greatly improved the BLEU scores (table 4.3) for all network sizes however this improvements came with a increase in training time. The biggest network with size of 1024 achieved a BLUE score of 0.4859 which means the translations produced have acceptable quality.

Seq2Seq using bidirectional GRU and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.3193	0.9034	0.7134	0.7945	0.5024	0.8499	0.3370
512	0.1263	0.9644	0.8529	0.7921	0.4563	0.8753	0.4175
1024	0.0039	0.9997	1.1517	0.7914	0.5592	0.8841	0.4859

Table 4.3: Metrics of a Seq2Seq model using bidirectional GRU and one-hot encoding.

Bidirectional LSTM

A bidirectional LSTM network showed the same improvement (table 4.4) over the simple LSTM. However the bi bidirectional achieve better BLEU scores with less training time and faster inference time.

Seq2Seq using bidirectional LSTM and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.2457	0.9273	0.6358	0.8159	0.4598	0.8645	0.3947
512	0.0755	0.9820	0.8278	0.8062	0.5097	0.8704	0.4171
1024	0.0247	0.9955	0.8478	0.8209	0.5233	0.8805	0.4639

Table 4.4: Metrics of a Seq2Seq model using bidirectional LSTM and one-hot encoding.

4.4.2 Embeddings

For all tests executed in this section the size of the embedding vectors was set to 30. This number was chosen due to the fact that our unique input and output tokens was approximately 80.

GRU

We can see (table 4.5) that increasing the size of the network the BLEU score increased, however, the evolution of this growth was approximately 10%. All networks achieved average BLUE scores with the highest value being less that 0.35.

Seq2Seq using GRU and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.4066	0.8775	0.7541	0.7771	0.5676	0.8296	0.2885
512	0.2791	0.9150	0.7381	0.7953	0.4879	0.8559	0.3083
1024	0.1841	0.9450	0.7841	0.8003	0.4807	0.8661	0.3404

Table 4.5: Metrics of a Seq2Seq model using GRU and embeddings.

LSTM

Similar to LSTM models used with the one-hot encoding, in this case we also observed bad performance with LSTM networks and all produced bad translations and achieved bad BLUE scores (table 4.6).

Seq2Seq using LSTM and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.4009	0.8789	0.7506	0.7751	0.5637	0.8291	0.2867
512	0.3167	0.9029	0.8032	0.7678	0.5094	0.8459	0.2318
1024	0.2329	0.9289	0.7669	0.7894	0.4957	0.8578	0.2539

Table 4.6: Metrics of a Seq2Seq model using LSTM and embeddings.

Bidirectional GRU

A bidirectional GRU network improved dramatically BLEU scores here as well (table 4.7). The biggest network achieved a BLUE score of 0.3936 that indicates average translation quality.

Seq2Seq using LSTM and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.3182	0.9032	0.7066	0.7942	0.4783	0.8552	0.3273
512	0.1701	0.9502	0.7670	0.7993	0.4582	0.8697	0.3902
1024	0.0855	0.9781	0.9174	0.7927	0.4921	0.8725	0.3936

Table 4.7: Metrics of a Seq2Seq model using bidirectional GRU and embeddings.

Bidirectional LSTM

A bidirectional LSTM showed dramatic improvement, especially the bigger one (table 4.8). It managed to reach a BLUE score of 0.4711 which is the biggest so far.

Seq2Seq using LSTM and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.3444	0.8955	0.6990	0.7929	0.5124	0.8450	0.2862
512	0.1683	0.9504	0.7942	0.7953	0.4794	0.8667	0.3159
1024	0.0089	0.9993	0.8839	0.8233	0.5068	0.8889	0.4711

Table 4.8: Metrics of a Seq2Seq model using bidirectional LSTM and embeddings.

4.5 Word level tokenization

Next we experimented with word level models, where each token is a word. During the tokenization the source sentences underwent minimal prepossessing, keeping any numbers or punctuation that existed, but all the characters were converted to ASCII.

4.5.1 One-hot encoding

GRU

We can see (table 4.9) increasing the size of the network causes the BLEU score to increase almost linearly. All networks achieved high blue scores, even the smallest one.

Seq2Seq using GRU and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.6541	0.8994	2.2979	0.7019	1.5255	0.7669	0.4251
512	0.3574	0.9445	2.2148	0.7179	1.3636	0.7916	0.4935
1024	0.2025	0.9705	2.2636	0.7240	1.3220	0.8027	0.5342

Table 4.9: Metrics of a Seq2Seq model using GRU and one-hot encoding.

LSTM

In contrast to character level LSTM models, the networks here reached very high BLUE scores (table 4.10), all of them achieving more than 0.56 BLUE score, producing very good translations.

Seq2Seq using LSTM and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.4098	0.9398	2.0789	0.7242	1.2400	0.8074	0.5655
512	0.2460	0.9628	2.1114	0.7234	1.2516	0.8069	0.5664
1024	0.1009	0.9866	2.2290	0.7250	1.2732	0.8124	0.5674

Table 4.10: Metrics of a Seq2Seq model using LSTM and one-hot encoding.

Bidirectional GRU

The usage of bidirectional networks in the encoder did not produce better results (table 4.11) than simple using of a GRU network. Moreover the training time increased greatly and the test using 1024 neurons could not complete due to a limitation of the available resources.

Seq2Seq using bidirectional GRU and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.4530	0.9308	2.0371	0.7102	1.3117	0.7910	0.4880
512	0.1935	0.9719	2.0379	0.7208	1.2388	0.8082	0.5380

Table 4.11: Metrics of a Seq2Seq model using bidirectional GRU and one-hot encoding.

Bidirectional LSTM

A bidirectional LSTM encoder, with 128 neurons, produced the best BLEU score in these experiments (table 4.12). However, an increase in the size of the encoder produced sub par results. Moreover, the training time was disproportional bigger and we could not finish the training of the model with the 1024 neurons.

Seq2Seq using bidirectional LSTM and one-hot encoding							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.3428	0.9480	1.9449	0.1250	1.1654	0.8112	0.5866
512	0.1826	0.9733	2.1549	0.7015	1.1589	0.8157	0.4251

Table 4.12: Metrics of a Seq2Seq model using bidirectional LSTM and one-hot encoding.

4.5.2 Embeddings

For the tests done with word level tokenization the size of embedding vectors was set to 300 because the max size of our vocabulary was approximately 23000 [].

GRU

The word level GRU models produced slightly better results (table 4.13) from the corresponding character level GRU model, with the best model reaching a BLEU score of 0.54.

Seq2Seq using GRU and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.3924	0.9359	2.0540	0.7207	1.3092	0.7917	0.4722
512	0.1969	0.9678	1.9853	0.7372	1.1901	0.8127	0.5193
1024	0.0683	0.9917	2.0578	0.7416	1.1764	0.8208	0.5459

Table 4.13: Metrics of a Seq2Seq model using GRU and embeddings.

LSTM

Similar to the GRU models the LSTM models produced comparable BLEU scores (table 4.14), although slightly lower. The best model was the middle one, with 512 neuron, achieving a BLEU score of 0.55.

Bidirectional GRU

The bidirectional GRU models produced similar results (table 4.15) with the unidirectional GRU models. In contrast to the bidirectional GRU models using one-hot encoding, the models using embeddings did not present a problem during training and all models was able to be trained.

Seq2Seq using LSTM and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.2697	0.9556	1.8182	0.7376	1.0951	0.8202	0.5699
512	0.1142	0.9818	1.9015	0.7368	1.1262	0.8205	0.5523
1024	0.1333	0.9777	1.9661	0.7302	1.2105	0.8080	0.5205

Table 4.14: Metrics of a Seq2Seq model using LSTM and embeddings.

Seq2Seq using bidirectional GRU and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.3827	0.9334	1.9373	0.7258	1.2474	0.7970	0.4697
512	0.0867	0.9885	2.0074	0.7322	1.1795	0.8143	0.5287
1024	0.0435	0.9947	2.1140	0.7344	1.2173	0.8171	0.5281

Table 4.15: Metrics of a Seq2Seq model using bidirectional GRU and embeddings.

Bidirectional LSTM

Similar to the character level tokenization, the bidirectional LSTM models produced the best BLEU scores (table 4.16). The highest BLEU score reached is 0.5934, which is the highest score considering all models.

Seq2Seq using bidirectional LSTM and embeddings							
RNN Size	Train loss	Train acc.	Val loss	Val acc.	Test loss	Test acc.	BLUE4
128	0.1771	0.9696	1.6653	0.7487	1.0129	0.8315	0.5893
512	0.0428	0.9948	1.9046	0.7377	1.0938	0.8266	0.5685
1024	0.0157	0.9987	1.9369	0.7469	1.0930	0.8347	0.5934

Table 4.16: Metrics of a Seq2Seq model using bidirectional LSTM and embeddings.

4.6 Reviewing the results

Based on the results from the previous section, we used the models that achieved the best BLEU scores in order to observe how these models act given some inputs. We tried to categorize the produced sentences so as to understand the particularities of each model.

4.6.1 One-hot character level model

The best model, according to BLEU score, is the bidirectional GRU with 1024 neurons. Below we demonstrate some typical sentences, with BLEU score more than 0.3, and their translations.

Low BLEU score but understandable translation

```
Source:      I know that you used to live in Boston.
Target:      So che lei una volta viveva a Boston.
Predicted:   So che hai venuto Boston.
BLEU score:  0.36
```

This is a sentence that although it has low BLEU score, the source sentence's meaning is captured and the goal, which is to transfer the meaning to the target language, is achieved.

Missing words

```
Source:      Tom looked genuinely interested.
Target:      Tom sembrava sinceramente interessato.
Predicted:   Tom sembrava interessato.
BLEU score:  0.57
```

```
Source:      Tom isn't afraid of death.
Target:      Tom non ha paura della morte.
Predicted:   Tom non ha paura.
BLEU score:  0.47
```

In the above examples we can see that in the predicted sentences some words are missing. For this reason the meaning of the sentences may be affected negatively, but in some exceptional cases the words removed may not have a vital role in the sentence.

Spelling mistakes

```
Source:      I think we need to find a new babysitter.
Target:      Penso che dobbiamo trovare una nuova babysitter.
Predicted:   Penso che bbbbiamo energiare di piaggua.
BLEU score:  0.40
```

```
Source:     Tom is already there, isn’t he?
Target:     Tom e gia li, vero?
Predicted:  Tom e gia le, vero?
BLEU score: 0.85
```

Due to the nature of the a character level model, spelling mistakes are quite often. This happens because the network cannot learn the meaning of the words, but instead learns a sequence of characters. However, as we can see above this may not cause serious problem in some cases.

Perfect translation

```
Source:     I’m not worried about Tom.
Target:     Non sono preoccupato per Tom.
Predicted:  Non sono preoccupato per Tom.
BLEU score: 1.00
```

Finally, this model can produce perfect translations even if sentences are unknown and do not belong, as is, in the training dataset. However, this is quite rare.

4.6.2 One-hot word level model

According to the previous section the best model, that uses one-hot encoding and word level tokenization, is a bidirectional LSTM model with 128 neurons. Below we present some example with the most common errors.

High BLEU score but wrong translation

```
Source:     Tom wanted something cold to drink .
Target:     Tom voleva qualcosa di freddo da bere .
Predicted:  Tom voleva qualcosa da bere bere .
BLEU score: 0.75
```

```
Source:     We ’ ve made a very interesting discovery .
Target:     Abbiamo fatto una scoperta molto interessante .
Predicted:  Abbiamo fatto una cosa molto interessante .
BLEU score: 0.83
```

Something we noticed in this model is high BLEU scores on some sentences, but the meaning of theses sentences is completely off. This happens due to the fact that a crucial, to the meaning, word is missing from the predicted translation or it is replaced with another irrelevant word. This may indicate a disadvantage to the BLEU algorithm.

Missing words

```
Source:     I ′ ve already started reading that book .
Target:     Ho gia iniziato a leggere quel libro .
Predicted:  Ho gia iniziato a libro .
BLEU score: 0.59

Source:     I wish I hadn ′ t looked out the window .
Target:     Vorrei non aver guardato fuori dalla finestra .
Predicted:  Vorrei non avere la finestra .
BLEU score: 0.50
```

As we can saw, in the one-hot word level model, we observed again the issue with missing words. This may have a negative effect to the meaning of the sentence but in some other cases the meaning of the translation is preserved.

Rare words

```
Source:     That doesn ′ t seem very hygienic .
Target:     Non sembra molto igienico .
Predicted:  Non sembra molto molto .
BLEU score: 0.66

Source:     He was born in the 19th century .
Target:     E nato nel diciannovesimo secolo .
Predicted:  E nato nel nel .
BLEU score: 0.24
```

Another problem we observed in the translations, is that rare words are difficult to be predicted by the model. These rare words are usually replaced by the preceding word of the sentence. For this reason, these sentences are not understandable.

Perfect translation

```
Source:     I need a pair of new shoes .
Target:     Ho bisogno di un nuovo paio di scarpe .
Predicted:  Ho bisogno di un nuovo paio di scarpe .
BLEU score: 1.00
```

One positive aspect of this model is that it can also produce perfect translations like the one-hot character level model.

4.6.3 Embedding character level model

The model that reached the best BLEU score (0.47) using embeddings and character level tokenization is a bidirectional LSTM model with 1024 neuron. The following sentences represent some translations produced by the model.

Substitution with a derivative word

```
Source:     Tom was able to pass the exam.
Target:     Tom era in grado di superare l'esame.
Predicted:  Tom e stato in grado di superare l'esame.
BLEU score: 0.81
```

```
Source:     Tom hopes that Mary won't do that.
Target:     Tom spera che Mary non lo fara.
Predicted:  Tom spera che Mary non lo fa.
BLEU score: 0.91
```

In the sentences above the translation meaning may differ from the target translation, but the word used in the predicted translation is a derivative of the target word. The meaning of sentences is captured, to some extent, although not completely.

Substitution with a irrelevant word

```
Source:     I wonder if Tom is still depressed.
Target:     Mi chiedo se Tom sia ancora depresso.
Predicted:  Mi chiedo se Tom sia ancora sposato.
```

```
BLEU score: 0.78

Source:     Tom couldn't have done this without me.
Target:     Tom non avrebbe potuto fare questo senza di me.
Predicted:  Tom non avrebbe potuto fare questo per me.
BLEU score: 0.79
```

A common occurrence in this model is to produce a correct translation except one single word. This incorrect word changes completely the meaning of the sentence. However, most of these sentence produce high BLEU score and this may skew the overall score higher.

Incorrect translations

```
Source:     She has a son everybody loves.
Target:     Ha un figlio che amano tutti.
Predicted:  Ha un giornatore con veloce.
BLEU score: 0.23

Source:     Could you talk a little slower?
Target:     Potresti parlare un po' piu lentamente?
Predicted:  Potresti andare a piedi fino al biuo?
BLEU score: 0.27
```

Another interesting remark about this model is that despite the high BLEU score it achieved, a big portion of the predicted translations have a very low BLUE score. Moreover, we can observe that these translations are completely wrong, which is reflected on each sentence's scores.

Perfect translation

```
Source:     The two kissed.
Target:     I due si sono baciati.
Predicted:  I due si sono baciati.
BLEU score: 1.00
```

This model is also capable of producing perfect translations like the previous one-hot models.

4.6.4 Embedding word level model

The best BLEU score (0.59) using embeddings and word level tokenization was achieved by a bidirectional LSTM model with 1024 neuron. Some example sentences are presented bellow in order to demonstrate some common errors.

Substitution with a derivative word

```
Source:      I ′ ll do your homework for you .
Target:      Faro i tuoi compiti per te .
Predicted:   Faro i miei compiti per te .
BLEU score:  0.83
```

In this model we also observed that some words got replaced by derivatives. In this instance ‘tuoi‘ got replaced by ‘miei‘, where both are possessive pronouns with the only difference being the grammatical person.

Substitution with a irrelevant word

```
Source:      I knew Tom wouldn ′ t enjoy the party .
Target:      Sapevo che a Tom non sarebbe piaciuta la festa .
Predicted:   Sapevo che Tom non sarebbe stato festa .
BLEU score:  0.67
```

```
Source:      You like to hunt , don ′ t you ?
Target:      Ti piace cacciare , vero ?
Predicted:   Ti piace disegnare , vero ?
BLEU score:  0.71
```

In this case words are replaced by other words with a completely different meaning. These sentences achieve high BLEU score but the overall meaning is quite different. However, there are some examples where these words are essential for the meaning of the sentence but in rare cases these words may not effect the meaning negatively.

Missing words

```
Source:      I don ′ t want to be involved in this affair .
```

```
Target:      Non voglio essere coinvolto in questo affare .
Predicted:   Non voglio essere coinvolto .
BLEU score:  0.55
```

```
Source:      We hope it doesn ’ t happen again .
Target:      Speriamo che non capiti ancora .
Predicted:   Speriamo che non capiti .
BLEU score:  0.74
```

We observe, in this model, that many sentences got translated only in part. Most of them only have a few of the first words translated correctly the rest of the sentence is missing. This may have a negative effect to the meaning of the sentence but in some other cases the meaning of the translation is preserved.

Rare words

```
Source:      Tom is a very good dancer , isn ’ t he ?
Target:      Tom e un ballerino molto bravo , vero ?
Predicted:   Tom e un molto molto , vero ?
BLEU score:  0.57
```

In addition to these cases there are some sentences that contains rare words. The model produces predictions that do not approach the target translation. Rare words are difficult to be predicted due to their low presence in the training data.

Perfect translation

```
Source:      I really like to read .
Target:      Mi piace davvero leggere .
Predicted:   Mi piace davvero leggere .
BLEU score:  1.00
```

```
Source:      Tom often helps me in the garden .
Target:      Tom mi aiuta spesso in giardino .
Predicted:   Tom mi aiuta spesso in giardino .
BLEU score:  1.00
```

Finally, the model managed to produce perfect translations, more than any other model, and approximately 10% of the sentences achieved a BLEU score of 1.

Chapter 5

Conclusions

In this chapter we will summarize our research on the neural machine translation domain and the results from our experiments using Seq2Seq models with different hyperparameters and architectures.

5.1 Summary and conclusions

The Seq2Seq architecture is a very interesting architecture that enables us to predict a sequence given another sequence. In this thesis we used this architecture for the task of neural machine translation. This task requires many choices that effect how the model works and involves some pre-processing steps. Character level or word level tokenization are two categories that we tested in order to observe the advantages and disadvantages of each method. Furthermore, we explored two different encoding techniques for the created tokens, either using embedding vectors or one-hot vectors.

After some extensive testing we came to some interesting conclusions. The best Seq2Seq model, according to BLEU score, was using word level tokenization, embedding vectors and bidirectional LSTM networks for the encoder and the decoder with 1024 neuron. This model achieved 0.5934 BLEU score and that means very high quality, adequate, and fluent translations.

Generally speaking the average BLEU score of the character level models was approximately 0.2 lower than word level models, but this may not reflect the real performance of the model. Due to the fact that character level models predict characters, spelling mistakes are not a rare occurrence. BLEU score calculates n-grams of words and for this reason spelling mistakes that change even a single letter in a word would produce a lower score. On the other hand word level models face problems with rare words, because there are limited usages of these words our the dataset. This probably can be solved by using a bigger dataset with more diverse vocabulary. Moreover, due to the pre-processing required by word level models spaces are introduced in the translated sentence between words and stop-words. However, our final observation is using word level tokenization produces better translations in comparison to character level tokenization , but word level tokenization may not be not be always feasible especially with huge vocabularies.

Comparing the two different encoding techniques we did not observe many differences in the translations produced. However the models that used embedding vectors was able to learn some relations between words. This can result in the model replacing the original words with derivative words without making big difference to the meaning.

Analyzing further the models we can draw conclusions about the performance of each recurrent network type and the encoding used. Simple GRU and LSTM networks had low BLEU score and only achieved average scores using the highest number of neuron, irrelevant of the encoding used. The best performance was achieved using bidirectional models as expected. Moreover, the size of the model influenced the scores directly but not in all cases. Usually, the best model was the one with the highest number of neuron except when using word level tokenization and one-hot encoding where the model with 128 neurons performed the best. The complexity of the model and thus the training time and the inference time is highly related to the size of the networks and the type of the encoding used. As we can see, in the following tables (5.1, 5.2), doubling the size of the model exponentially increases the number of it's parameters. More specifically, in the model that uses one-hot encoding the vocabulary size has a huge impact on the number of parameters, while using embedding the created model is not effected that much.

5.2 Future prospects

In this thesis we described the ordinary Seq2Seq model and it's applications in neural machine translation. However, we can at temped to improve it in various way. One simple way is to use multiple layer in the encoder and decoder, however this may not impact BLEU scores. Furthermore, we can implement attention in a Seq2Seq model. Using attention, the model will

Parameters versus size				
Size	LSTM - CL	BI-LSTM - CL	LSTM - WL	BI-LSTM - WL
128	226,130	583,250	22,703,010	45,513,890
512	2,477,138	7,051,346	92,315,298	186,704,546
1024	9,148,498	26,685,522	188,801,698	385,968,802

Table 5.1: Complexity of a Seq2Seq model using LSTM or BI-LSTM and one-hot encoding. CL: Character Level, WL: Word Level

Parameters versus size				
Size	LSTM - CL-30	BI-LSTM - CL-30	LSTM - WL-300	BI-LSTM - WL-300
128	178,284	482,668	14,903,754	18,443,978
512	2,271,084	6,634,348	26,704,074	44,010,698
1024	8,731,500	25,846,636	46,107,850	89,109,706

Table 5.2: Complexity of a Seq2Seq model using LSTM or BI-LSTM and embedding encoding. CL: Character Level, WL: Word Level

be able produce better and more accurate translations without increasing the complexity. In addition, we can also change the way the predicted translated sentence is created. To create a translation we have to find the best combination of tokens. However, these tokens will not always be the ones with the highest probability and to solve this problem we can introduce the beam search algorithm. Finally, to deal with the neural machine translation task, we can introduce another another family of models called transformer models [39]. These models are based on attention and do not require to process the data in order. This can lead to faster training time and better results.

Bibliography

[1] Deep learning: The latest trend in ai and ml. https://www.qubole.com/blog/deep-learning-the-latest-trend-in-ai-and-ml/. Date accessed: 25-02-2021.

[2] Wikipedia linear regression. https://en.wikipedia.org/wiki/Linear_regression. Date accessed: 25-02-2021.

[3] Wikipedia decision tree learning. https://en.wikipedia.org/wiki/Decision_tree_learning. Date accessed: 25-02-2021.

[4] Wikipedia cluster analysis. https://en.wikipedia.org/wiki/Cluster_analysis. Date accessed: 25-02-2021.

[5] Wikipedia support-vector machine. https://en.wikipedia.org/wiki/Support-vector_machine. Date accessed: 25-02-2021.

[6] Wikipedia artificial neural network. https://en.wikipedia.org/wiki/Artificial_neural_network. Date accessed: 25-02-2021.

[7] The perceptron. https://towardsdatascience.com/the-perceptron-3af34c84838c. Date accessed: 25-02-2021.

[8] Knet - multilayer perceptrons. https://towardsdatascience.com/the-perceptron-3af34c84838c. Date accessed: 25-02-2021.

[9] Explaining recurrent neural networks. https://www.bouvet.no/bouvet-deler/explaining-recurrent-neural-networks. Date accessed: 25-02-2021.

[10] Wikipedia long short-term memory. https://en.wikipedia.org/wiki/Long_short-term_memory. Date accessed: 25-02-2021.

[11] Wikipedia gated recurrent unit. `https://en.wikipedia.org/wiki/Gated_recurrent_unit`. Date accessed: 25-02-2021.

[12] Tab-delimited bilingual sentence pairs. `http://www.manythings.org/anki/`. Date accessed: 25-02-2021.

[13] seq2seq-pytorch. `https://github.com/sooftware/seq2seq`. Date accessed: 25-02-2021.

[14] Warren Weaver. Translation. In William N. Locke and A. Donald Boothe, editors, *Machine Translation of Languages*, pages 15–23. MIT Press, Cambridge, MA, 1949/1955. Reprinted from a memorandum written by Weaver in 1949.

[15] Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. A statistical approach to machine translation. *Comput. Linguist.*, 16(2):79–85, June 1990.

[16] Sergei Nirenburg. Knowledge-based machine translation. *Machine Translation*, 4(1):5–24, 1989.

[17] Makoto Nagao. A framework of a mechanical translation between japanese and english by analogy principle. In *Proc. of the International NATO Symposium on Artificial and Human Intelligence*, page 173–180, USA, 1984. Elsevier North-Holland, Inc.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[19] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[20] C. Van Der Malsburg. Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. In Günther Palm and Ad Aertsen, editors, *Brain Theory*, pages 245–248, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

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

[22] Rumelhart, David E, McClelland, and James L. *Parallel distributed processing : explorations in the microstructure of cognition*. Cambridge, Mass. : MIT Press, 1986.

[23] Stanford cs 230 - deep learning - recurrent neural networks cheatsheet. `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks`. Date accessed: 25-02-2021.

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

[25] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[26] Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. *CoRR*, abs/1805.04908, 2018.

[27] Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

[28] Tab-delimited bilingual sentence pairs. `http://www.manythings.org/anki/`. Date accessed: 25-02-2021.

[29] Google machine learning course - embeddings. `https://developers.google.com/machine-learning/crash-course/embeddings/video-lecture`. Date accessed: 25-02-2021.

[30] 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 of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[31] Automl - evaluating models. `https://cloud.google.com/translate/automl/docs/evaluate`. Date accessed: 25-02-2021.

[32] 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, Lukasz 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.

[33] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *CoRR*, abs/1912.01412, 2019.

[34] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Corwin Press, 0.16.1 edition, 2019.

[35] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[36] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[37] Neural networks for machine learning - lecture 6a. `https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. Date accessed: 25-02-2021.

[38] Tensorflow. `https://www.tensorflow.org/`. Date accessed: 25-02-2021.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.