University of Brasília

Exact Sciences Institute
Computer Science Department

Natural Language Inference using models based on Long-Short Term Memory: A Comparative Study

Felipe Xavier Barbosa da Silva

Monograph submitted in partial fullfilment of
the requirements to Bachelor Degree in Computer Engineering

Advisor
Prof. Dr. Vinicius Ruela Pereira Borges

Brasília
2022

University of Brasília

Exact Sciences Institute
Computer Science Department

Natural Language Inference using models based on Long-Short Term Memory: A Comparative Study

Felipe Xavier Barbosa da Silva

Monograph submitted in partial fullfilment of
the requirements to Bachelor Degree in Computer Engineering

Prof. Dr. Vinicius Ruela Pereira Borges (Advisor)
CIC/UnB

Prof. Dr. Luís Paulo Faina Garcia
CIC/UnB

Prof. Dr. Marcus Vinicius Lamar
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordinator of Bachelor Degree in Computer Engineering

Brasília, May 11th, 2022

Acknowledgements

I would like to thank my advisor, Prof. Dr. Vinícius Ruela Pereira Borges, for his contributions to this research and to the writing of this thesis. I am also grateful to my family, for their continuous support, and to my friends, for helping me go through this path of my life.

Abstract

Natural Language Inference (NLI) is the task of automatically classifying the relation between two texts as one of inference or not, with one of these texts being usually called the premise, and the other the hypothesis. NLI models have improved significantly in the last few years due to the increasing performance of deep learning systems. However, they have also grown in complexity, which led to long training times. As such, hyperparameter tuning is rarely performed within the field of NLI. Additionally, many new papers don't provide a sufficient justification for the hyperparameters used. In this work, the effects of a large-scale hyperparameter tuning on simple LSTM models for the NLI task were analyzed with the goal of determining if this optimization could be worth the computational cost. It was also studied if it could be a valid strategy to reuse hyperparameters optimized for a different dataset, as some works in the literature have done. The experiments were conducted on three datasets: ASSIN 2, SICK, and a reduced version of SNLI, named RSNLI. In all three datasets, a significant variation in performance caused by hyperparameter tuning was verified, reaching as high as 15.4% on the ASSIN 2 dataset, 4.7% on the SICK dataset, and 17.5% on the RSNLI dataset. Additionally, the analysis of the average accuracy of each hyperparameter combination showed a correlation ranging from 0.520 to 0.744 for the performance of the same hyperparameters in different datasets. The conclusion of this research is that the optimization of hyperparameters had a significant effect on simple LSTM models, but it remains to be tested in-depth on more complex systems.

Keywords: natural language inference, hyperparameter tuning, LSTM, textual entailment

Resumo

Inferência de Linguagem Natural é a tarefa de classificar automaticamente a relação entre dois textos como sendo de inferência ou não, com um desses textos sendo geralmente chamado de premissa, e o outro de hipótese. Modelos de Inferência de Linguagem Natural melhoraram significativamente nos últimos anos devido ao desempenho cada vez melhor de sistemas de aprendizado profundo. Contudo, eles também ficaram mais complexos, o que levou a tempos de treinamento longos. Por causa disso, a otimização de hiperparâmetros é raramente feita dentro da área de Inferência de Linguagem Natural. Ademais, muitos dos novos artigos não justificam as suas escolhas de hiperparâmetros. Neste trabalho, foram analisados os efeitos de uma otimização de hiperparâmetros de larga escala em modelos de LSTM simples para a tarefa de Inferência de Linguagem Natural, com o objetivo de determinar se essa otimização é viável considerando o tempo de processamento computacional. Além disso, foi estudado se seria uma estratégia válida reusar parâmetros que foram otimizados para outro conjunto de dados, como alguns trabalhos na literatura fizeram. Os experimentos foram conduzidos em três conjuntos de dados: ASSIN 2, SICK, e uma versão reduzida do SNLI, RSNLI. Em todos os três conjuntos de dados, foi verificada uma variação significativa de desempenho causada pela otimização de hiperparâmetros, chegando a 15,4% no ASSIN 2, 4,7% no SICK, e 17,5% no RSNLI. Além disso, a análise da acurácia média de cada combinação de hiperparâmetro mostrou uma correlação de $0,520$ a $0,44$ para o desempenho dos mesmos hiperparâmetros em conjuntos de dados diferentes. A conclusão desta pesquisa é que a otimização de hiperparâmetros teve um efeito significativo em modelos de LSTM simples, mas ainda resta testá-la em sistemas mais complexos.

Palavras-chave: inferência de linguagem natural, otimização de hiperparâmetros, LSTM

Contents

1 Introduction **1**

1.1 Goals . . . 3

1.2 Structure . . . 4

2 Theoretical background **5**

2.1 The task of Natural Language Inference . . . 5

2.2 Principles of machine learning . . . 6

2.3 Artificial Neural Networks . . . 6

2.3.1 Artificial neurons . . . 7

2.3.2 Neuron layers . . . 8

2.3.3 Loss function . . . 9

2.3.4 Epochs and Early Stopping . . . 11

2.4 Recurrent Neural Networks . . . 11

2.5 Convolutional Neural Networks . . . 14

2.6 Word Embeddings . . . 16

2.7 Evaluation strategies . . . 16

2.8 Final considerations . . . 18

3 Related works **19**

3.1 Datasets . . . 19

3.2 Hyperparameter Tuning . . . 21

3.3 Contributions of this thesis . . . 22

4 Methodology **23**

4.1 Dataset . . . 23

4.2 Preprocessing . . . 24

4.3 Embedding . . . 25

4.4 Model . . . 25

4.5 Method of Analysis . . . 28

4.6 Final considerations . . . 28

5 Results **29**
5.1 Experiment Setup 29
5.2 ASSIN 2 30
5.3 SICK 32
5.4 RSNLI 35
5.5 Comparison of results across datasets 37
5.6 Comparison with other models 41
5.6.1 ASSIN 2 41
5.6.2 SICK 43
5.6.3 SNLI 43
5.7 Final Considerations 43

6 Conclusion **44**
6.1 Future work 45

References **46**

List of Figures

2.1 Visual abstraction of an artificial neuron. Based on the illustration in [1]. . 7
2.2 An example of artificial neural network with three layers and eight neurons. Based on the illustration in [2]. 8
2.3 Elman's recurrent network, as depicted in the original work [3]. Dotted connections between layers represent learnable weights. The connection from the hidden layer to the context layer has a fixed weight of 1. 12
2.4 Illustration of how RNNs can process sequences. Adapted from [4]. 13
2.5 Common representation of the LSTM architecture. Each sigmoid activation function represents one of the gates of the LSTM, from left to right: Forget Gate, Input Gate, and Output Gate. Lines joining without an operation indicates concatenation. Based on the illustration by [5]. 14
2.6 A 2×2 filter being applied to a 3×3 matrix. 15
2.7 CBOW and Skip-gram algorihms. Based on the illustration by [6]. 17

4.1 Diagram of the methodology used in this work. 23
4.2 Simple LSTM model. 26
4.3 CNN-LSTM model. 26
4.4 Dual LSTM model. Note that the green and the blue LSTMs are different. 27
4.5 The Concatenation LSTM model. 27

5.1 Complete results for the ASSIN 2 dataset. 30
5.2 15% best results on the ASSIN 2 dataset. 31
5.3 15% worst results on the ASSIN 2 dataset. 32
5.4 Complete results for the SICK dataset. 33
5.5 15% best results on the SICK dataset. 34
5.6 15% worst results on the SICK dataset. 34
5.7 Complete results for the RSNLI dataset. 36
5.8 15% best results on the RSNLI dataset. 36
5.9 15% worst results on the RSNLI dataset. 37
5.10 Comparison between the results on ASSIN 2 and SICK. 38

5.11 Comparison between the results on ASSIN 2 and RSNLI. 39
5.12 Comparison between the results on SICK and RSNLI. 39
5.13 Heatmap of the average performance for each learning rate-hidden layer size combination in the ASSIN 2 dataset. 40
5.14 Heatmap of the average performance for each learning rate-hidden layer size combination in the SICK dataset. 41
5.15 Heatmap of the average performance for each learning rate-hidden layer size combination in the RSNLI dataset. 42

List of Tables

1.1 Examples of premise-hypothesis pairs from the SNLI corpus. 2

4.1 Label distribution in the ASSIN 2 dataset. 24
4.2 Label distribution in the SICK dataset. 24
4.3 Label distribution in the RSNLI dataset. 25

5.1 Results for each model-embedding combination on the ASSIN 2 dataset. . 33
5.2 Results for each model-embedding combination on the SICK dataset. . . . 35
5.3 Results for each model-embedding combination on the RSNLI dataset. . . 38
5.4 Spearman's rank correlation coefficient for each dataset-model combination. The numbers between parenthesis are the p-values of the respective coefficients. 40
5.5 Results on the ASSIN 2 compared with other models. 42

Chapter 1

Introduction

Natural Language Understanding (NLU) is a field of Natural Language Processing (NLP) that focuses on topics related to the understanding of human writing by artificial intelligence. It has multiple applications in other tasks within NLP, such as machine translation [7] and question answering [8], that rely on a proper interpretation of language to perform well. Natural Language Inference (NLI), also called Textual Entailment, is the task of identifying the relation between two texts, which are often called the premise and the hypothesis. This relation is usually classified as either entailment, if the premise implies in the hypothesis, or neutral otherwise. Some versions of this task further split the label neutral into two labels: contradiction, if the premise implies that the hypothesis is false, and neutral if there is no relation between the texts. In general, the objective is to determine the label based on not only context but also world knowledge. For example, the premise "The door is locked" should entail the hypothesis "The door is closed", because it is understood by most humans that a door that is locked is also closed, even if this information is not present in the premise. Note that the relation is not bidirectional: "The door is closed" does not entail "The door is locked".

NLI models can also be applied, sometimes directly, to other tasks in NLP. For example, in the task of machine translation, it is necessary for the original text, which could be considered to be the premise, to entail the translated text, which could be considered to be the hypothesis, and vice-versa. Similarly, an NLI model adapted for two languages could be used to verify if a translation is correct. Poliak et al. [9] used an NLI classifier to determine the extent to which a machine translation model preserved certain features in its encoding. In the task of question answering, the answer can be combined with the question to compose a hypothesis. If the hypothesis is entailed by a given text, that would mean that the answer is correct. Demszky et al. [10] used this to derive NLI datasets from question answering datasets. In text summarization, the original text could be considered to be the premise, and it would be necessary for it to entail the summarized text

for the latter to be considered valid, as shown by Mishra et al. [11]. In general, NLI can be considered to be somewhat related to most tasks that involve the understanding of human language.

The Textual Entailment task was first introduced by Dagan and Glickman [12] in 2004 as a way of recognizing language variability, i.e., sentences that present the same meaning despite being written in a different way. His approach relied on the use of predetermined sentence structures to assign a probability of entailment to each premise-hypothesis pair. In 2006, Hickl et al. [13] achieved state-of-the-art performance in the Second PASCAL Recognising Textual Entailment Challenge (RTE-2) [14] using a decision tree classifier, configuring one of the earliest successful attempts at tackling the task using machine learning. In 2014, Bowman et al. [15] achieved competitive results using a recursive neural network, which would then become the predominant architecture for NLI models over the next years.

In 2015, Bowman et al. published the Stanford Natural Language Inference corpus (SNLI) [16], a dataset for NLI constituted by 570K manually-labeled premise-hypothesis pairs of texts taken from image descriptions. The other existing datasets for Textual Entailment at the time were either tens of times smaller or used artificial methods for the construction or labeling of sentences, and would not present the same quality as manually labeled corpora. In this sense, the SNLI corpus enabled more effective use of deep learning techniques for NLI. The Long Short-Term Memory (LSTM) [17] architecture became popular in the field due to its ability to deal with variable-length input and their memory mechanism, which allows them to learn the semantic information of entire sentences.

Premise	**Label**	**Hypothesis**
Two children play outside in a field.	Entailment	Kids are playing outdoors.
A family between a van and fence.	Neutral	A family by a moving van.
A man riding a dirt bike.	Entailment	A human is riding a vehicle.
People are on a stage performing.	Contradiction	People are sleeping.
A black dog digs in the snow.	Contradiction	The dog sleeping in his bed.

Table 1.1: Examples of premise-hypothesis pairs from the SNLI corpus.

Some examples from the SNLI dataset can be found in Table 1.1. It can be noted that the relation is not purely logical: in the fourth pair, it is possible for a person to be sleeping and performing at the same time if the act of sleeping is part of the performance. Rather, the goal is to determine what a human would typically infer from the given information, which would be that the statements are contradicting each other. This property holds valid for the most versions of the task of textual entailment. Currently, there are multiple other datasets for NLI. The MultiNLI [18], for instance, contains texts from ten distinct genres, such as telephone records, letters and fiction. The SciTail [19] dataset uses texts

adapted from science questions, and the SherLIiC [20] focuses on lexical inferences, to name a few others.

Despite enormous progress in NLI over the last years, there are still many aspects of its state-of-the-art models that are not properly understood. One of them is whether such models can actually understand the semantics of the human language, or just rely on falsifiable heuristics. A well-known example of this is that models that completely ignore the premise are still able to perform well in some datasets, as shown by Gururangan et al. [21], who achieved a 67% accuracy on the SNLI using a hypothesis only model, while a classifier that assigns labels at random would be expected to achieve only 33%. McCoy, R. Thomas, Ellie Pavlick, and Tal Linzen [22] were able to identify some of the wrong heuristics used by state-of-the-art models, and showed that their accuracy would drop to much lower than chance when presented with examples that contradicted the heuristics. There is also a lack of literature on the effects of hyperparameter tuning on the performance of NLI models, which is one of the topics that this thesis will address.

This undergraduate thesis describes a comparative study of the performance of multiple LSTM-based models on selected NLI datasets, one in Portuguese and two in English languages. Moreover, the effects of hyperparameter tuning in the inference performance are explored on these models, by varying the learning rate and the size of the hidden layer of the LSTM. Finally, three different word embeddings, namely GloVe, Word2Vec, and FastText are explored and tested.

1.1 Goals

This research has two main goals. The first goal is to study the effect of the variation of hyperparameters, word embeddings, and model architecture on the performance of LSTM-based models for the NLI task. The second goal is to compare the best performing models and parameters across datasets, and determine if there is any correlation between them. To achieve these main goals, the following specific goals are established:

- Choose three NLI corpora for the comparative evaluation, in which one is in Portuguese language and two in English language.
- Choose how each factor will be varied for the comparative study, that is: which models will be used, which hyperparameter values will be tested, and which word embeddings will be chosen.
- Train models for each dataset, variating the factors chosen in the previous item.
- Compare the results of these models for each dataset and analyze how each factor impacted their performance.

- Compare the results of these models across datasets and determine whether there is a correlation for the best performing variations in each one.
- If there is a correlation, determine if it is lower for the Portuguese models than it is between the English models.

This research has two main contributions. The first main contribution is to the literature on parameter variation for NLI models, by determining how significant is the effect of these variations in the chosen datasets. The second main contribution is to help determine if there is a significant adaptation required for the best performing models in English datasets when using them for Portuguese datasets.

1.2 Structure

This undergraduate thesis is structured as follows: This first chapter introduces the NLI task, along with a brief history and open challenges in the field. It also introduces the main goals of this research and the steps that will be taken to achieve them. Chapter 2 presents the theoretical background required to understand this work, covering briefly topics such as LSTMs and attention mechanisms. Chapter 3 reviews the literature related to this thesis, and show how this research can integrate it. Chapter 4 describes the methodology used for training and testing models, along with analyzing their results. Chapter 5 shows the results of the experiments and complete the established goals. Finally, Chapter 6 concludes this thesis and suggests the next steps for the research.

Chapter 2

Theoretical background

This chapter covers the theoretical background required to understand this research. Section 2.1 formally describes the task of NLI. Section 2.2 presents basic principles of machine learning. Section 2.3 explains the basics of artificial neural networks, including concepts such as learning rate and hidden layers, which are the factors considered in the hyper-parameter tuning. Sections 2.4 and 2.5 cover concepts that were used in the proposed models. Section 2.6 approaches word embeddings and their use in the proposed models. Section 2.7 presents evaluation strategies for NLI, and finally, Section 2.8 concludes the chapter.

2.1 The task of Natural Language Inference

Let t_1 and t_2 be two non-empty groups of sentences in natural language. The NLI task is the problem of automatically classifying the relation between t_1 and t_2 with respect to whether one of them can be inferred from the other. It was created by Dagan and Glickman [12] and originally referred as Recognising Textual Entailment (RTE). Currently, both Textual Entailment and Natural Language Inference are used frequently in literature. The original version of this task has two possible relations:

- Entailment, if $t_1 \implies t_2$;
- Non-entailment, if $t_1 \not\Longrightarrow t_2$.

The entailment case states that t_1 entails t_2, meaning that t_1 infers t_2, or that t_1 implies t_2. A common variation of this task splits the non-entailment relation into two parts:

- Contradiction, if $t_1 \implies \neg t_2$;

- Neutral, otherwise.

Other non-entailment variations can be found. For example, other labels can be used to represent $t_1 \impliedby t_2$ and $\neg t_1 \impliedby t_2$. These variations are, however, much less common, and will not be used in this thesis. In this sense, t_1 is assumed to be premise and t_2 the hypothesis, as this is the nomenclature used in literature.

It is important to note that the goal of NLI is to guess the inference that would be made by most humans, not the most logical inference. For example, consider the premise "The person is swimming" and the hypothesis "The person is in the water". A purely logical analysis would conclude that the relation is of non-entailment, since it is possible for a person to be swimming in something other than water. However, if a person in a conversation says they were swimming, it is not typical to question what they were swimming in, and it is usually assumed to be water when in lack of greater context. Due to this, the relation is actually of entailment.

This definition also means that the relation is subjective to interpretation, which is something important to consider when annotating datasets for NLI. In the SNLI [16], for instance, a council of five annotators was used to create a gold label for some of the premise-hypothesis pairs in the dataset. It was found that the original labels, given by the writer of each pair, only agreed with the gold labels 85.8% of the time.

2.2 Principles of machine learning

Machine learning is the field of study of algorithms that are capable of learning and making predictions. If the list of possible predictions forms a finite and well-defined set of categories, the prediction is said to be a classification. On the other hand, if the possible predictions form a continuous range, the prediction is said to be a regression.

To make predictions, a machine learning algorithm first needs to learn the kind of data that it will be predicting. The data that is used to train it is called training data. There can also be a testing data, used to evaluate the performance of the model on unseen instances. In some cases, it is also necessary to test the model to decide something related to its training. Allowing the testing set to interfere with the training makes it no longer unseen, so it cannot be used here. Instead, a third dataset would be used, called development set or validation set. An example of it will be shown later.

2.3 Artificial Neural Networks

An artificial neural network [23] is essentially a composition of mathematical functions. It was proposed as a mathematical model for how biological neurons work, before becoming

a useful concept for machine learning. It is generally used as a system that can automatically learn the desired output for each valid input when given sufficient training data as examples. More details are described next.

2.3.1 Artificial neurons

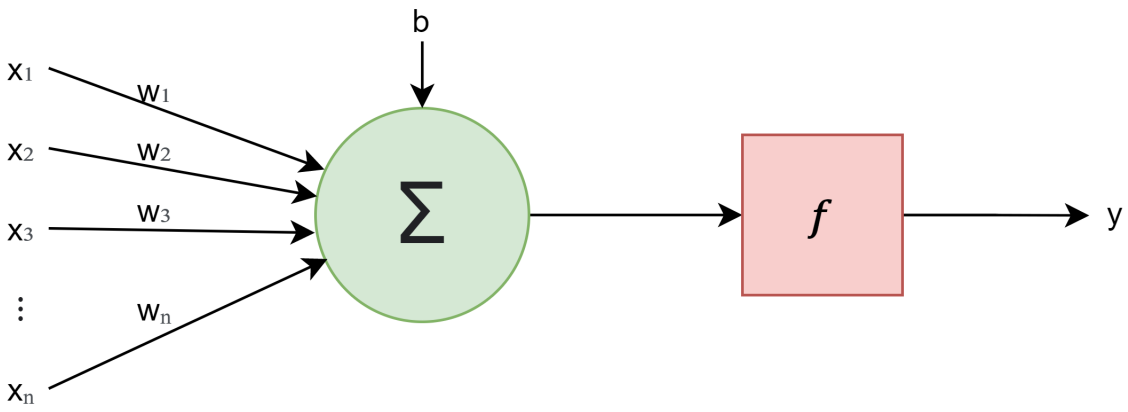

Figure 2.1: Visual abstraction of an artificial neuron. Based on the illustration in [1].

Artificial neurons are functions that receive a series of inputs and combine them into a single output. It was first introduced by McCulloch et al. [23] as a model for biological neurons, with the perceptron [24] being an important improvement. Each input received can either be the output of another neuron or part of the original input to the neural network. They were modeled after the idea that biological neurons can either activate or not when given an input, and in a similar way, are designed such that their output is a high value if they were activated, or a low value if they were not. Each input x_i is first multiplied by a weight w_i, which is considered the weight of the connection. A random initial value is usually set to this weight and it is modified during learning, which will be explained later. The results are then summed, producing a single value u.

$$u = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

An activation function f can now be chosen, and it will determine if the value u is enough to activate the neuron or not. In order to solve complex problems, this function needs to be non-linear; otherwise, the output of the neuron itself will be a linear combination of its inputs, which can make it unable to properly approximate the function required for some tasks. Some of the more commonly used functions are *sigmoid*, *tanh* and *ReLU* (Rectified Linear Unit). The *tanh* function limits the output to the range $(-1, 1)$, with a value close to 1 signaling that the neuron was activated, and a value close to -1 signaling

that the neuron was not activated. Similarly, the *sigmoid* function limits the output to the range $(0, 1)$, and the $ReLU$ function limits the output to the range $[0, +\infty)$, with a 0 signaling a non activated neuron. It is also helpful to add a bias term b to the value u before applying it to the activating function. The bias can be used to shift the activation function, allowing the neuron to change when it should activate. The value of the bias can be learned during training, along with the other weights. Combining these concepts leads to the formula of the output y of a neuron, depicted in equation 2.2.

$$y = f(b + \sum_{i=1}^{n} x_i w_i) \tag{2.2}$$

2.3.2 Neuron layers

More complex systems can be formed by connecting the output of an artificial neuron to the input of others. Some of the earliest neuron layers were shown in [25] and [26]. Figure 2.2 shows a neural network formed by connecting eight artificial neurons. It can be noted that each neuron still has only one output, but that output is repeated to other neurons. Each vertical stack of neurons can be thought of as a layer in the neural network. The first one is called the Input Layer, which receives the input of the neural network. The last one is called the Output Layer, and its output is the output of the neural network. All layers between the first and the last are called Hidden Layers. Additionally, a neural network without cycles is called feedforward neural network.

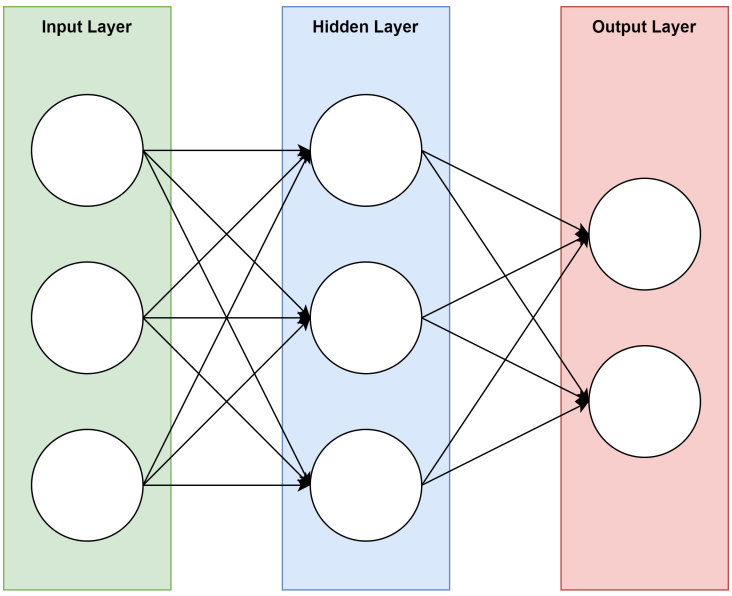

Figure 2.2: An example of artificial neural network with three layers and eight neurons. Based on the illustration in [2].

Increasing the number of neurons in a layer increases how much information can pass through it, but can also increase the time and amount of data needed to train the network. The number of layers and number of neurons in each layer need to be set before the training of the model starts. In the case of the input layer, the number of neurons needs to be equal to the number of attributes in the training data, as each neuron of this layer will receive one such feature as input. For the output layer, it is common to have one neuron for each possible label. At the end of the processing, the output neuron with highest output would correspond to the predicted label. The parameters of the network that need to be set beforehand and cannot be learned during training are also called hyperparameters. In this work, the number of neurons in the hidden dimension will be one of the variables considered in the hyperparameter tuning.

2.3.3 Loss function

Neural Networks learn by the process of adjusting their weights and biases to better solve the problem [27]. In supervised learning, the training data is composed of data samples that will be given as input to the network, and well as their expected output [28]. The initial weights and biases of a neural network are often either randomized or set to zero, and are refined during training to improve the performance of the network. In some cases, if the dataset used for training is too small, then a model might be first trained in a large dataset with a related task, and then have its parameters fine-tuned in the smaller data. This process is called Transfer Learning [29].

Let y be the vector of expected outputs of n data samples and $\hat{y}$ the vector of outputs of the neural network for the same data samples. The loss value can be computed by comparing y and $\hat{y}$, which indicates the difference between the output of the network and the expected output for a given input. There are many ways to calculate the loss value. A simple approach is using the $L1$ and $L2$ loss [30]. The $L1$ loss uses the Mean Absolute Error function, which is the sum of the absolute differences between the expected output and the network output (Equation 2.3). The $L2$ loss uses the Mean Squared Error function, which is similar, but uses the squared differences instead of absolute ones (Equation 2.4). Since this value is squared, large differences will greatly increase the loss, while small differences will have little impact.

$$Mean\ Absolute\ Error = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.3}$$

$$Mean\ Squared\ Error = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.4}$$

For any fixed training data, the loss value is the same for the same weights and biases. The loss of a neural network can thus be thought of as a function of its internal parameters. The problem of minimizing the loss is then the same problem as finding the global minimum of this function, which are difficult and time consuming processes. Alternatively, optimization algorithms for neural networks such as Stochastic Gradient Descent (SGD) [31] or Adam [32] attempt to determine a local minimum. Different optimizer functions can lead to different minima, but most of them work by taking steps through the parameters space of the function in a direction where a local minimum might be found. The length of this step is determined both by the loss and the learning rate, which is a hyperparameter. Higher values for the learning rate might lead to larger steps, thus making it harder to find a local minimum. On the other hand, lower values for the learning rate might result in smaller steps, in such a way that the optimizer might get stuck in a bad local minimum. Choosing a good learning rate is, then, very important to the model's performance.

This concept of updating weights is implemented through the backpropagation algorithm [33]. It works by calculating the gradient of the loss with respect to the weights, and updating each weight accordingly. The name backpropagation comes from the fact that it first calculates the weight changes of the final layer, and then propagates backwards to the first layer of the network, updating the remaining weights on its way. Formally, for feedforward neural networks, the gradient at layer i for the loss function C, referred as $\nabla_{W_i} C$, can be defined as shown in the Equation 2.5. W_i is the set of weights at layer i, a_{i-1} is the output of layer $i-1$, and δ_i is defined as shown in Equation 2.6, where f_{i-1} is the activation function used at the layer $i-1$ and $\circ$ is the element-wise product.

$$\nabla_{W_i} C = \delta_i (a_{i-1})^T \tag{2.5}$$

$$\delta_{i-1} = (f_{i-1})' \circ (W_i)^T \cdot \delta_i \tag{2.6}$$

Note that the gradient at layer i is multiplied by the gradient at layer $i+1$. If the neural network has a large number of layers, and the gradients frequently have values higher than 1, the gradient at the first layer will be extremely high, a problem known as exploding gradient. Conversely, if the values of the gradients are frequently between 0 and 1, the gradient at the first layer will tend to zero, a problem known as vanishing gradient [34].

2.3.4 Epochs and Early Stopping

During training, a neural network may not be able to keep the entire data in memory, and will take as input only a fraction of the complete dataset at a time, called a batch. The accumulated loss for the entire batch is then summed and used to update the weights of the network. This process is repeated until the network has passed through the entire training dataset once, which defines an epoch. A rudimentary approach to training machine learning models is to set a fixed number of epochs for the model to be trained. However, this can lead to the following outcomes: the model might spend much more time than needed training; the number of epochs might be too little, causing the model to perform poorly; and the number of epochs might be too much, causing the model to overfit regarding the training data and to lose generalization abilities in unknown data to the model. To avoid this problem, early stopping techniques [35] can be used to stop the training at just the right time.

Ideally, the training loop should be stopped at the point where the model can no longer learn. It is common to use the loss to determine this: if the loss is still decreasing, the training continues; otherwise, the training stops. However, using the loss of the training data is not appropriate as it can continue to decrease while the model overfits, leading it to perform well on the training set, but poorly on new data. The loss of the testing data should also be avoided since using the test set in the training of the model would make it no longer representative of unseen data. What can be done, instead, is to split the training set into a development set and a new, smaller, training set. The loss on the development set can then be used to determine the point of early stop. It may also be needed to use a dropout layer, which randomly zeroes elements of the output of some layers, so as to avoid that the neural network learns too much about the training data.

2.4 Recurrent Neural Networks

A Recurrent Neural Network (RNN) [3] [36] is a specific type of artificial neural network that is capable of dealing with sequences: ordered lists of elements, often with each element having a relation to others. It can also preserve information from previous elements, which allows it to give different outputs to an input depending on earlier timesteps, allowing the underlying model to take context into consideration. This is in contrast to traditional neural networks, which always consider the input to be independent. RNNs are thus particularly useful for dealing with sequential data such as time series or text.

Figure 2.3 shows a simple recurrent network model. Formally, for Elman's recurrent network, its output at timestep t, y_t will be given by Equation 2.7. Additionally, it will have a hidden layer vector h_t that can be used in further steps. In the equations, W, U

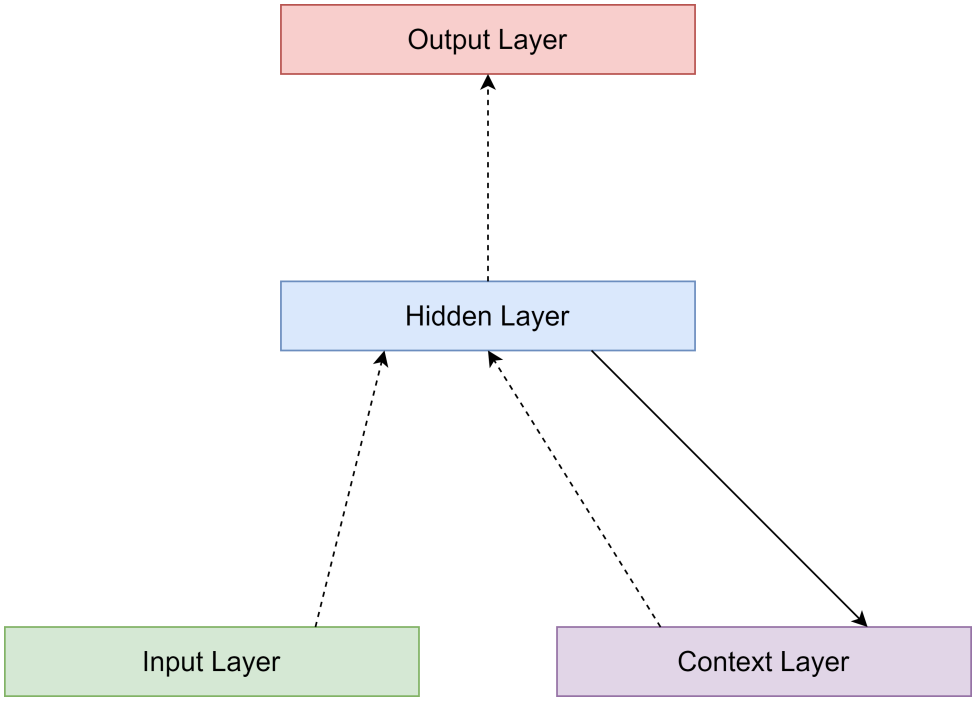

Figure 2.3: Elman's recurrent network, as depicted in the original work [3]. Dotted connections between layers represent learnable weights. The connection from the hidden layer to the context layer has a fixed weight of 1.

and b are learnable parameters, x_t is the input at timestep t, y_t is the output, and f is an activation function.

$$y_t = f_y(W_y h_t + b_y)$$

$$h_t = f_h(W_h x_t + U_h h_{t-1} + b_h) \tag{2.7}$$

By connecting the hidden states of consecutive timesteps, sequences of arbitrary lengths can be processed one item at a time, as shown in Figure 2.4. Each blue block in each of the sequences corresponds to the same cell at different timeteps, and thus have the same weights and biases. They are aligned in sequence for the better visualization of how the processing of sequences work.

Note that, through time, RNNs behave similarly to feedforward neural networks, with each RNN block applying a composition of functions to its input and passing it to the next block. Their weights can also be updated using the backpropagation algorithm, with the gradient being calculated for each timestep. However, if the sequence being processed is large enough, RNNs can become especially susceptible to the exploding and vanishing gradient problems.

The Long Short-Term Memory (LSTM) [17] architecture is a type of RNN designed to

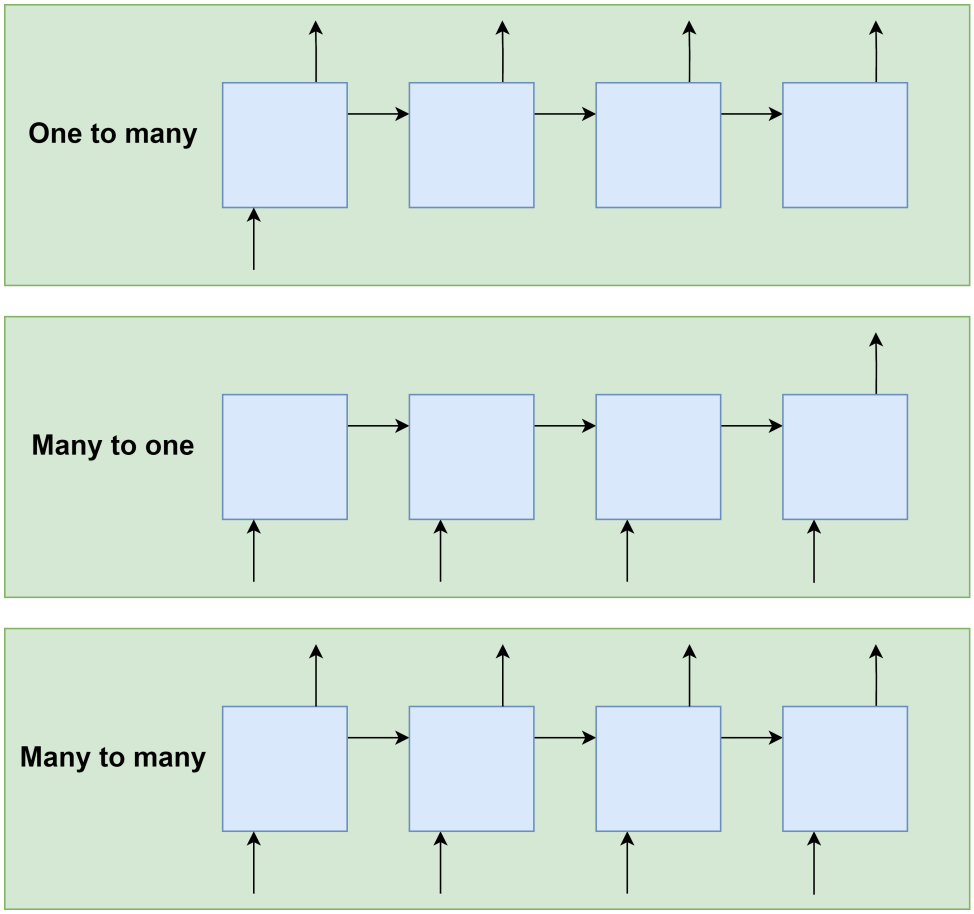

Figure 2.4: Illustration of how RNNs can process sequences. Adapted from [4].

deal with these gradient problems. LSTMs do this through a forget gate, which indicates how much the cell should forget about previous context. Figure 2.5 illustrates an LSTM cell and its constituent units. Note that it also has a c_{t-1} as input and c_t as output, indicating the cell state at a timestep t. The cell state is responsible for storing information over long periods of time, while the hidden state just encodes the information of the most recent timesteps. In this case, $y_t = h_t$.

The LSTM gates and their states can be mathematically described according to Equations (2.8):

$$f_t = sigmoid(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = sigmoid(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = sigmoid(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

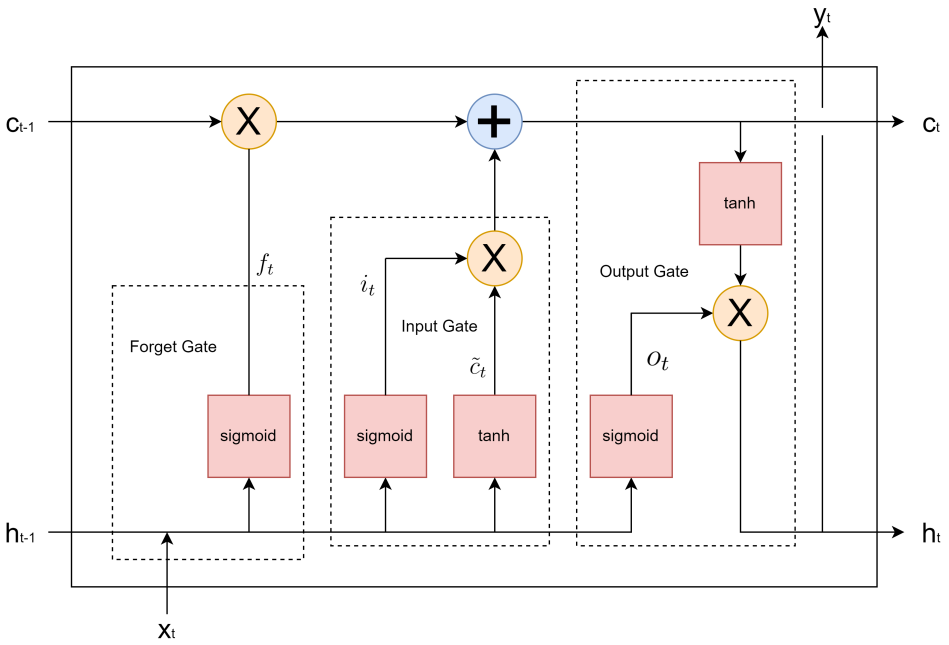

Figure 2.5: Common representation of the LSTM architecture. Each sigmoid activation function represents one of the gates of the LSTM, from left to right: Forget Gate, Input Gate, and Output Gate. Lines joining without an operation indicates concatenation. Based on the illustration by [5].

$$h_t = o_t \circ tanh(c_t) \tag{2.8}$$

in which $\circ$ refers the element-wise product of matrices, $c_0 = 0$ and $h_0 = 0$ are the initial values, and for each unit k ($k \in \{o, f, i, c\}$), W_k and U_k are weight matrices, and b_k is a bias term. t is the index of the current timestep, i_t is called the input gate's activation vector, and o_t is called the output gate's activation vector. Finally, $\tilde{c}_t$ is the cell input activation vector, and f_t is the forget gate activation vector.

2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [37] [38] are artificial neuron networks that use the convolution operation in some of their layers to extract features from segments of the data. They are commonly used in computational vision due to them being able to automatically learn the most important attributes of images. In contrast, other architectures rely on handpicked features, which may not be good enough to properly represent the image. Convolutional layers work by applying filters to their input. Figure 2.6 shows a filter with arbitrarily chosen weights W (Equation 2.9) being applied to subsets of an input matrix I (Equation 2.10). These weights are usually learned during training time, just like in other neural network architectures.

$$W = \begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix} \tag{2.9}$$

$$I = \begin{bmatrix} 0.88 & 0.09 & 0.63 \\ 0.53 & 0.28 & 0.11 \\ 0.03 & 0.67 & 0.29 \end{bmatrix} \tag{2.10}$$

When using CNNs for images, the input matrix can be the pixels of the image, with the value of the pixel being its color. For colored images, a conventional strategy is to split the image into three matrices, one for each of the main colors in the RGB color model, and applying a filter, which can present three dimensions. Multiple convolutional layers can also be applied in sequence to generate more complex features, sometimes using techniques such as pooling to reduce the number of layers required to achieve satisfying results.

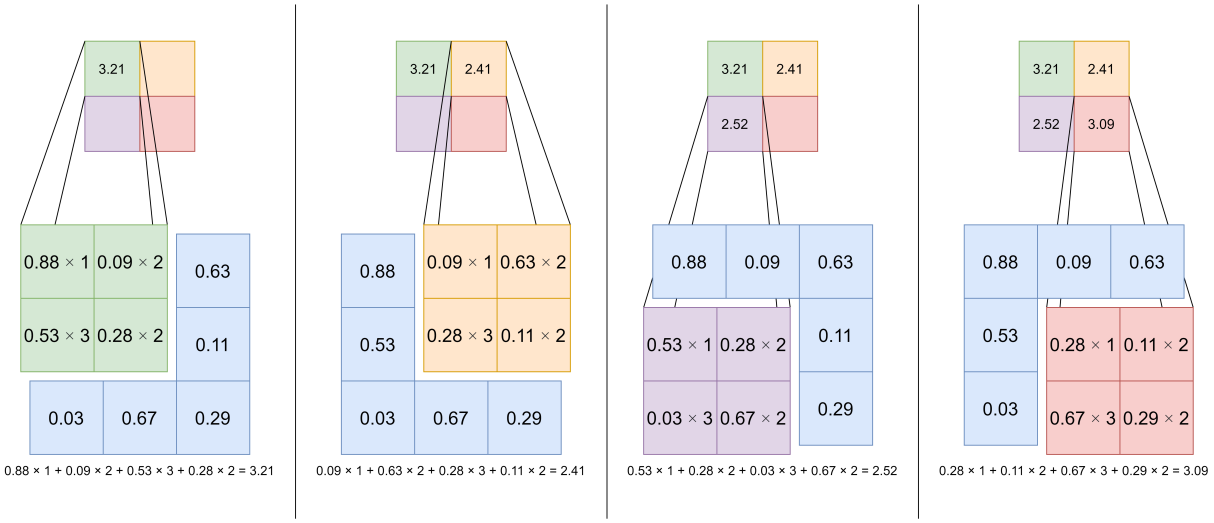

Figure 2.6: A 2×2 filter being applied to a 3×3 matrix.

The application of a filter produces a feature map, with each element of the output corresponding to a feature of the original input. The convolution shown in Figure 2.6 is two-dimensional since that operation is performed on two-dimensional images. In cases of one-dimensional inputs, such as texts and signals, the one-dimensional convolution can be employed as shown in Equation 2.11:

$$[s * \omega](t) = \sum_{i=-d}^{d} s(t-i)\omega(i), \tag{2.11}$$

in which ω is a zero-centered filter presenting length $2d+1$ and s is the input signal. The 1D Convolution is commonly used for texts and other data types that are one-dimensional by nature.

2.6 Word Embeddings

Words must be somehow represented as numbers to be used as input to neural networks. A possible approach is to map each word to an unique id and use an array to represent the word: if the value of the array is 1 in position i and 0 in other position, then the word represented by that array is the word of id i. This representation is known as one-hot encoding. Although it works well for some NLP problems, it requires the model to learn the meaning of the word, which can be troublesome or even impossible. A better approach is to use word embeddings, which are sets of words represented in a multidimensional space. Word embeddings can be pre-trained on large corpora to generate a semantically meaningful representation for words. These pre-trained models can then be used in neural networks, greatly simplifying architectures that deal with texts.

One example of word embedding generation algorithm is word2vec [6], which uses a neural network to learn a representation for each word according to the context in which it appears in the training corpora. In this embedding, each word is represented by a vector, and similar words will have close vectors. There are two architectures for the neural network: CBOW and Skip-gram. In the CBOW architecture, the model tries to predict a word based on the surrounding words. In the Skip-gram architecture, the model tries to use a word to predict surrounding words.

Another word embedding model, Global Vectors (GloVe) [39], puts two words together when, according to the training data, they are likely to appear together in the text. The fastText algorithm [40] takes a less traditional approach, learning how to represent the context of characters rather than words. This architecture makes it easier for models to understand affixes and the root of words. The fastText algorithm also learns its embedding through a neural network, and also uses CBOW and Skip-gram.

2.7 Evaluation strategies

In machine learning, evaluation strategies can vary according to the task and dataset used.. Common metrics used in most classification tasks are precision, recall, and f1-score. Let y_i be the predicted label for the instance of index i, and $\hat{y}_i$ the true label for this same instance. For a given label A in a classification problem, each of the predictions can be assigned one of four categories:

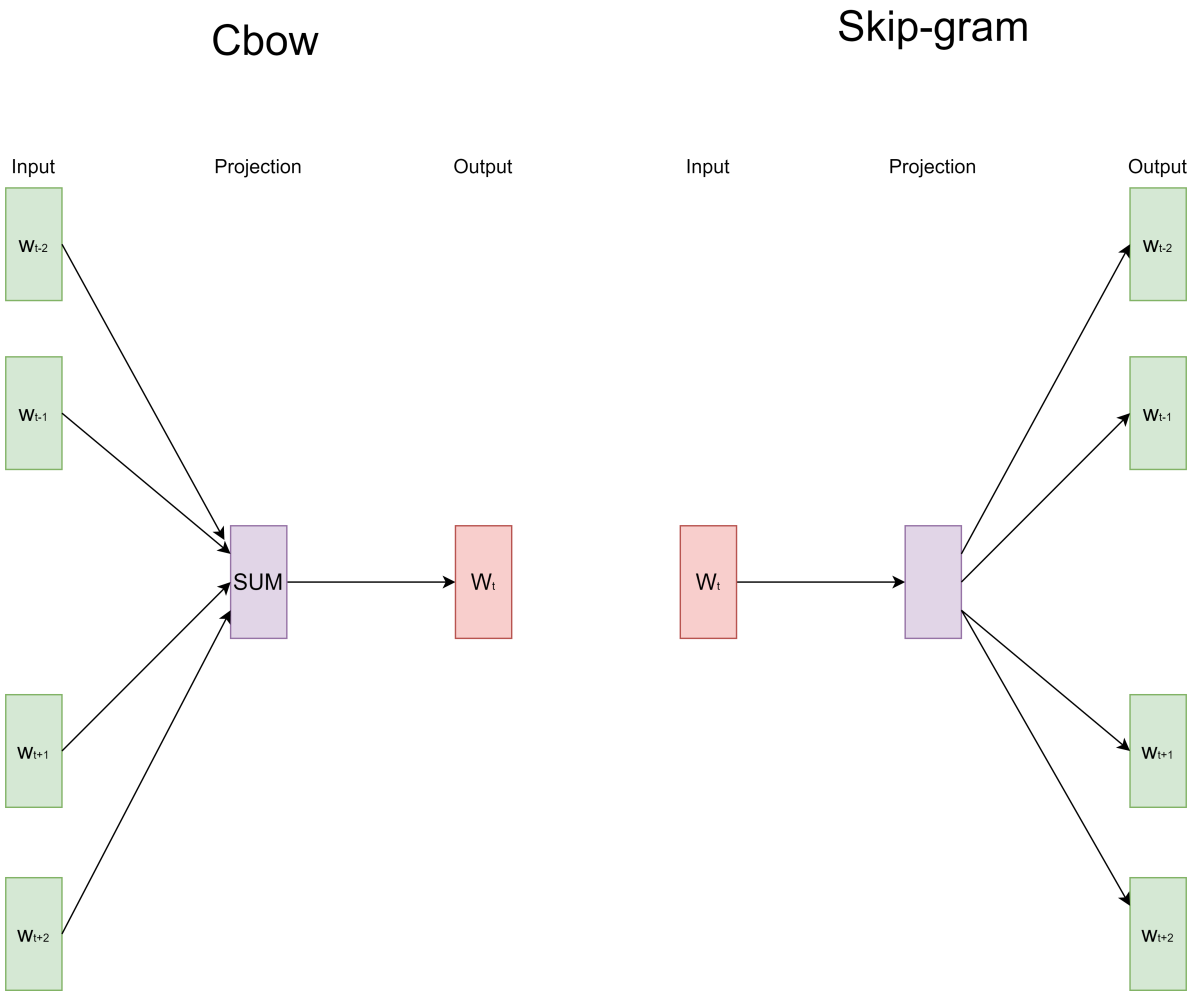

Figure 2.7: CBOW and Skip-gram algorihms. Based on the illustration by [6].

- True Positive (TP), if $y_i = \hat{y}_i = A$;
- True Negative (TN), if $y_i = \hat{y}_i \neq A$;
- False Positive (FP), if $y_i = A$, but $\hat{y}_i \neq A$;
- False Negative (FN), if $y_i \neq A$, but $\hat{y}_i = A$.

The precision is defined as shown in Equation (2.12), and it is higher the fewer false positives are there. The recall is defined as shown in Equation 2.13, and it is higher the fewer false negatives. The f1-score, which is the harmonic mean of recall and precision (Equation 2.14), can be used as a balanced combination of the two metrics. These three metrics are used in many types of problems, including NLI. For this task, however, a more common metric is the accuracy score, which is the number of instances correctly classified divided by the total number of instances (Equation 2.15). However, it is important to note that the accuracy score is usually used if the dataset is balanced in terms of its labels.

$$precision = \frac{TP}{TP + TF} \tag{2.12}$$

$$recall = \frac{TP}{TP + FN} \tag{2.13}$$

$$f1 = \frac{2 \cdot precision \cdot recall}{precision + recall} \tag{2.14}$$

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.15}$$

2.8 Final considerations

This chapter presented the concepts required to understand this thesis. Moreover, some of the common nomenclature in literature that will be used in the rest of this work were presented. These concepts will now be applied over the next sessions.

Chapter 3

Related works

This chapter covers some of the literature related to this work. Section 3.1 presents the original articles of some of the most notable datasets in NLI, including those used in the experiments of this thesis. Section 3.2 covers two papers that also performed hyperparameter tuning for NLI tasks. Finally, Section 3.3 describes the contributions of this work to the literature.

3.1 Datasets

Soon after the introduction of the NLI task, the PASCAL Recognising Textual Entailment Challenge (RTE-1) was published, containing just over 1000 premise-hypothesis pairs. It became the most predominant NLI dataset at the time, and it had six new editions over the following years, with the last one being the RTE-7 published in 2011. In 2014, the Sentences Involving Compositional Knowledge (SICK) [41] corpus was published with the aim of being a dataset for the evaluation of Compositional Distributional Semantic Models (CDSMs).

CDSMs are models that attempt to understand the semantics of sentences using distributional semantics, which makes their task very similar to NLI. The SICK dataset was also built with 10000 premise-hypothesis pairs, and the main difference from it to other NLI datasets is that it focused on presenting a variety of lexical, syntactical, and semantical phenomena that were supposed to be used by CDSMs. These factors contributed to the SICK dataset becoming popular in the NLI field over the following years of its publishing.

The SICK dataset was constructed using descriptions of images and videos from the 8K ImageFlickr dataset and the SemEval 2012 STS MSR-Video Description data set. The sentences went through hand-built processing techniques in order to artificially create the linguistic phenomena that would be helpful for CDSMs. These techniques also facilitated

the process of artificially generating hypothesis for each premise. The pairs were then labeled using the Amazon Mechanical Turk [1] (MTurk) marketplace.

In 2018, Real et al. published the SICK-BR [42], a version of the SICK dataset translated to Portuguese. The SICK-BR was later used in the ASSIN 2 challenge [43], published in 2019, though it was changed to only have "entailment" and "not entailment" labels, as opposed to the three possible labels in the original paper. The ASSIN 2 is a task within the fields of both NLI and Semantic Textual Similarity and it has the submission record as a public ranking.

The first large-scale manually labeled dataset was the Stanford Natural Language Inference corpus (SNLI), published by Bowman et al. [16]. In their work, the authors used the Mturk website to create premise-hypothesis pairs. The premises were taken from the Flickr30k corpus, a dataset with image captions, and the participating workers from MTurk had to create three hypothesis for each premise: one that was entailed by the premise, one that was neutral, and one that was a contradiction, resulting in 570,152 sentence-hypothesis pairs. The workers were only given the captions, not the image itself, so that they would be able to extract no further information from the image that was not present on the caption. As was later observed in other research [21], this method for creating sentences introduced biases into the sentences, allowing models to achieve considerable performance using just the hypothesis.

In addition, Bowman et al. used a set of 30 highly trusted workers to conduct a validation of the dataset. This was performed by using councils of 5 workers to give an extra label, named gold label, to nearly 10% of the complete corpus. The gold label was determined by a majority vote among the 5 workers, with each pair only getting a label if at least 3 of 5 workers agreed. Notably, the council only had a unanimous vote in around 58% of the cases, and the gold label was different from the label of the author of the pair in 6.8% of the cases. These statistics show that NLI is highly susceptible to interpretation, and since the objective of the task is to infer the relation of a pair the same way as a typical human would, there may not be an objectively correct answer in some cases.

The authors also evaluated multiple models on their dataset. Models from the Excitement Open Platform [44], an architecture with state-of-the-art models for Textual Entailment, achieved a maximum accuracy of 75%. However, these models were evaluated using only two labels: entailment and non-entailment. A classifier using lexical features, such as unigrams and the number of words overlapping in the premise and hypothesis, achieved a maximum accuracy of 78.2%. Finally, three deep learning models were tested: one that simply concatenated the premise and hypothesis, one using RNNs to process both premise and hypothesis, and one using LSTMs. Out of these three models,

[1] https://www.mturk.com/

the LSTM performed better, achieving an accuracy of 77.6%. The authors also achieved near state-of-the-art performance on the SICK dataset doing Transfer Learning from the SNLI.

3.2 Hyperparameter Tuning

NLI datasets are usually either too large, to the point that doing hyperparameter tuning would be difficult, to too small, to the point where Transfer Learning is required to achieve a good performance. Due to this, papers that extensively cover hyperparameter tuning of models in the field of NLI are rare. Cases et al. [45], in their work, compared the performance of different word embeddings and their effect on hyperparameters. They compared random word embeddings, pre-trained GloVe, and word2vec, and retrofitted word embeddings, which were pre-trained embeddings with added WordNet information. They also varied the learning rate and initial weights of the network. For the latter, they varied both their strategy for initialization, between Gaussian initialization and orthogonal random initialization, as well as a hyperparameter they called initialization constant.

Their model used an encoder-decoder architecture with attention, and LSTMs for both encoding and decoding. The model was tested on the SNLI and on the classification of lexical relations in WordNet. For the classification of lexical relations in WordNet task, the retrofitted GloVe performed best, with an accuracy of 95.68%. The authors also hypothesized that the retrofitted embeddings did not perform as well as the traditional embeddings in the SNLI due to sentence complexity, and created a small dataset to test this theory. The first tests would have simple word pairs, and subsequent tests would have the word "not" appended to the beginning of each sentence. It was verified that the performance of the retrofitted embeddings fell quicker than the performance of traditional embeddings, suggesting that the latter is better in tasks with complex semantics.

Pang et al. [46] also used hyperparameter tuning in their work, and four common NLI datasets: SNLI, SciTail, MNLI matched, and MNLI mismatched. They also experimented with four models: DA (Decomposable Attention), ESIM (Enhanced Sequential Inference Model), BERT (Bidirectional Encoder Representations from Transformers), and MT-DNN (Multi-Task Deep Neural Networks). For the DA model, their tuning included learning rate, 3 hidden dimensions, and 3 dropout values, for a total of 7 hyperparameters. For the ESIM model, only the learning rate and two dropout rates were tuned. However, for the BERT and MT-DNN models, the same hyperparameters as the original work were kept.

The tuning for the DA and the ESIM models was also only performed on one dataset, and kept for the other 3 others. They also used a pre-trained parser to encode word

representations for these models. Overall, the MT-DNN had the best results in three of the four datasets, with the only exception being the MNLI matched, in which BERT had better performance. However, both MT-DNN and BERT had similar accuracy values, with both being significantly ahead of the DA and ESIM models.

3.3 Contributions of this thesis

The large size of NLI datasets, along with the increasing complexity of models, have both contributed to the avoidance of hyperparameter tuning in the field. Although some authors have included this optimization as part of their work, most new models do not make clear the employe strategy for choosing the hyperparameters, and some of them just reuse hyperparameters determined for other datasets. This work expands the literature on the effects of hyperparameter tuning on NLI models, and studies whether reusing parameters from other datasets can really be a viable strategy. This comparison of parameters on several datasets will also be explored in two different languages.

Chapter 4

Methodology

This chapter covers the methodology used in the experiments of this work. Section 4.1 presents the selected datasets. Section 4.2 describes the preprocessing used in the experiments. Section 4.3 presents the embeddings used, and Section 4.4 the proposed models based on LSTM. Figure 4.1 shows a diagram of the methodology of this thesis, with each of the first four blocks corresponding to a section of this chapter. They represent, in order, the steps taken to generate the models and to analyse the results. Although the last two blocks will only be fully covered in the next chapter, the method of analysis used in this research is described in Section 4.5. Finally, Section 4.6 concludes the chapter.

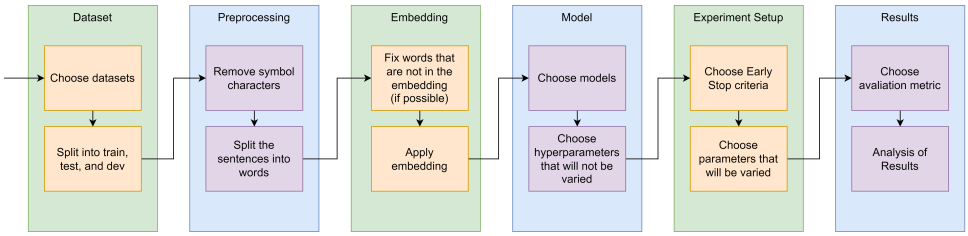

Figure 4.1: Diagram of the methodology used in this work.

4.1 Dataset

The first step comprises the selection of three datasets that will be employed in the experiments: one in the Portuguese language and two in the English language. It is important to choose three datasets that are similar in their text complexity and topic, as their results need to be compared fairly. Additionally, the datasets cannot be too large, as that will make the hyperparameter tuning computationally expensive, and they cannot be too small either, as that can make it hard to achieve decent results without Transfer Learning.

The dataset from ASSIN (*"Avaliação de Similaridade Semântica e Inferência Textual"*) 2 was the first dataset to be chosen, since it follows all of these conditions. The SICK dataset was also chosen as the second dataset for the experiments since ASSIN 2 is just its adapted translation. The SICK (and the ASSIN 2) is a dataset build from image and video descriptions, and it could then be interesting to choose another dataset of the same topic. Unfortunately, there are not many options that would present all these requirements. Because of that, a reduced version of the SNLI was employed, which is also a dataset built on image descriptions. Although the original SNLI has around 570K pairs, the reduced version had only 40K. This third dataset is now called RSNLI (Reduced SNLI). To improve the quality of the underlying data, only pairs with a gold label were chosen to integrate this dataset.

Finally, it is necessary to split each dataset into three sections: train, for training; test, for testing; and dev, as a development set to test the performance of the model during training. The splits of both ASSIN 2 and RSNLI datasets are available by the respective authors, so they are kept as it is for the experiments. However, the SICK dataset had to be manually split, which was performed as follows: 20% for testing, 16% for development, and 64% for training. Tables 4.1, 4.2 and 4.3 show the label distribution in each dataset.

	ASSIN 2	
	Entailment	None
Train set	3250	3250
Dev set	250	250
Test set	250	250
Total	**7500**	

Table 4.1: Label distribution in the ASSIN 2 dataset.

	SICK		
	Entailment	Neutral	Contradiction
Train set	1809	3545	944
Dev set	459	920	196
Test set	553	1130	284
Total	**9840**		

Table 4.2: Label distribution in the SICK dataset.

4.2 Preprocessing

The preprocessing used in text classification can present variations, but should generally follow the same preprocessing used by the authors of the word embedding being in con-

	RSNLI		
	Entailment	Neutral	Contradiction
Train set	10197	9660	10143
Dev set	1703	1634	1663
Test set	1701	1651	1648
Total	**40000**		

Table 4.3: Label distribution in the RSNLI dataset.

sideration. In this work, this recommendation was followed for the Portuguese dataset, as the preprocessing steps were available for it. As there were not preprocessing recommendations for the English datasets, the Portuguese versions were just adapted.

The recommended preprocessing converted all letters to their lower case version, removed some symbols such as "..." or "-", removed parenthesis and quotes, and others. An additional step of removing every non-letter symbol before executing the recommended preprocessing was added. At the end of the preprocessing, the sentences were split into words using the NLTK library [47].

4.3 Embedding

Three embeddings were considered: GloVe, word2vec, and fastText. These embeddings were used due to their already established success in other tasks that deal with text. The Portuguese dataset employed the available pre-trained embeddings from the NILC repository [48]. For the English datasets, the pre-trained embeddings were used: the GloVe embedding was the one from the original GloVe paper [39]; the word2vec embedding from Google was used [49]; and finally, the fastText model from the original website was used [40].

For the GloVe and word2vec datasets, some words that were in the dataset but not in the embedding were encountered, for both the Portuguese and English cases. For the ASSIN 2 and SICK datasets, these words were manually substituted for semantically equal words or expressions that could be recognized by the embedding. In the RSNLI, there were too many words to fix manually. Because of that, the RSNLI was left without a full correction. The FastText did not have this problem in any of the datasets, as it deals with characters and not with words.

4.4 Model

The requirements for choosing models rely on the fact that they could not be too complex, as it would lead to large training times and that they are based on LSTM. Four models fit-

ting this requirement were considered, which were named as Simple LSTM, CNN-LSTM, Dual LSTM, and Concatenation LSTM. These models and their fixed hyperparameters will be described in this section. In the following figures, the premise is represented as an ordered set of words p of size n, and the hypothesis is represented as an ordered set of words h of size m.

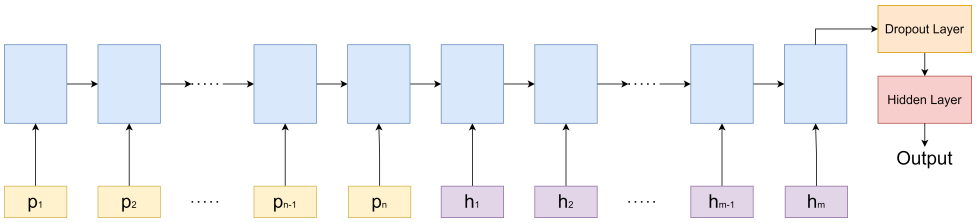

Figure 4.2: Simple LSTM model.

The Simple LSTM model performs the concatenation of the premise and hypothesis and runs an LSTM layer over it. The output of the last step of the LSTM is then sent to a dropout layer, which randomly converts some of the elements it received to zero. The probability of it converting any one element was set to 50% for all models. The output value of this dropout layer is then sent to a hidden layer, whose input size will be varied in the hyperparameter tuning. This hidden layer has an output size of size 2 for the ASSIN 2, or 3 for the SICK and RSNLI, which will determine the label of the premise-hypothesis pair. The reason for these sizes is that the proposed architecture used one output neuron for each class, and the ASSIN 2 dataset has two labels, while the others have three. This output layer will present the same size for all models. Additionally, all LSTMs had their initial hidden state and cell state defined to zero.

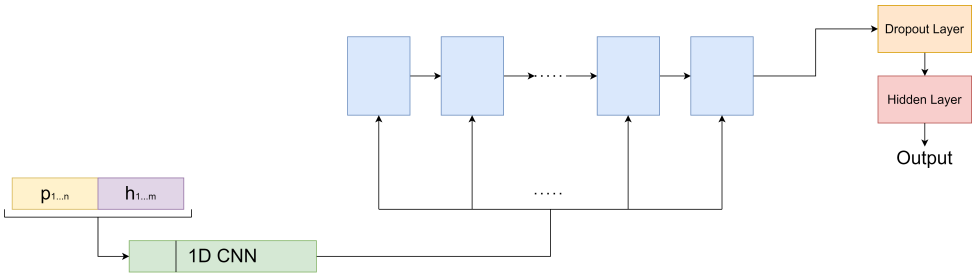

Figure 4.3: CNN-LSTM model.

The CNN-LSTM is similar to the Simple LSTM model, but the input first passes through a one-dimensional CNN. This CNN uses is composed of a single convolutional layer, and its output is split among the different timesteps of the LSTM cell. The output of the CNN has a size of 30 in its second dimension, with a kernel of size 10. These settings were arbitrarily chosen, as performing an extra hyperparameter tuning over them to find the best numbers would drastically increase the time required to complete all experiments.

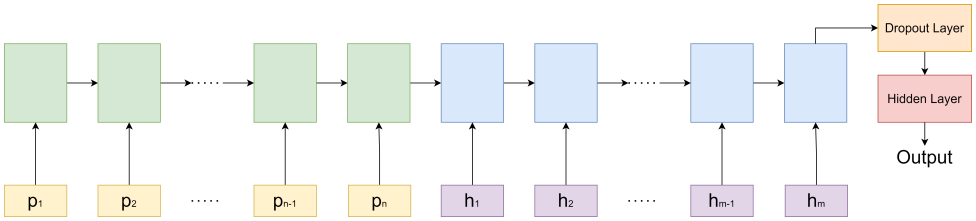

Figure 4.4: Dual LSTM model. Note that the green and the blue LSTMs are different.

The Dual LSTM model is also similar to the Simple LSTM, but it uses two LSTMs: one for the premise and one for the hypothesis. The final hidden state and cell state of the premise LSTM are used as the initial hidden state and cell state of the hypothesis LSTM. The advantage of this architecture is that it allows the model to learn different weights for the premise and hypothesis, which could help to identify the relation between these two sentences.

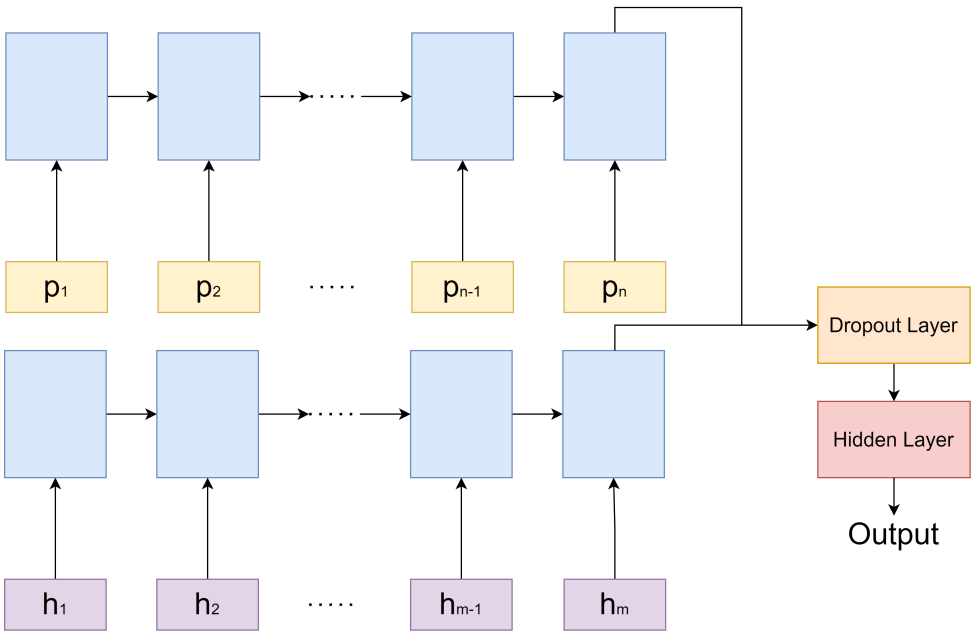

Figure 4.5: The Concatenation LSTM model.

Finally, the Concatenation LSTM process the premise and the hypothesis separately and then concatenates their outputs from the LSTM layer. The advantage of this architecture is that it can learn to represent the sentences in a semantically meaningful encoding, proper for detecting relations of entailment, contradiction, or neutrality.

4.5 Method of Analysis

The first step of the analysis will be to determine the most important factors among model architecture, embedding and hyperparameters on the performance of the models. This will be done by counting the frequency of each specific factor among the 15% best and 15% worst performing models, and highlighting the most common. The direct impact of hyperparameter tuning will also be considered by looking at the highest and lowest performance for each model architecture and embedding combination.

To determine if the best performing hyperparameters are similar among the datasets, the correlation between the performance of models will be analysed for each pair of datasets. There will be two types of analysis in this step. First, the model will be fixed and the correlation for the remaining factors will be considered. Then, the correlation of the average performance for each hyperparameter combination will be analysed. The analysis will conclude with a comparison of the best models of this research with other works from the literature on the same datasets.

4.6 Final considerations

This chapter presented the methodology used in this thesis, from the selection of datasets to the selection of models. The next chapter covers the experiment setup and the analysis of the results.

Chapter 5

Results

This chapter shows the results of the experiments. Section 5.1 describes the setup of the experiments. Section 5.2 covers the results for the ASSIN 2 dataset. Section 5.3 shows the results for the SICK dataset. Section 5.4 presents the results for the RSNLI dataset. Section 5.5 describes a cross-dataset analysis. Finally, Section 5.6 presents a comparison of the results of this research with other works, and Section 5.6 concludes the chapter.

5.1 Experiment Setup

The experiments were conducted using a Tesla V100S 32GB GPU and an Intel Xeon Gold 5220R CPU. To avoid overfitting, the loss of the model was checked at the end of each epoch using the development set. The chosen early stop criteria was that the training would stop after 500 epochs without improving the loss. After the training was finished, the models were reverted to their point of minimum loss on the development set, and then tested on the testing set. Due to the large number of experiments and time constraints, the traditional data split into training, development and test was employed instead of the K-Fold Cross Validation strategy.

The proposed models employed the negative log likelihood function as loss metric and the Adam optimizer. The hyperparameter tuning was done over the learning rate and the hidden layer size. For the learning rate, the values of $1 \cdot 10^{-6}$, $4 \cdot 10^{-6}$, $1.6 \cdot 10^{-5}$, $6.4 \cdot 10^{-5}$ and $2.56 \cdot 10^{-4}$ were tested, that is, starting at $1 \cdot 10^{-6}$ and increasing by a factor of 4 each time. For the hidden layer size, the values of 10, 20, 40, 80 and 160 were tested, that is, starting at 10 and increasing by a factor of 2 each time.

For the SICK and the SNLI datasets, the most commonly used evaluation metric is the accuracy score. For the ASSIN 2, both f1-score and accuracy are used. In order to keep all results comparable to each other, the accuracy score was the one considered in all

datasets. Using 3 datasets, 3 embeddings, 4 models, 5 learning rates and 5 hidden sizes, a final amount of 900 experiments (300 per dataset) were performed.

5.2 ASSIN 2

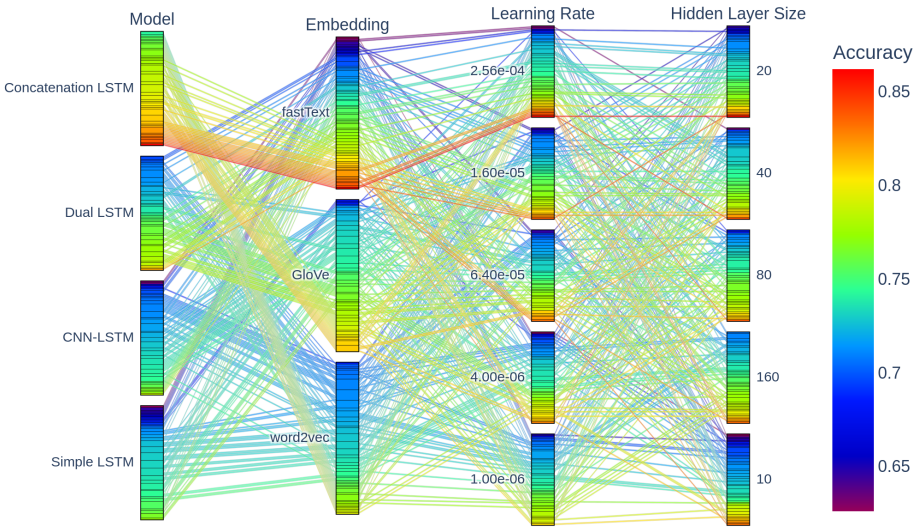

Figure 5.1: Complete results for the ASSIN 2 dataset.

Figure 5.1 shows the results for each experiment on the ASSIN 2 dataset. The color scale represents the accuracy of each model, ranging from the worst result, 0.626, to the best result, 0.862. There is a total of 300 lines, with each line going from the first column, Model, to the last column, Hidden Layer Size. Each line represents an unique combination of all four factors, and the color of this line indicates the accuracy obtained using that combination. Note, for instance, that the most red line starts at Concatenation LSTM, goes to fastText, then $2.56 \cdot 10^{-4}$, and then 20. This means that the best performing model used these parameters. The color range within each category in a column represents the accuracy distribution of models that used that category.

This dataset has two labels, entailment and non-entailment, and the accuracy of a randomized model would be 50%. It is easy to see that the Concatenation LSTM was the best performing model, followed by the Dual LSTM. For better visualization, Figure 5.2 shows the 15% best performing models and their accuracy values, and Figure 5.3 shows the 15% worst performing models. Each one of these two shows the result of 45 experiments.

Figure 5.2 shows that the best results were mostly dominated by the Concatenation LSTM model. In fact, it accounts for 42 of the 45 models, that is, around 93%. In the embeddings column, the fastText accounts for 25 (55.6%) of the results, and GloVe for 17 (37.8%). There is not a predominant learning rate or hidden layer size. The most common learning rate value in the image is 2.56e-04 and 4.00e-06, both with 12 (26.7%) experiments, and the most common hidden layer size was 10, with 11 (24.4%) of the 45 results, followed close by 160 with 10 (22.2%). The experiment with best result used a Concatenation LSTM model with fastText as embedding, learning rate of $2.56e-04$ and hidden layer size of 20, achieving an accuracy of 0.862.

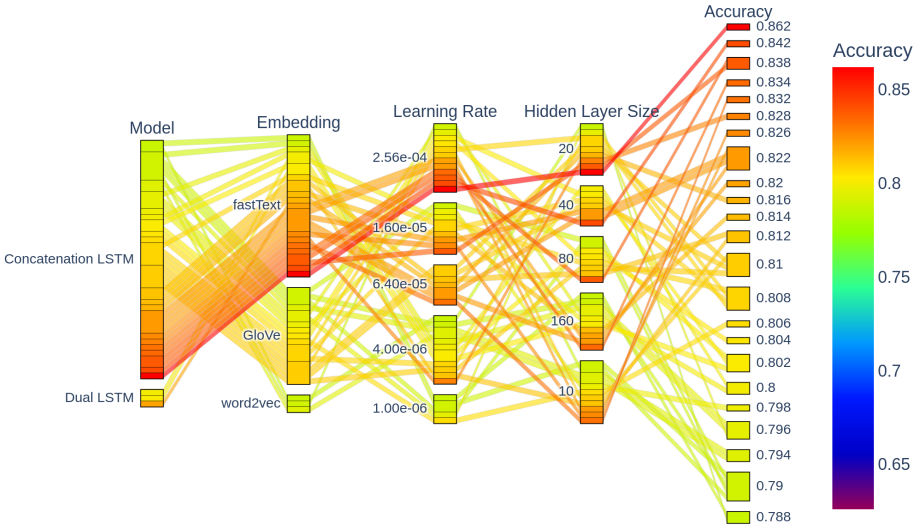

Figure 5.2: 15% best results on the ASSIN 2 dataset.

Figure 5.3 shows that the most predominant models in the worst results were CNN-LSTM, with 20 (44.4%) of the results, and Simple LSTM, with 16 (35.6%). It is not unexpected that the Simple LSTM did not perform well, as all the other models are improvements on it. It is a bit surprising, however, that the CNN-LSTM had more of the worst results.

Figure 5.3 shows still that the fastText embedding was the most common among the worse models, accounting for 22 (48.9%) of them. This is interesting, since fastText was also the most predominant model among the best results. It could be that the Concatenation LSTM was the only model to properly learn how to use this embedding, while other models could not use it properly. The most common learning rate was 4.00e-

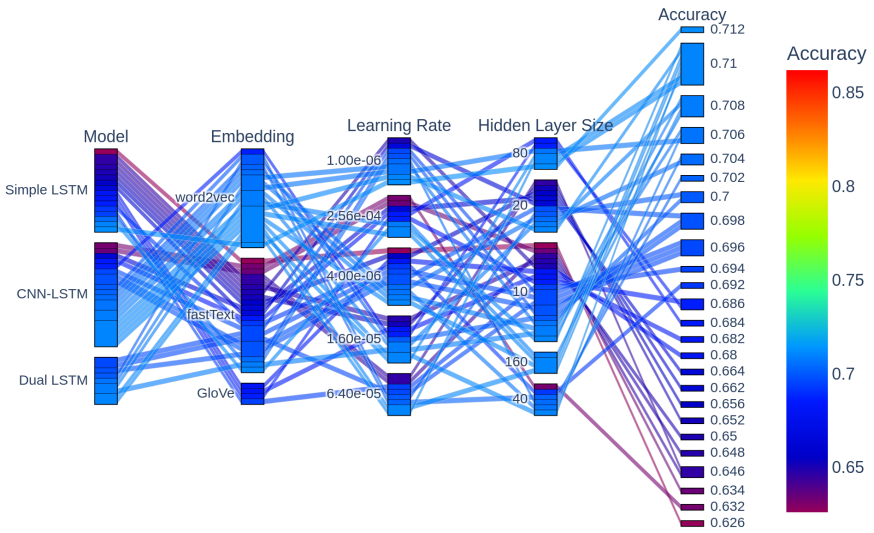

Figure 5.3: 15% worst results on the ASSIN 2 dataset.

06, with 11 (24.4%) of the results, and the most common hidden layer size was 10, with 19 (42.2%).

Table 5.1 shows the best and worst results on the ASSIN 2 dataset for each model-embedding combination, as well as the range. The range columns shows the difference between the best and worst results. It is possible to see that, in some cases, the hyperparameter tuning of the learning rate and hidden layer size accounted for differences of accuracy as high as 15.4%, while going as low as 2.8% in other model-embedding combinations. This maximum difference is 15.4% is around 65.2% of the maximum variation for all models in the dataset, which is 23.8%. The average range was 7.5%, while the median range was 5.8%. This indicates that the hyperparameter tuning had a significant effect in the results.

5.3 SICK

The experiments on the SICK dataset did not show a large variation regarding their accuracy values. Figure 5.4 shows the results of all proposed models. The color indicates the accuracy, ranging from 0.556 to 0.626 - a variation of just 7%. This dataset has three labels and the accuracy of a randomized model would be about 33.3%.

Figure 5.5 shows the 15% best results on the SICK dataset, that is, the 45 best out of 300 total. The most frequent model in these results was CNN-LSTM, with 21 (46.7%) of

Model	Embedding	Worst	Best	Range
Concatenation LSTM	fastText	0.776	0.862	0.086
Concatenation LSTM	GloVe	0.778	0.810	0.032
Concatenation LSTM	word2vec	0.742	0.794	0.052
Dual LSTM	fastText	0.696	0.820	0.124
Dual LSTM	GloVe	0.722	0.780	0.058
Dual LSTM	word2vec	0.700	0.740	0.040
CNN-LSTM	fastText	0.632	0.786	0.154
CNN-LSTM	GloVe	0.716	0.744	0.028
CNN-LSTM	word2vec	0.686	0.744	0.058
Simple LSTM	fastText	0.626	0.766	0.140
Simple LSTM	GloVe	0.680	0.772	0.092
Simple LSTM	word2vec	0.712	0.750	0.038

Table 5.1: Results for each model-embedding combination on the ASSIN 2 dataset.

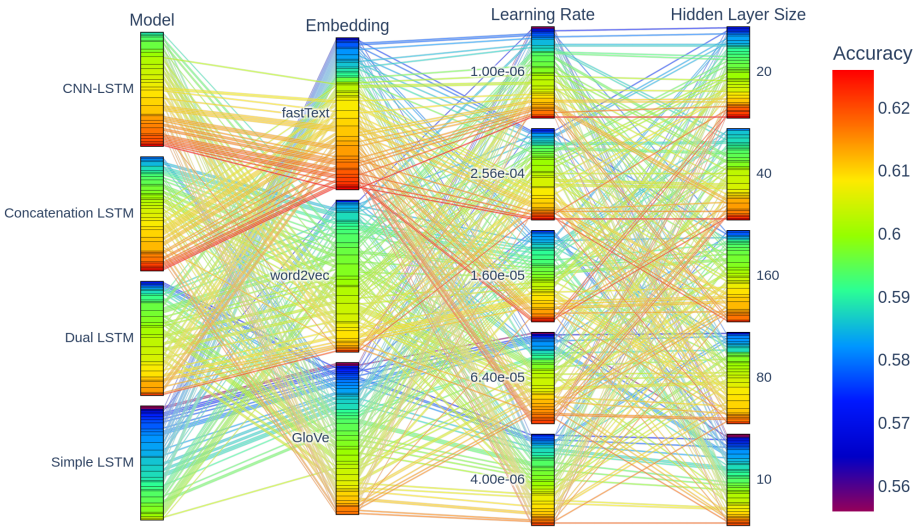

Figure 5.4: Complete results for the SICK dataset.

the 45, followed by Concatenation LSTM, with 15 (33.3%). The fastText embedding was by far the most common here, with 34 (75.6%) of the best results. The most common learning rate was 6.40e-05, with 14 (31.1%) results, and the most common hidden layer size was 40, also with 14 (31.1%) of the best results. It should be noted, however, that these 45 best results encompass an accuracy range of only 1.4%, from 61.2% to 62.6%. Consequently, these results may not be too reliable. The experiment with best result used the CNN-LSTM model, fastText embedding, learning rate of 1.00e-06 and hidden layer size of 20, achieving an accuracy of 62.6%.

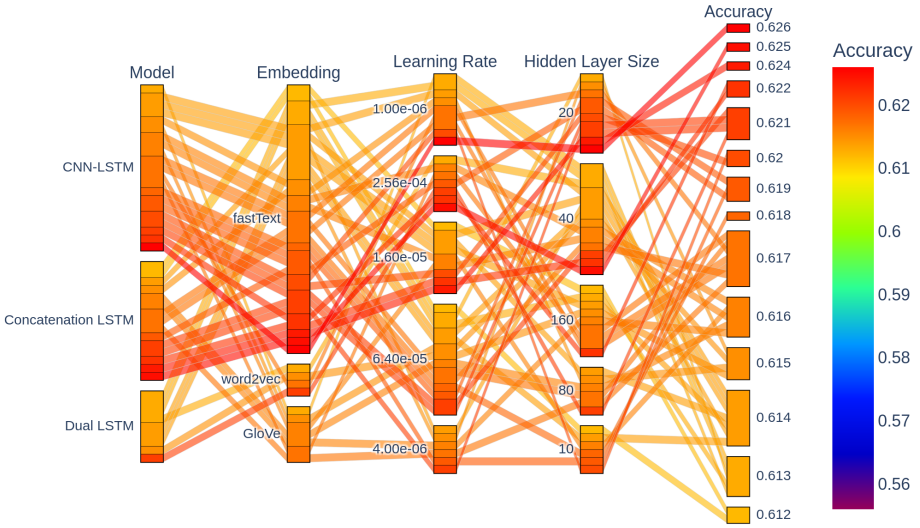

Figure 5.5: 15% best results on the SICK dataset.

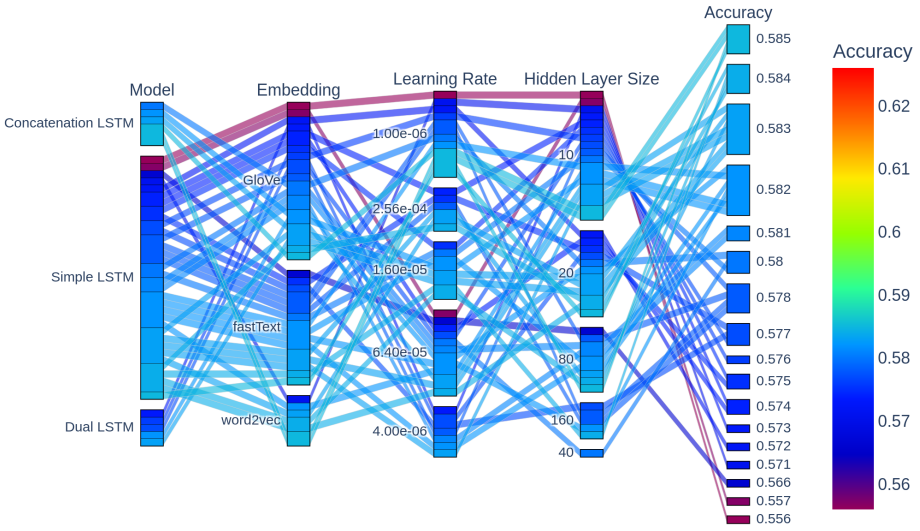

Figure 5.6: 15% worst results on the SICK dataset.

For the worst results, shown in Figure 5.6, the simple LSTM was the most common, with 34 (75.6%) of the 45. Once again, it is not surprising that the Simple LSTM performs worse than the other models. The GloVe embedding was the most common, with 22 (48.9%) experiments, followed by the fastText with 16 (35.6%). The most common

learning rates were 1.00e-06 and 6.40e-05, both with 12 (26.7%) of the worst results. Finally, the hidden layer size of 10 was the most common, with 18 (40%) results. Once again, it should be noted that the range in accuracy for the models in Figure 5.6 is of only 2.9%.

Table 5.2 shows the best and worst results for each model-embedding combination on the SICK dataset, as well as the difference between them. The largest range was 4.7%, and the lowest was 2.1%. This maximum difference is 4.7% is around 67.1% of the maximum variation for all models in the dataset, which is 7%. The average range was 3.0%, while the median range was 2.8%. This indicates that the hyperparameter tuning did not have a significant effect on the results of this dataset. However, since the variation between the best and the worst out of all results was only 7%, it could be possible that the impact of the hyperparameter tuning was limited by other factors, such as the simple architecture of the models.

Model	**Embedding**	**Worst**	**Best**	**Range**
CNN-LSTM	fastText	0.602	0.626	0.024
CNN-LSTM	word2vec	0.589	0.617	0.028
CNN-LSTM	GloVe	0.591	0.617	0.026
Concatenation LSTM	fastText	0.603	0.625	0.022
Concatenation LSTM	word2vec	0.585	0.612	0.027
Concatenation LSTM	GloVe	0.580	0.617	0.037
Dual LSTM	fastText	0.592	0.614	0.022
Dual LSTM	word2vec	0.588	0.621	0.033
Dual LSTM	GloVe	0.573	0.605	0.032
Simple LSTM	fastText	0.566	0.591	0.025
Simple LSTM	word2vec	0.571	0.604	0.033
Simple LSTM	GloVe	0.556	0.603	0.047

Table 5.2: Results for each model-embedding combination on the SICK dataset.

5.4 RSNLI

Figure 5.7 shows the complete results for the RSNLI dataset. The colors of the lines represent an accuracy range from 0.340 to 0.595. This dataset has three labels, and a randomized model would score about 33.3%, which is just slightly less than the worst scores of the proposed models.

Figure 5.8 shows the 45 best results on the RSNLI dataset. These experiments all had similar results, with a variance of 2.3%. It should be noted that the variance among all results of this dataset is of 25.5%, and it is interesting that so many models had close performance near the higher end. Out of the 45, 30 (66.7%) used the Concatenation LSTM

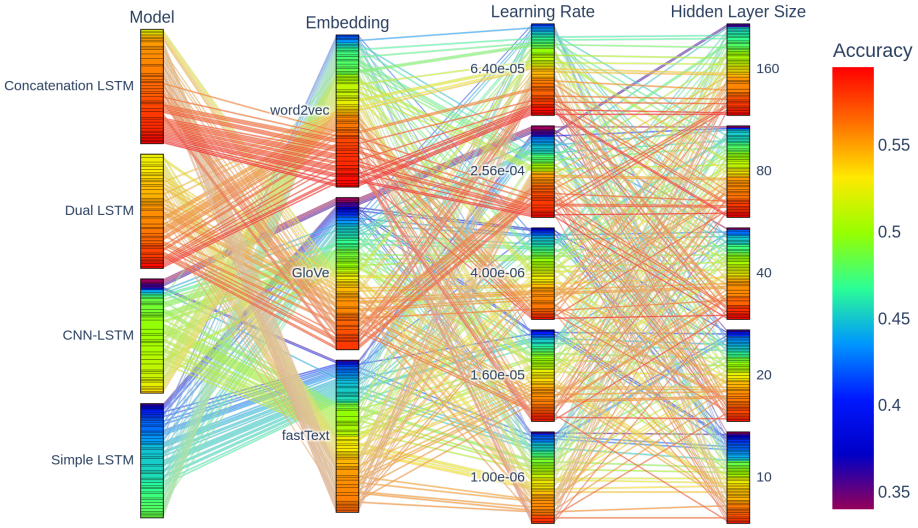

Figure 5.7: Complete results for the RSNLI dataset.

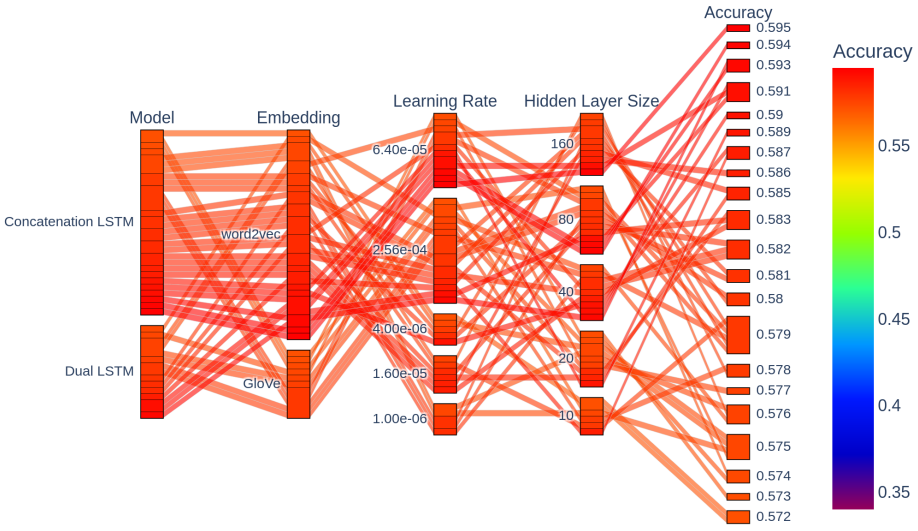

Figure 5.8: 15% best results on the RSNLI dataset.

model, while the other 15 (33.3%) had the Dual LSTM model. For the embeddings, 34 (75.6%) used word2vec. The most predominant learning rate 2.56e-04, with 17 (37.8%) of the best results, and the most predominant hidden layer size was 80, with 11 (24.4%) of the results. The experiment with the most result used the Concatenation LSTM model,

word2vec embedding, learning rate of 6.40e-05 and hidden layer size of 160, with a 59.5% accuracy.

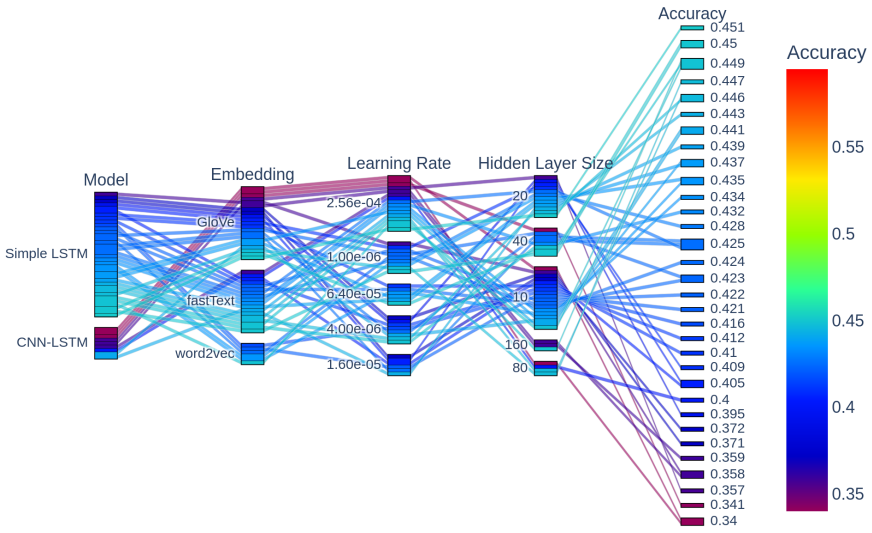

Figure 5.9: 15% worst results on the RSNLI dataset.

Figure 5.9 shows the 45 worst results on the RSNLI dataset. The accuracy range in the image goes from 0.340 to 0.451, a variance of 11.1%, significantly more than the variation seen in the higher end. Among these results, the Simple LSTM was used in 36 (80%). The GloVe embedding was used in 21 (46.7%), while the fastText was used in 18 (40%). The most common learning rate was 2.56e-04, with 16 (35.6%) of the results, and the most common hidden layer size was 10, with 18 (40%).

Table 5.3 shows the best and worst results for each model-embedding combination. The largest range is of 17.5%, the highest yet, while the smallest range is of only 1.6%, which is also the lowest among all datasets. This maximum difference is 17.5% is around 68.6% of the maximum variation for all models in the dataset, which is 25.5%. The average range was of 7.2%, while the median was of 5.7%. These results indicate that the hyperparameter tuning had a significant effect on the performance of the models.

5.5 Comparison of results across datasets

Figures 5.10, 5.11 and 5.12 show a comparison of results between each pair of datasets. The Ordinary Least Square trendline was used to track the correlation between results of

Model	Embedding	Worst	Best	Range
Concatenation LSTM	word2vec	0.565	0.595	0.030
Concatenation LSTM	GloVe	0.522	0.579	0.057
Concatenation LSTM	fastText	0.551	0.567	0.016
Dual LSTM	word2vec	0.542	0.593	0.051
Dual LSTM	GloVe	0.527	0.579	0.052
Dual LSTM	fastText	0.526	0.558	0.032
CNN-LSTM	word2vec	0.483	0.540	0.057
CNN-LSTM	GloVe	0.340	0.515	0.175
CNN-LSTM	fastText	0.359	0.510	0.151
Simple LSTM	word2vec	0.416	0.486	0.070
Simple LSTM	GloVe	0.358	0.476	0.118
Simple LSTM	fastText	0.405	0.465	0.060

Table 5.3: Results for each model-embedding combination on the RSNLI dataset.

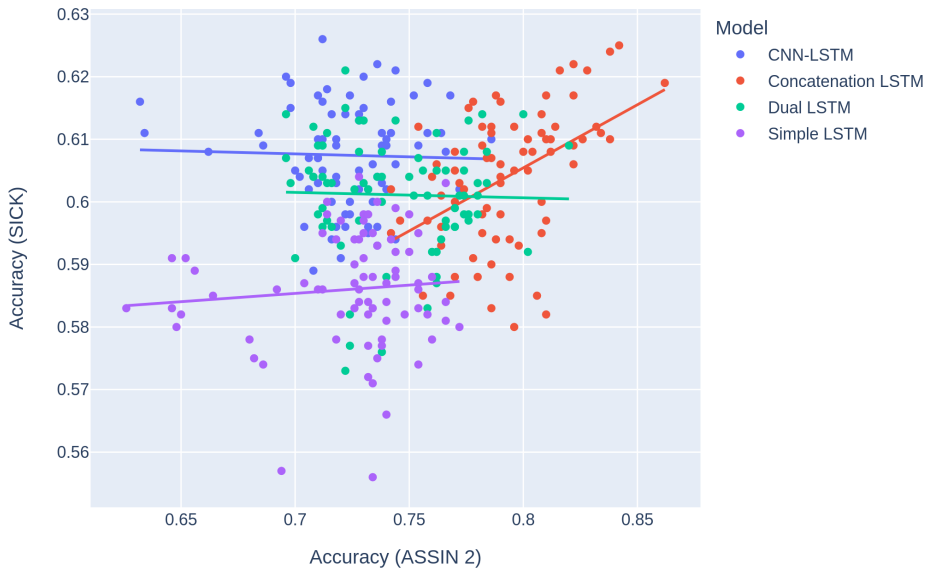

Figure 5.10: Comparison between the results on ASSIN 2 and SICK.

different models. It is possible to see that, in most cases, there was only a small correlation between the accuracies of models in any two datasets.

There were four cases where the coefficient of determination (R^2), that is, the percentage of the variability of the accuracy in a dataset that is predictable using the results in another dataset, was greater than or equal to 10%. In the comparison between the results in the ASSIN 2 and SICK datasets, the Concatenation Model had an R^2 of 21.7%. When comparing the results of ASSIN 2 and RSNLI, the Simple LSTM model had an R^2 of 12.8%, and the Concatenation LSTM model had 29.1%. An interesting aspect is the

Figure 5.11: Comparison between the results on ASSIN 2 and RSNLI.

latter case which presented a negative correlation between the results of the two datasets. Finally, for the comparison between the SICK dataset and the RSNLI dataset, the Simple LSTM model had an R^2 of 36.9%, the highest found.

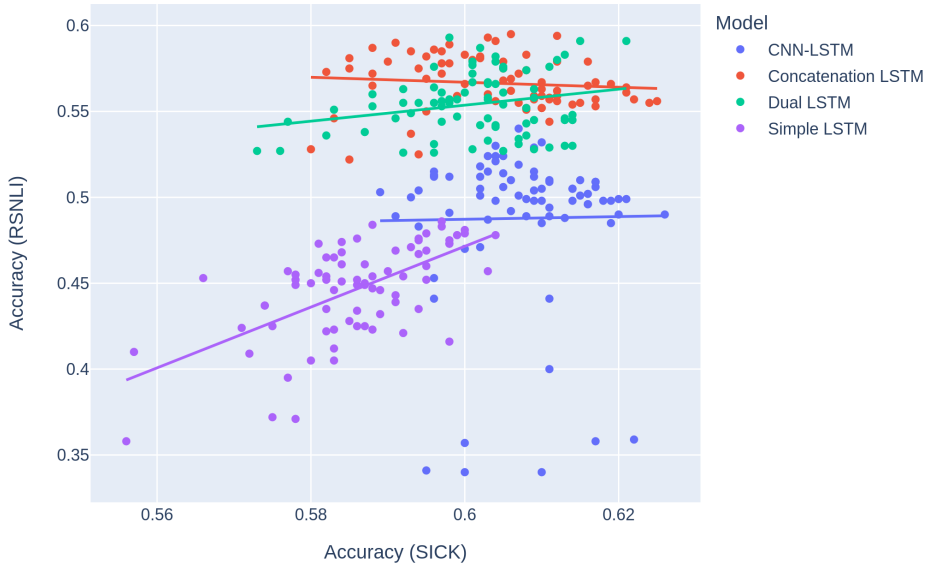

Figure 5.12: Comparison between the results on SICK and RSNLI.

Table 5.4 shows the Spearman's correlation coefficients for the same data. This was calculated by comparing the accuracy values of each set of model architecture, embedding and hyperparameters on different datasets. Note that the previously highlighted models with high coefficient of determination also had a high Spearman's correlation coefficient, confirming the previous findings. Additionally, this coefficient shows Concatenation LSTM models in the SICK × RSNLI comparison as also having a significant negative correlation.

Comparison	**ASSIN 2 × SICK**	**ASSIN 2 × RSNLI**	**SICK × RSNLI**
Simple LSTM	0.002 (9.87e-01)	0.229 (4.79e-02)	0.568 (1.07e-07)
CNN-LSTM	0.011 (9.26e-01)	-0.179 (1.24e-01)	-0.097 (4.06e-01)
Dual LSTM	-0.084 (4.75e-01)	-0.184 (1.15e-01)	0.137 (2.42e-01)
Concatenation LSTM	0.471 (1.99e-05)	-0.614 (4.82e-09)	-0.274 (1.75e-02)

Table 5.4: Spearman's rank correlation coefficient for each dataset-model combination. The numbers between parenthesis are the p-values of the respective coefficients.

Let's also compare the correlation in performance for each learning rate and hidden layer size combination. Figures 5.13, 5.14 and 5.15 show heatmaps for the average performance of the models with only these two parameters varying. Using these averages, it is now possible to calculate the Spearman's correlation coefficient for each pair of dataset with respect to the variation of learning rate and hidden layer size.

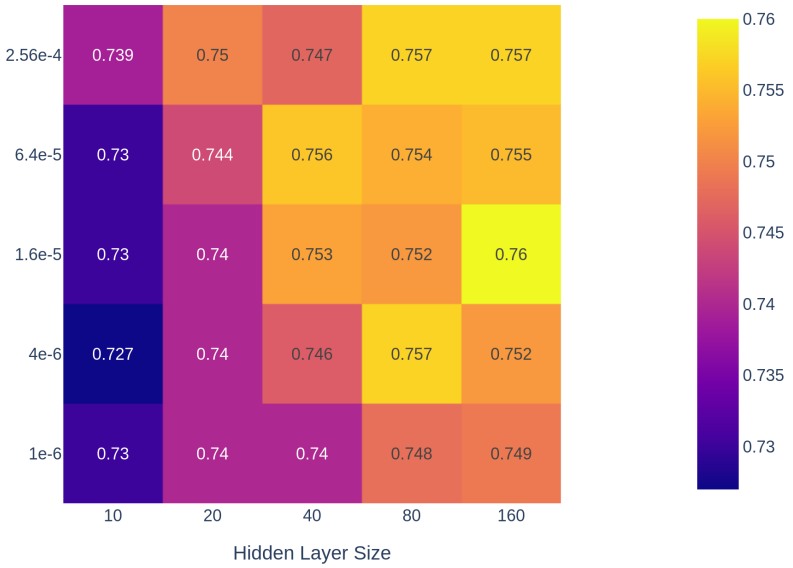

Figure 5.13: Heatmap of the average performance for each learning rate-hidden layer size combination in the ASSIN 2 dataset.

These correlation coefficients are: 0.744 (p-value of 1.9e-05) for the ASSIN 2 x SICK comparison; 0.643 (p-value of 5.24e-04) for ASSIN 2 × RSNLI; and 0.520 (p-value of 7.70e-03) for SICK × RSNLI. This indicates that the hyperparameters that performed well on any dataset also had a tendency of performing well on the others. Notably, the ASSIN 2 and the SICK had a significantly high correlation with each other, despite being written in different languages.

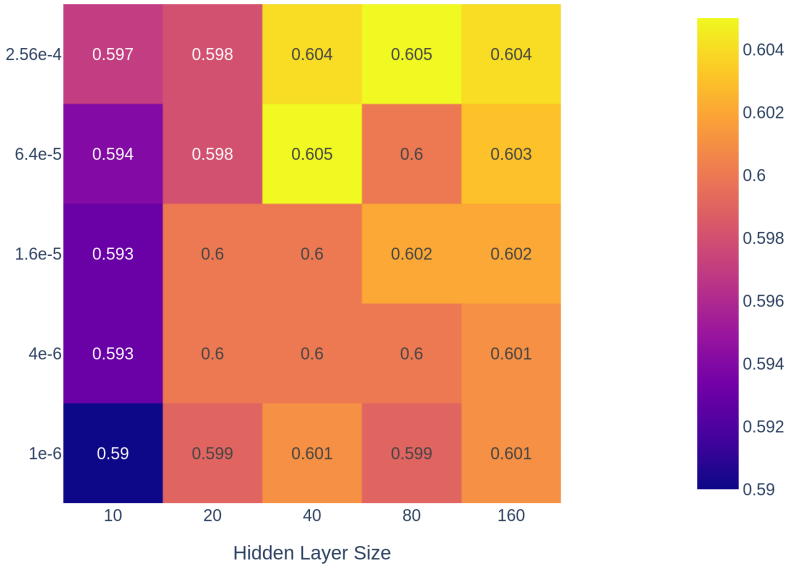

Figure 5.14: Heatmap of the average performance for each learning rate-hidden layer size combination in the SICK dataset.

5.6 Comparison with other models

In this section, the results obtained in this research will be compared to other results from the literature for the same datasets. Note that the model chosen to represent this work in the comparison is not necessarily the one that had the highest accuracy on the test set, but rather, the one that had the lowest loss on the development set. This is to avoid an unfair comparison.

5.6.1 ASSIN 2

For the ASSIN 2 dataset, the model with the lowest loss on the development set used the Concatenation LSTM architecture, fastText as embedding, $2.56 \cdot 10^{-4}$ as learning rate, and 20 as hidden layer size. It was also the one that obtained the highest accuracy, 86.2%.

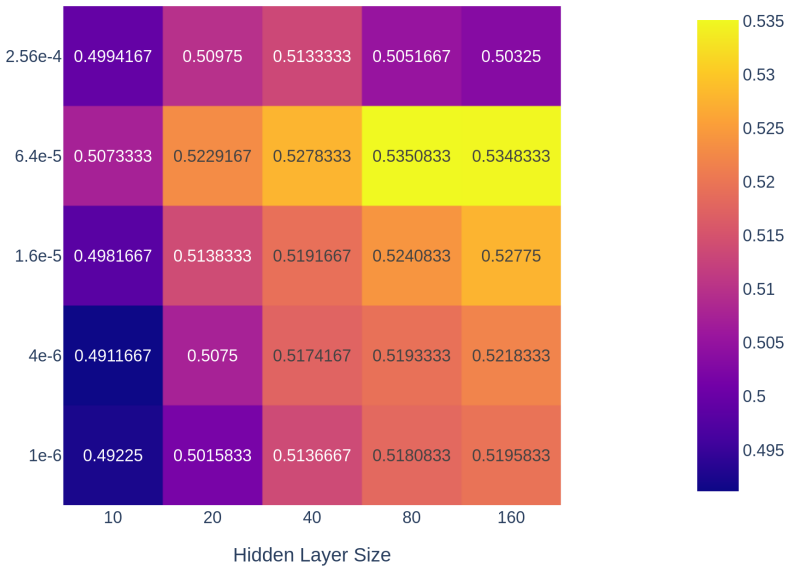

Figure 5.15: Heatmap of the average performance for each learning rate-hidden layer size combination in the RSNLI dataset.

Its performance will be compared with the results of multiple teams shown in the original ASSIN 2 paper [43].

Team	**Accuracy**
Deep Learning Brasil	88.32
IPR	87.58
NILC	87.17
Stilingue	86.64
This research	**86.20**
L2F/INESC	78.47
LIACC	77.41
CISUC-ASAPPpy	66.67
CISUC-ASAPPj	62.05

Table 5.5: Results on the ASSIN 2 compared with other models.

The ASSIN 2 paper mentions that the IPR, Deep Learning Brasil, L2F/INESC and Stilingue teams explored BERT [50] contextual word embeddings. Despite not reaching state-of-the-art, it is impressive that a simple model such as the Concatenation LSTM could achieve an accuracy close to much more complex architectures. This could be concluded to be, in part, due to the hyperparameter tuning and testing of multiple embeddings.

5.6.2 SICK

The model with lowest loss on the development set of SICK used Concatenation LSTM, fastText as embedding, learning rate of $2.56 \cdot 10^{-4}$, and hidden layer size of 40. It was the second model with best performance on the test set of SICK, achieving 62.5% accuracy. This performance beats the baseline of the original paper [41], of 56.7%.

However, it is still much lower than what new models usually achieve. In [16], for instance, an accuracy of 71.8% was achieved, increasing to 80.8% with transfer learning. It could be stated that this dataset is too challenging and complex for the model used.

5.6.3 SNLI

On the RSNLI dataset, the model with lowest loss used the Concatenation LSTM architecture, word2vec as embedding, learning rate of $6.4 \cdot 10^{-5}$, and hidden layer size of 160, achieving an accuracy of 59.5%. This model was retrained on the entire SNLI dataset, under the same parameters as its RSNLI version, and achieve an accuracy of 66.11%. This is lower than the baseline of the original paper, of 78.2% [16]. Once again, this suggests that the SNLI dataset is complex for this model.

5.7 Final Considerations

Before concluding this chapter, it is important to revisit the main goals described in Chapter 1. The first main goal was to study the effect of hyperparameter tuning on the NLI task. This was accomplished in sections 5.1 to 5.3, where the best and worst performing models were compared for each dataset. The second main goal was to determine if there is a correlation between the best hyperparameters on the considered datasets and their different languages. This was performed in Section 5.4, in which the results of all models were compared between them. The following chapter concludes this work and suggest next steps for the research on this topic.

Chapter 6

Conclusion

This thesis aimed at improving the literature on hyperparameter tuning in NLI by studying its effects on the performance of models and how the best parameters change across datasets. To achieve this, three datasets were chosen: ASSIN 2, SICK, and a reduced version of the SNLI dataset, which was named RSNLI. Moreover, the proposed comparative study considered 4 LSTM-based architectures of neural networks, 3 embeddings, 5 learning rates, and 5 hidden layer sizes, and tested all possible combinations of these in each of the 3 datasets, for a total of 900 models trained and tested.

During the analysis of the results in each dataset, it was found that hyperparameter tuning had a significant effect on the performance of the proposed models. The highest difference in accuracy found to be caused by this tuning was 17.5%, while the lowest was 1.6%. The average maximum difference of model-embedding combinations was 7.5% for the ASSIN 2 dataset, 3.0% for the SICK dataset, and 7.2% for the RSNLI dataset. The results also showed that, in all of the datasets, the variation caused by hyperparameter tuning reached a significant fraction of the maximum overall variation. These observations suggest that, in the NLI task, this optimization of hyperparameters can improve the performance of a model architecture by a considerable amount.

When comparing the results across datasets, the analysis using ordinary least squares showed that, among the 12 possible combinations of a fixed model and a fixed comparison (e.g. ASSIN 2 $\times$ SICK), 8 had a coefficient of determination of less than 10%, 3 had a coefficient between 10% and 30%, and only one had a coefficient greater than 30%. Furthermore, it was possible to verify the combinations that had a high correlation using Spearman's Rank Correlation Coefficient. Finally, in a model-invariant and embedding-invariant approach, the average performance was compared for each pair of learning rate and hidden layer size across datasets. The results suggest that there is indeed a relation between the best performing hyperparameters on different NLI datasets.

With the method used in this work, it was possible to verify the effects of hyperparameter tuning on LSTM-based models for the NLI task. However, this approach presents some disadvantages. The fact that the optimization of hyperparameters was effective for the simple models does not imply that it will also be effective for more complex, state-of-the-art models. Additionally, due to time constraints, it was not possible to test a greater amount of hyperparameters, models, and datasets.

This thesis' contribution to the literature is that it supported to demonstrated that optimization of hyperparameters can significantly improve the performance of models in the NLI task, at least for simple architectures. It has also shown that these hyperparameters can be at least somewhat transferable across similar datasets.

6.1 Future work

A direct continuation of this work would be to repeat the experiments using state-of-the-art models, larger datasets, and more hyperparameters. Hyperparameter tuning could be used to not only improve the currently best performing models but also contribute to the understanding of how these parameters affect these models, which could be helpful in constructing new architectures. The difficulty of this, however, is that it would take a long time to do this tuning, and due to that, it may not be a reasonable work prospect.

Instead, some measures could be taken to reduce the training time. For instance, it might be possible to use reduced versions of datasets, such as the RSNLI used in this work, to find good hyperparameters for a model. The complete dataset could then be used to fine-tune these parameters. Another approach would be to use models that are not necessarily state-of-the-art but still perform relatively well in the task of NLI.

References

[1] Yacim, Joseph Awoamim and Douw Gert Brand Boshoff: *Impact of artificial neural networks training algorithms on accurate prediction of property values.* Journal of Real Estate Research, 40(3):375–418, 2018. viii, 7

[2] Uzair, Muhammad and Noreen Jamil: *Effects of hidden layers on the efficiency of neural networks.* In *2020 IEEE 23rd international multitopic conference (INMIC)*, pages 1–6. IEEE, 2020. viii, 8

[3] Elman, Jeffrey L: *Finding structure in time.* Cognitive science, 14(2):179–211, 1990. viii, 11, 12

[4] Murugan, Pushparaja: *Learning the sequential temporal information with recurrent neural networks.* arXiv preprint arXiv:1807.02857, 2018. viii, 13

[5] Rengasamy, Divish, Mina Jafari, Benjamin Rothwell, Xin Chen, and Grazziela P Figueredo: *Deep learning with dynamically weighted loss function for sensor-based prognostics and health management.* Sensors, 20(3):723, 2020. viii, 14

[6] Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean: *Efficient estimation of word representations in vector space.* arXiv preprint arXiv:1301.3781, 2013. viii, 16, 17

[7] Minker, Wolfgang, Marsal Gavalda, and Alex Waibel: *Hidden understanding models for machine translation.* In *ESCA Tutorial and Research Workshop (ETRW) on Interactive Dialogue in Multi-Modal Systems*, 1999. 1

[8] Namazifar, Mahdi, Alexandros Papangelis, Gokhan Tur, and Dilek Hakkani-Tür: *Language model is all you need: Natural language understanding as question answering.* In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7803–7807. IEEE, 2021. 1

[9] Poliak, Adam, Yonatan Belinkov, James Glass, and Benjamin Van Durme: *On the evaluation of semantic phenomena in neural machine translation using natural language inference.* arXiv preprint arXiv:1804.09779, 2018. 1

[10] Demszky, Dorottya, Kelvin Guu, and Percy Liang: *Transforming question answering datasets into natural language inference datasets.* arXiv preprint arXiv:1809.02922, 2018. 1

[11] Yin, Wenpeng, Dragomir Radev, and Caiming Xiong: *DocNLI: A large-scale dataset for document-level natural language inference*. arXiv preprint arXiv:2106.09449, 2021. 2

[12] Dagan, Ido and Oren Glickman: *Probabilistic textual entailment: Generic applied modeling of language variability*. Learning Methods for Text Understanding and Mining, 2004:26–29, 2004. 2, 5

[13] Hickl, Andrew, John Williams, Jeremy Bensley, Kirk Roberts, Bryan Rink, and Ying Shi: *Recognizing textual entailment with LCC's GROUNDHOG system*. In *Proceedings of the Second PASCAL Challenges Workshop*, volume 18, 2006. 2

[14] Bar-Haim, Roy, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, and Bernardo Magnini: *The second PASCAL recognising textual entailment challenge*. 2006. 2

[15] Bowman, Samuel R, Christopher Potts, and Christopher D Manning: *Recursive neural networks for learning logical semantics*. CoRR, abs/1406.1827, 5, 2014. 2

[16] Bowman, Samuel R, Gabor Angeli, Christopher Potts, and Christopher D Manning: *A large annotated corpus for learning natural language inference*. arXiv preprint arXiv:1508.05326, 2015. 2, 6, 20, 43

[17] Hochreiter, Sepp and Jürgen Schmidhuber: *Long short-term memory*. Neural computation, 9(8):1735–1780, 1997. 2, 12

[18] Williams, Adina, Nikita Nangia, and Samuel Bowman: *A broad-coverage challenge corpus for sentence understanding through inference*. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018. `http://aclweb.org/anthology/N18-1101`. 2

[19] Khot, Tushar, Ashish Sabharwal, and Peter Clark: *Scitail: A textual entailment dataset from science question answering*. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2

[20] Schmitt, Martin and Hinrich Schütze: *Sherliic: A typed event-focused lexical inference benchmark for evaluating natural language inference*. arXiv preprint arXiv:1906.01393, 2019. 3

[21] Gururangan, Suchin, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith: *Annotation artifacts in natural language inference data*. arXiv preprint arXiv:1803.02324, 2018. 3, 20

[22] McCoy, R Thomas, Ellie Pavlick, and Tal Linzen: *Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference*. arXiv preprint arXiv:1902.01007, 2019. 3

[23] McCulloch, Warren S and Walter Pitts: *A logical calculus of the ideas immanent in nervous activity*. The bulletin of mathematical biophysics, 5(4):115–133, 1943. 6, 7

[24] Rosenblatt, Frank: *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychological review, 65(6):386, 1958. 7

[25] Block, H Dv, BW Knight Jr, and Frank Rosenblatt: *Analysis of a four-layer series-coupled perceptron. ii.* Reviews of Modern Physics, 34(1):135, 1962. 8

[26] Fukushima, Kunihiko: *Cognitron: A self-organizing multilayered neural network.* Biological cybernetics, 20(3):121–136, 1975. 8

[27] Hinton, Geoffrey E: *How neural networks learn from experience.* Scientific American, 267(3):144–151, 1992. 9

[28] Cunningham, Pádraig, Matthieu Cord, and Sarah Jane Delany: *Supervised learning.* In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008. 9

[29] Weiss, Karl, Taghi M Khoshgoftaar, and DingDing Wang: *A survey of transfer learning.* Journal of Big data, 3(1):1–40, 2016. 9

[30] Zhao, Hang, Orazio Gallo, Iuri Frosio, and Jan Kautz: *Loss functions for image restoration with neural networks.* IEEE Transactions on computational imaging, 3(1):47–57, 2016. 9

[31] Robbins, Herbert and Sutton Monro: *A stochastic approximation method.* The annals of mathematical statistics, pages 400–407, 1951. 10

[32] Kingma, Diederik P and Jimmy Ba: *Adam: A method for stochastic optimization.* arXiv preprint arXiv:1412.6980, 2014. 10

[33] Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams: *Learning representations by back-propagating errors.* nature, 323(6088):533–536, 1986. 10

[34] Bengio, Yoshua, Patrice Simard, and Paolo Frasconi: *Learning long-term dependencies with gradient descent is difficult.* IEEE transactions on neural networks, 5(2):157–166, 1994. 10

[35] Prechelt, Lutz: *Early stopping-but when?* In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998. 11

[36] Jordan, Michael I: *Serial order: A parallel distributed processing approach.* In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997. 11

[37] Fukushima, Kunihiko and Sei Miyake: *Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition.* In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982. 14

[38] LeCun, Yann, Yoshua Bengio, *et al.*: *Convolutional networks for images, speech, and time series.* The handbook of brain theory and neural networks, 3361(10):1995, 1995. 14

[39] Pennington, Jeffrey, Richard Socher, and Christopher D Manning: *Glove: Global vectors for word representation.* In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014. 16, 25

[40] Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov: *Enriching word vectors with subword information.* Transactions of the association for computational linguistics, 5:135–146, 2017. 16, 25

[41] Marelli, Marco, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli: *A sick cure for the evaluation of compositional distributional semantic models.* In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, 2014. 19, 43

[42] Real, Livy, Ana Rodrigues, Andressa Vieira e Silva, Beatriz Albiero, Bruna Thalenberg, Bruno Guide, Cindy Silva, Guilherme de Oliveira Lima, Igor Câmara, Miloš Stanojević, *et al.*: *Sick-br: a portuguese corpus for inference.* In *International Conference on Computational Processing of the Portuguese Language*, pages 303–312. Springer, 2018. 20

[43] Real, Livy, Erick Fonseca, and Hugo Gonçalo Oliveira: *The ASSIN 2 shared task: a quick overview.* In *International Conference on Computational Processing of the Portuguese Language*, pages 406–412. Springer, 2020. 20, 42

[44] Magnini, Bernardo, Roberto Zanoli, Ido Dagan, Kathrin Eichler, Günter Neumann, Tae Gil Noh, Sebastian Pado, Asher Stern, and Omer Levy: *The excitement open platform for textual inferences.* In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 43–48, 2014. 20

[45] Cases, Ignacio, Minh Thang Luong, and Christopher Potts: *On the effective use of pretraining for natural language inference.* arXiv preprint arXiv:1710.02076, 2017. 21

[46] Pang, Deric, Lucy H Lin, and Noah A Smith: *Improving natural language inference with a pretrained parser.* arXiv preprint arXiv:1909.08217, 2019. 21

[47] Bird, Steven, Ewan Klein, and Edward Loper: *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009. 25

[48] Hartmann, Nathan, Erick Fonseca, Christopher Shulby, Marcos Treviso, Jessica Rodrigues, and Sandra Aluisio: *Portuguese word embeddings: Evaluating on word analogies and natural language tasks.* arXiv preprint arXiv:1708.06025, 2017. 25

[49] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean: *Distributed representations of words and phrases and their compositionality.* Advances in neural information processing systems, 26, 2013. 25

[50] Devlin, Jacob, Ming Wei Chang, Kenton Lee, and Kristina Toutanova: *Bert: Pre-training of deep bidirectional transformers for language understanding.* arXiv preprint arXiv:1810.04805, 2018. 42