FEDERAL UNIVERSITY OF TECHNOLOGY – PARANÁ

RICARDO CORSO FERNANDES JÚNIOR

IMPROVING MODEL PERFORMANCE: COMPARING COMPLETE FINE-TUNING WITH PARAMETER EFFICIENT LANGUAGE MODEL TUNING ON A SMALL, PORTUGUESE, DOMAIN-SPECIFIC, DATASET

MEDIANEIRA

2022

RICARDO CORSO FERNANDES JÚNIOR

IMPROVING MODEL PERFORMANCE: COMPARING COMPLETE FINE-TUNING WITH PARAMETER EFFICIENT LANGUAGE MODEL TUNING ON A SMALL, PORTUGUESE, DOMAIN-SPECIFIC, DATASET

Melhorando a Performance do Modelo: Comparando Ajuste-fino Completo com Ajuste-fino Eficiente de Modelos de Língua em um Conjunto de Dados Pequeno, em Português e de Domínio Específico

Bachelor's Dissertation presented as a requirement to obtain the title of Bachelor in Computer Science from the Undergradute Program in Computer Science by Federal University of Technology – Paraná.

Advisor: Prof. Dr. Jorge Aikes Júnior

Co-advisor: Prof. Dr. Arnaldo Candido Junior

MEDIANEIRA

2022

RICARDO CORSO FERNANDES JÚNIOR

IMPROVING MODEL PERFORMANCE: COMPARING COMPLETE FINE-TUNING WITH PARAMETER EFFICIENT LANGUAGE MODEL TUNING ON A SMALL, PORTUGUESE, DOMAIN-SPECIFIC, DATASET

Bachelor's Dissertation presented as a requirement to obtain the title of Bachelor in Computer Science from the Undergradute Program in Computer Science by Federal University of Technology – Paraná.

Approval date: November 28^{th}, 2022

Jorge Aikes Júnior
Doctor
Federal University of Technology – Paraná

Alan Gavioli
Doctor
Federal University of Technology – Paraná

Alessandra Bortoletto Garbelotti Hoffmann
Doctor
Federal University of Technology – Paraná

MEDIANEIRA

2022

ACKNOWLEDGEMENTS

I must start by thanking my parents, as they really went through some tough times to always give me access to knowledge and fulfill my needs during my whole life, even not letting me help with household costs in order to focus exclusively in studying. They always did their best to bargain better prices in the best schools they could afford, so I could develop myself the most. Although I'm not very close with my father, both of my parents always set an example of hardworking and trusted me in my endeavours, and for that I am grateful. I am specially thankful for my mother, for all her support throughout my entire life.

I want to thank Arnaldo Candido Junior and Augusto Camargo for teaching, guiding, advising and supporting me through my path in the beginning of my carreer. Arnaldo teached and helped me to learn the whole foundation of my carreer, by having weekly guidance meetups, just so I could follow my curiosity and learn artificial intelligence. Augusto was the first person to truly value my work. He not only trusted me to help him out in his masters class while I was still an undergraduate student, but also helped me figuring carreer paths that fit with my long-term goals and worldviews, as well as helping me through some hard times in my life. Both of you immensely helped me to grow both personally and professionally.

I would also like to thank all friends that crossed my way for the amazing company throughout life. I would like to specially thank my closest friends, even if we parted ways at some point. With all of you, life is bright. Without you, I wouldn't have made it this far.

We can build a much brighter future where humans are relieved of menial work using AI capabilities.
Andrew Ng

ABSTRACT

Deep neural networks, a class of machine learning algorithms, added a huge leap in performance for many different tasks since they won ImageNet competition in 2012. Among the benefited fields, Natural Language Processing (NLP) was specially impacted by the publication of “Attention is All you Need” paper, in 2017, which gave foundation to many posterior advances in the field. Since then, models are getting progressively more accurate, at the cost of getting bigger and more expensive to train. Transfer Learning contributes by enabling the reuse of large Language Models pre-trained parameters, as they are expensive to optimize. It is possible to fine-tune them from the pre-trained model checkpoint for downstream (derived) tasks. This helps with computational costs of training such large models as well as it avoids the need to gather all data needed for such endeavour. Parameter Efficient Language-model Tuning (PELT) strategies tries to deepen fine-tuning advantages by at least maintaining final model performance while fine-tuning as few parameters as possible. This enables two things: even less computational costs and competitive performance on small datasets for fine-tuning. This work leveraged these advantages in order to obtain better model performance on a legal text classification dataset, built during this work. BitFit performance on small, domain-specific, real-world dataset was compared with complete fine-tuning performance. Results have shown that BitFit fine-tuning is more resistant to fine-tuning data noise and, perhaps, solves the “Catastrophic Forgetting” problem. Also, BitFit outperformed complete fine-tuning on 3 out 5 dataset versions. Finally, the model was presented to and amused the brazillian Federal Court of Audits (from Portuguese: TCU - Tribunal de Contas da União).

Keywords: natural language processing; machine learning; artificial neural networks.

RESUMO

Redes Neurais Profundas, uma classe de algoritmos de aprendizado de máquina, ocasionaram um grande salto de performance para várias tarefas diferentes desde que ganharam a competição ImageNet, em 2012. Entre as áreas beneficiadas, Processamento de Linguagem Natural (PLN) foi especialmente impactada desde a publicação do artigo “Attention is All You Need”, em 2017, o qual deu base para muitos dos avanços recentes no campo. Desde então, modelos estão ficando cada vez mais precisos, ao custo de se tornarem maiores e mais custosos de treinar. A Transferência de Aprendizado permite o reuso dos parâmetros pré-treinados de grandes Modelos de Língua. Esse torna possível fazer o ajuste-fino dos modelos à partir do Modelo de Língua pré-treinado para a realização de tarefas afluentes. Isso contribui para a redução dos custos computacionais de treinar um modelo deste tamanho, assim como evita a necessidade de coletar todos os dados necessários para a realização de um pré-treinamento. Estratégias de Ajuste Eficiente de Parâmetros de Modelos de Língua (PELT, do inglês) buscam aprofundar as vantagens do ajuste-fino ao pelo menos manter a performance do modelo com ajuste-fino de todos os parâmetros com o mínimo de parâmetros ajustados possível. Isso permite duas coisas: um custo computacional ainda menor e performance competitiva para ajuste-fino em conjuntos de dados pequenos. Este trabalho utilizou destas vantagens para melhorar a performance do modelo no conjunto de dados de classificação de texto jurídico, feito ao decorrer deste trabalho. Foi comparada a performance entre o ajuste-fino parcial com BitFit e o ajuste-fino completo para um conjunto de dados pequeno, de domínio específico e do mundo real, utilizando-se do ambiente do Google Collaboratory. Os resultados mostraram que o ajuste-fino com BitFit é mais resistente a ruídos nos dados de ajuste-fino e, talvez, resolve o problema de “Esquecimento Catastrófico”. Ajuste-fino com BitFit também superou o ajuste-fino completo em 3 das 5 versões do conjunto de dados construído. Finalmente, o modelo foi apresentado e causou uma boa impressão no Tribunal de Contas da União.

Palavras-chave: processamento de linguagem natural; aprendizado de computador; redes neurais.

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

Pseudo-Acronyms

ANN	Artificial Neural Networks
BERT	Bidirectional Encoder Representations from Transformers
CDF	Cumulative Distribution Function
DNN	Deep Neural Network
GELU	Gaussian Error Linear Unit
GPU	Graphical Processing Unit
LLM	Large Language Model
MLP	Multilayer Perceptron
NLP	Natural Language Processing
OCR	Optical Character Recognition
PELT	Parameter Efficient Language-model Tuning
PPMI	Positive Pointwise Mutual Information
RELU	Rectified Linear Unit
RNN	Recurrent Neural Networks
SiLU	Sigmoid Linear Unit
SOTA	State-of-the-Art
TF-IDF	Term Frequency Inverse Document Frequency

CONTENTS

1 INTRODUCTION

Electronic devices intensified the rise of text generation (KADHIM, 2019), broadening text data availability. As more textual data is available, it is increasingly profitable to have automated pipelines in order to treat and process such data, capable of adding valuable information. One possible way to extract value from text is through NLP.

NLP is the area of research that studies the processing of language and its components. It has been trending for the four to five years prior to this work's publication, mainly driven by new Neural Network applications to NLP, which gave birth to attention based models and, thus, better text classification techniques (TRIGGS *et al.*, 2021). Among these applications to NLP, models based on the Attention Mechanism achieved special notoriety since 2017 (VASWANI *et al.*, 2017), as Attention is a way to process relations between input elements, extracting contextual embeddings from text.

The study of Attention Techniques, as done by Bahdanau, Cho e Bengio (2016), Luong, Pham e Manning (2015), Yang *et al.* (2016) and many others, lead later to the development of the Transformer architecture (VASWANI *et al.*, 2017), and models like BERT (DEVLIN *et al.*, 2019), RoBERTa (LIU *et al.*, 2019), and beyond. Those are widely used in text-related tasks, achieving State-Of-The-Art performance in, but not restricted to, text classification for relevant benchmarks.

Recent work on text classification is mostly in chinese and english languages (CHEN *et al.*, 2022), both which have vast amounts of data available for model pre-training. On lower availability languages, though, pre-trained models are not as robust as in these languages. This leads to performance degradation when compared with more resourceful languages (ABONIZIO *et al.*, 2020).

Models are also getting prohibitively big as Large Language Model (LLM) gets common. In works like Brown *et al.* (2020) and Smith *et al.* (2022), we can exemplify models that are too big to fine-tune on regular computers. In those cases, it is necessary industry-grade infrastructure, which are capable to store all model parameters gradients during training. Those are expensive, making it difficult for smaller companies to afford it.

This work aims at leveraging Parameter Efficient Language-model Tuning (PELT) strategies (MAO *et al.*, 2021) in order to improve completely fine-tuned model results. These strategies attempt to more efficiently learn features from data by reducing the parameter search space (using a subset of all parameters) as much as possible, and thus reducing the risk of overfitting (MAO *et al.*, 2021) while also reducing total memory/processing consumption (as you optimize approximetely 5% of all parameters). These methods are specially useful for limited data availability environments, which is the case for the Portuguese language when we compare data availability in English, with work being done exclusively to deal with such problems (ABONIZIO *et al.*, 2020). This work uses BitFit (ZAKEN; RAVFOGEL; GOLDBERG, 2021) as its PELT strategy,

in order to more efficiently train models, both in terms of data and in terms of parameters/energy consumption. BitFit aims at solely optimizing bias parameters.

This research results from a legal texts processing company applying Natural Language Processing techniques, in order to extract valuable information from legal texts. They use an automated pipeline for document processing, which was the solution found to meeting their client's document classification large demands. Legal text classification is specially useful in large corporations, which make heavy use of NLP and attention-based methods, as the number of documents gets prohibitively high. It is a core part of intelligent pipelines for automated legal text processing, which helps humans to deal with such large number of documents, by automating smaller tasks which would otherwise consume a worker's time (NOGUTI; VELLASQUES; OLIVEIRA, 2020).

1.1 General and Specific Objectives

The general objective of this work is to develop the best possible legal text classification model, fine-tuned for Portuguese, capable of correctly classifying document paragraphs into 16 different classes. In order to achieve this, BitFit (partial) and complete (default) fine-tuning were compared.

The specific goals of this work are:

- Further fine-tune a legal text classification pre-trained BERT model using BitFit and regular fine-tuning, using the HuggingFace framework to load pre-trained model parameters, in order for it to correctly classify document paragraphs into 16 classes;
- Compare performance of both trained models: a completely fine-tuned BERT model, where all parameters are updated, and a BitFit fine-tuned BERT model, which has only its bias parameters updated during the fine-tuning process.

1.2 Document Structure

This document will be structured as follows: Chapter 2 will present the theoretical background necessary to understand our proposed model. Afterwards, recent related work will be presented, in order to situate this work in the current research context along with recent field advancements and problems. Chapter 3 describes software and hardware necessary to this work, why they were chosen and how they were obtained. Also, the chapter describes the followed methodology, talking about protocols and experiments details, in order to guarantee reproducibility. In Chapter 4, experiments and results will be discussed and analysed. At last, Chapter 5 will present this work conclusions, along with suggestions for future researches.

2 BIBLIOGRAPHIC REVIEW

This chapter will expose a brief overview of the explored topics, in order to guide the reader through the concepts that give foundation to this work.

2.1 Artificial Neural Networks

Deep Artificial Neural Networks (ANN) models are used for classification and regression problems and may be used to support or automate decisions (TRIGGS *et al.*, 2021).

This section describes ANN basics and the field State-Of-The-Art for NLP tasks. Firstly it will introduce the Perceptron, which is the basic unit of a neural network. From the basic unit, Multilayer Perceptrons, Convolutional Neural Networks and Recurrent Neural Networks can be comprehended.

2.1.1 Perceptron

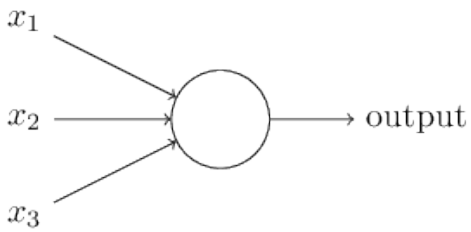
Biological neurons consist of cells controlled by biochemical reactions that control whether it is activated by a given stimuli or not (GOODFELLOW; BENGIO; COURVILLE, 2016). If the inputs to a neuron excite the cell enough (over a certain threshold), it forwards the signal to the rest of the neural network.

Inspired by the biological neuron, a Perceptron is a mathematical model of a biological neuron, which receives inputs, weightening and summing them. If the sum goes over the threshold, the signal is 1, and if it is negative, it is 0. A Perceptron is demonstrated in Figure 1. The Perceptron operation is demonstrated in Equation 1.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j \mathbf{W}_j \mathbf{X}_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j \mathbf{W}_j \mathbf{X}_j > \text{ threshold} \end{cases} \tag{1}$$

Where W_j are the weights associated with j input features X_j. The threshold can be also called the bias term (TRIGGS *et al.*, 2021), as it affects the threshold, making it higher or lower.

Figure 1 – The Perceptron



Source: (NIELSEN, 2018).

2.1.2 Multilayer Perceptron

Multilayer Perceptron (MLP) emerges from a group of Perceptrons, arranged in layers and connected to each other, containing input, output and, often, hidden layers (GOODFELLOW; BENGIO; COURVILLE, 2016). Associating Perceptrons and partnering them with activation functions, it is usual to obtain a model stronger then a simple Perceptron, modelling non-linear relations (NIELSEN, 2018).

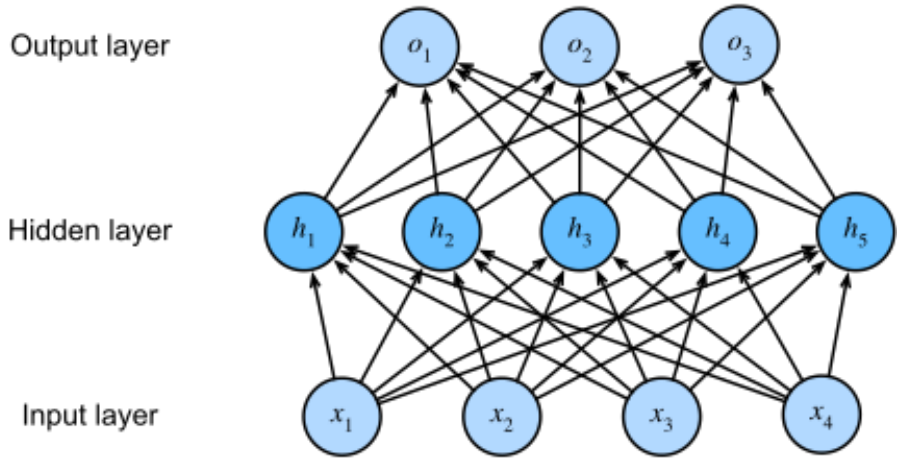
A MLP is an association of perceptrons, so it is many weighted sums, which may be followed by activation functions. Each Perceptron computation inside the network can be represented as shown in Equation 2.

$$O = XW + b \tag{2}$$

Where $X \in \mathbb{R}^{1\times d}$ denotes 1 instance with d features, W represents the Perceptron weights $W \in \mathbb{R}^{d\times 1}$ and $b \in \mathbb{R}^1$ is the bias (TRIGGS *et al.*, 2021). This can be interpreted as pondering over given inputs X through W and adjusting the defined line through b to properly fit the data, splitting the instances into desired classes by defining a decision boundary (the line splitting data points) between target labels being discriminated, and then adding the bias term.

MLPs have an input layer, hidden layers and an output layer, as shown in Figure 2, and its neurons forward each of their outputs to later layers connected neurons, which may or may not be fully-connected (or connected to all previous layer neurons) (TRIGGS *et al.*, 2021). For those connections, fully-connected MLPs may be also named Fully-Connected or Densely-Connected Neural Networks. The difference between a shallow and a deep model is the number of hidden layers, with varying thresholds among bibliographies, but, generally: a neural model that have two or more hidden layers can be considered a Deep Neural Network (DNN) (NIELSEN, 2018).

Figure 2 – Multilayer Perceptron/Fully-Connected Layer



Source: (TRIGGS *et al.*, 2021).

A MLP is a generalization of a Perceptron, and one with two layers can be defined as in Equations 3 and 4 (TRIGGS *et al.*, 2021):

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}), \tag{3}$$

$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}. \tag{4}$$

H is the first hidden layer output, where it is applied the Perceptron computation. $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are layer weights and biases, and in this case $l = 1$ and $l = 2$ for the first and second layers, respectively, but followed by an activation function $\sigma(.)$, which is a commom function to use as an example in most Deep Learning literature. Not all MLPs use an activation function, but it enables them to model non-linear relations, being usually useful for complex problems (TRIGGS *et al.*, 2021).

Sigmoid activation functions were amongst the first activations to be paired with MLP, apart from having some problems. The Sigmoid function will be discussed in Section 2.2, with other activation functions.

2.1.3 Convolutional Neural Networks

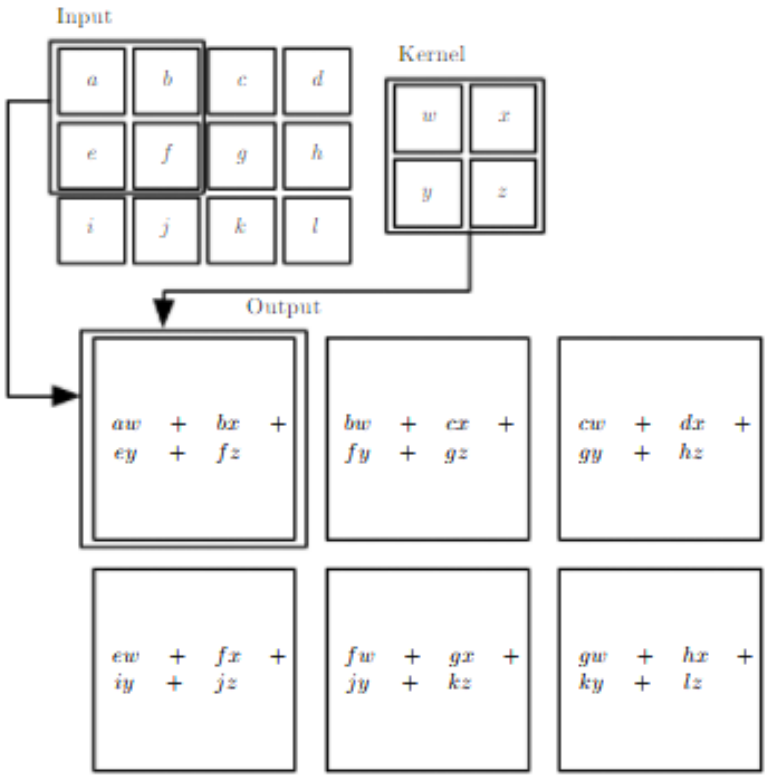
Convolutional Neural Networks are commonly used to learn image features by restraining a MLP to learn local features, organizing the hidden weights as a matrix instead of a vector, and using shared parameters to reduce the algorithm runtime, in a way that it is basically a sliding feature detector (kernel) through the whole image, that automatically learns to discriminate between classes throughout the training process (Backpropagation followed by Gradient Descent based on loss gradients, iteratively), improving the relevance of extracted features. The activation function in each neuron over the matrix representing the output of a instance in a mini-batch detects whether a feature was present or not by the kernel in that neuron receptive field (GOODFELLOW; BENGIO; COURVILLE, 2016).

A convolutional layer is demonstrated in Figure 3. Two principles guided convolutional neural networks design in order for it to work: translation invariance and the locality principle (TRIGGS *et al.*, 2021). The hidden representation calculated by a convolutional layer is given by Equation 5.

$$[\mathbf{H}]_{i,j} = b + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} [\mathbf{W}]_{a,b}[\mathbf{X}]_{i+a,j+b} \tag{5}$$

Where W and X are model weights and inputs, respectively. This is a way of calculating a hidden representation for the pixel in the ith row and jth column, considering information in some range $|a| < \Delta$ or $|b| < \Delta$, where Δ is two times the kernel size. By applying this constraint, $[W]_{ab} = 0$ where $|a| > \Delta$ or $|b| > \Delta$. This interval creates an $a \times b$ matrix called kernel. The kernel works as a sliding feature detector, applying cross-correlation with one kernel

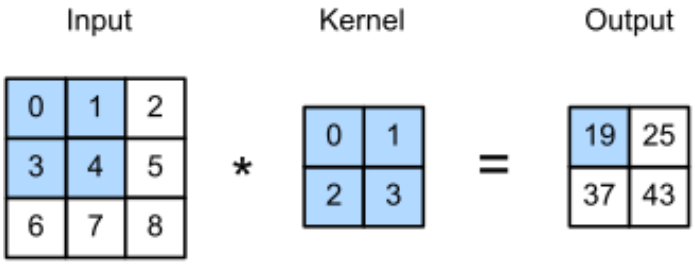
Figure 3 – Convolutional Layer Computation



Source: (GOODFELLOW; BENGIO; COURVILLE, 2016).

per feature to be detected. This is demonstrated in Figure 4, in parallel, across the whole image, classifying where it detects the learned features. The cross-correlation is a very similar operation to a convolution.

Figure 4 – Cross-Correlation Operation



Source: (TRIGGS *et al.*, 2021).

In order to create hidden representations $[\mathbf{H}]_{ij}$ for every region in the input, a convolutional layer will slide its kernel over an entire input (e.g. image, time series, text, previous hidden layer outputs, among others) in parallel (computing every step of the kernel at once), in “steps”, or small translations, called strides, represented in Figure 5. The larger the stride size, the less steps it will take and $[\mathbf{H}]_{ij}$ will have less elements, as it would be ignoring or attributing less weight to some input data. It is also possible to apply padding to a given input in order not to lose input data information in the boundaries, as demonstrated in Figure 5.

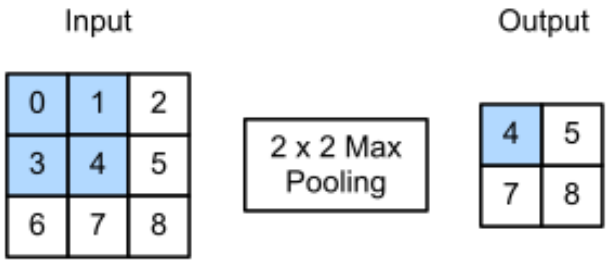
Figure 5 – (a) From 3x3 to 5x5 input matrix with padding and (b) Cross-correlation with strides of 3 and 2 for height and width, respectively

Source: (TRIGGS *et al.*, 2021).

Kernel detects features, but in order to achieve complex relations, a CNN must capture structures formed by features, which can vary in many ways (TRIGGS *et al.*, 2021). For example, if the problem is to detect a face, the distance between both eyes could be considered, and each eye would be something close to a circle (which might be 3 or 4 curves) in the kernel. Eyes size (circle size) and distance may vary greatly with the camera's angle and distance when the image is captured, so the method outputs should be robust against this kind of perturbation in data.

A pooling layer works in a way that solves this problem. As a kernel, the pooling window slides over the hidden representation, computing a single number for each step. The computation applied in each step will, normally, be the maximum or average values of the elements inside the pooling window, as shown in Figure 6. This is useful, in the sense that it captures whether a given feature was detected anywhere inside the pooling window, assuring the model is resistant to small translations and, thus, assuring the model respects the translation invariance principle (GOODFELLOW; BENGIO; COURVILLE, 2016).

Figure 6 – Pooling



Source: (TRIGGS *et al.*, 2021).

Features detected by chained Convolutions and Pooling (Convolutional layers detects features and Pooling layers summarize detected features) operations compose the usual final hidden representation of data after a Convolutional and a Pooling layer (GOODFELLOW; BENGIO; COURVILLE, 2016). These can be passed to a second set of Convolutional and Pooling layers, and the associations between input features will come from previously detected ones, associating them to progressively detect more complex structures. Considering a second Convolutional and Pooling layer, what is being considered as inputs are the outputs from the previous layer that were inferred from a larger set of features (essentially summarizing the previous summarized data). Doing this, the model is receiving information from bigger regions with each extra

Convolutional and Pooling layers, learning relationships with ever more complex structures and encoding them in lower dimensional hidden representations. All elements from input data and previous layers affecting a given computation in some layer are its receptive field (TRIGGS *et al.*, 2021).

At the end of the chained Convolution and Pooling operations, hidden representations are usually flattened into a vector and sent to a classifier.

2.1.4 Sequences and Sequence Models

In order to model sequences, where past events might affect future outcomes, sequence models can be used. This kind of model considers data from the beginning of a sequence x until current time step t input data, as in Equation 6.

$$x_t \sim P(x_t \mid x_{t-1}, x_{t-2}, \ldots, x_2, x_1) \tag{6}$$

Some variation to x_t is expected, but the goal is to learn series dynamics and characteristics that might be helpful in order to solve a given problem and discriminate between classes or predict values (HYNDMAN; ATHANASOPOULOS, 2021; TRIGGS *et al.*, 2021). Unfortunately, $x_{t-1}, \ldots, x_1$ depends on t, eventually becoming computationally intractable. To account for this problem, there are two main strategies: fixing a context window τ, that limit how far to the past a given time series goes, and summarizing all past events into a hidden state h_t, leading to models that estimate x_t with $\hat{x}_t = P(x_t \mid h_t)$.

Given a fixed context window by defining τ that approximates accurately the series, it satisfies a Markov condition and can be considered a Markov Model (TRIGGS *et al.*, 2021). If τ is set to 1, it is a First Order Markov model, which is defined in Equation 7.

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}) \text{ where } P(x_1 \mid x_0) = P(x_1). \tag{7}$$

2.1.5 Recurrent Neural Networks

Markov Models were outperformed by Recurrent Neural Networks (RNN) (TRIGGS *et al.*, 2021), which may consider all or a fixed number of instances in $x_1, ..., x_{t-1}$, summarizing all available contextual information, maintaining it with a hidden state h_{t-1} and being able to work with an arbitrarily long-lasting series, modelling sequences as in Equation 8.

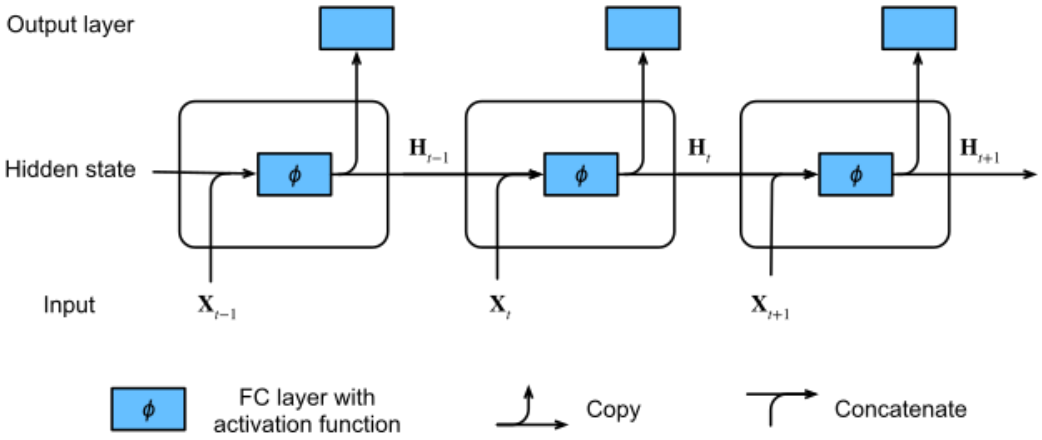
$$P(x_t \mid x_{t-1}, \ldots, x_1) \approx P(x_t \mid H_{t-1}). \tag{8}$$

For a function P capable of modelling linear and non-linear relations across time into a hidden state, this hidden state model can approximate the whole series model, if it is able to store

all information seen so far (GOODFELLOW; BENGIO; COURVILLE, 2016). But simply storing all the data on H_t may get expensive on space and may explode computationally, so it learns a locally optimal way to manipulate H_t by training the model described by Equation 9 (TRIGGS *et al.*, 2021). A RNN is represented in Figure 7.

$$H_t = \phi(X_t W_{xh} \oplus H_{t-1} W_{hh} + b_h). \tag{9}$$

Figure 7 – Recurrent Neural Network



Source: (TRIGGS *et al.*, 2021).

The RNN is learning the parameters W_{xh}, W_{hh} and b_h, while $\mathbf{X}_t \in \mathbb{R}^{1\times d}$ is the input vector in time step t with d features and $\mathbf{H}_t \in \mathbb{R}^{1\times h}$ is the hidden state from the previous time step $t-1$. In practice, it concatenates weights and inputs in order to execute a single operation between the concatenation of weights $(W+W) \in \mathbb{R}^{(h+x)\times h}$ and inputs with hidden state $(X+H) \in \mathbb{R}^{n\times(d+h)}$ (TRIGGS *et al.*, 2021).

RNNs are capable of receiving input and outputting both vectors and vector sequences, in all 4 possible combinations (vector to vector, vector to sequence, sequence to vector and sequence to sequence), using many different architectures in order to model each one. These many different architectures helps modelling many different vector sequence problems (GOODFELLOW; BENGIO; COURVILLE, 2016). This allows the modelling of various tasks in Natural Language Processing, from sentiment analysis (WANG *et al.*, 2016) to text generation (SANTHANAM, 2020), which will be further explored in Section 2.4.

2.2 Activation functions

Activation functions are a way to decide whether a neuron will activate or not. It will map inputs to the neuron into an output in order to introduce non-linearity, thus enabling all neural architectures to model non-linear relationships. This solves inumerous problem, from image to audio and text processing, and beyond (TRIGGS *et al.*, 2021).

Activation functions turn even MLPs (the simplest architecture) with a single hidden layer into Universal Approximators (NIELSEN, 2018) if an unlimited number of Perceptrons is considered in this single layer, as it is able to approximate, theoretically, any given continuous function.

The following sections will describe activation functions, which can be used in all neural network architectures (including Convolutional and Recurrent Neural Networks, which will be seen in following sections).

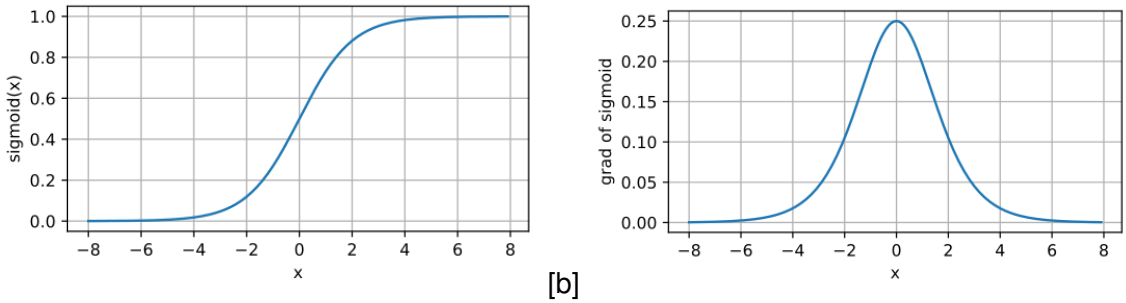
2.2.1 Sigmoid

The Sigmoid function is one of the first activation functions to be popularized, being a differentiable Step function. The definition of a Sigmoid activation funtion can be seen in Equation 10.

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{10}$$

Where x is the function input, usually the neuron activation potential. This function ranges from $(0, 1)$, being useful to map probabilities, and is also entirely differentiable, meaning that it is possible to calculate its gradients without worrying about numerical instabilities. It is monotonic, so gradients will not vary unexpectedly from positive to negative, which may disrupt the training process. For higher magnitudes, vanishing gradient is a problem as function derivative tends to zero. Sigmoid function and derivative graph can be seen in Figures 8)a and 8)b.

Figure 8 – (a) Sigmoid function and (b) its derivative



[a] [b]

Source: (TRIGGS *et al.*, 2021).

For classifying among many different classes, it is possible to use $n - 1$ Sigmoid neurons to model n classes, but this is not the current trend, as Softmax neurons generate a more appropriate class probability distribution, summing to 1 (100%) (TRIGGS *et al.*, 2021). This will be further explored in the Softmax function subsection.

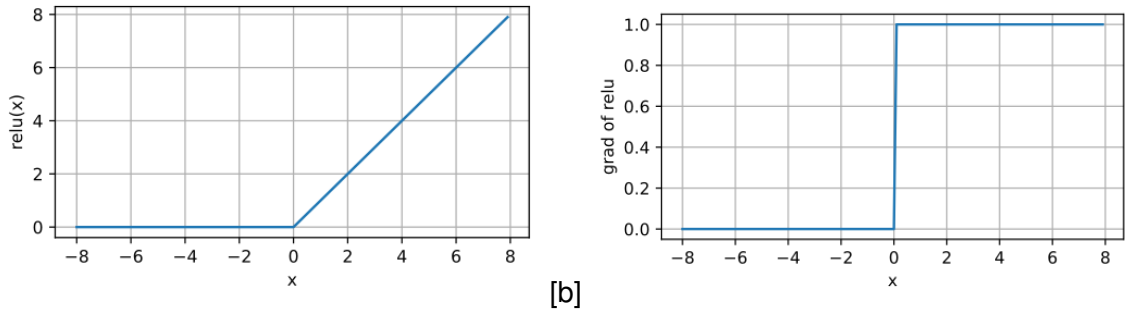
2.2.2 Rectified Linear Unit

The Rectified Linear Unit (RELU) activation function is the most fundamental activation function in NLP, as SOTA models use ReLU (as in Transformer (VASWANI *et al.*, 2017) and Transformer-based architectures like BERT (DEVLIN *et al.*, 2019)) or ReLU-based activations like Gaussian Error Linear Unit in RoBERTa (LIU *et al.*, 2019) and in the 2021 novelty Ernie 3.0 (SUN *et al.*, 2021), from Baidu. ReLU is defined in Equation 11:

$$\mathrm{ReLU}(x) = \max(x, 0) \tag{11}$$

While monotonic and fully differentiable, ReLU is, most of all, efficient to compute. Its gradient is always one for positive values, which accelerates gradient dependent calculations during training. ReLU allows for faster convergence, even being prone to overfitting (HENDRYCKS; GIMPEL, 2020). The ReLU function and its derivative can be seen in Figure 9)a and 9)b, respectively.

Figure 9 – (a) Rectified Linear Unit - ReLU and (b) its derivative



[a] [b]

Source: (TRIGGS *et al.*, 2021).

2.2.3 Gaussian Error Linear Unit

To diminish the overfitting problem, Gaussian Error Linear Unit (GELU) was proposed as a variation to ReLU, which introduces a small insaturation and tends to zero when applied to negative values (HENDRYCKS; GIMPEL, 2020), as shown in Equation 12 and can be seen in Figure 10.

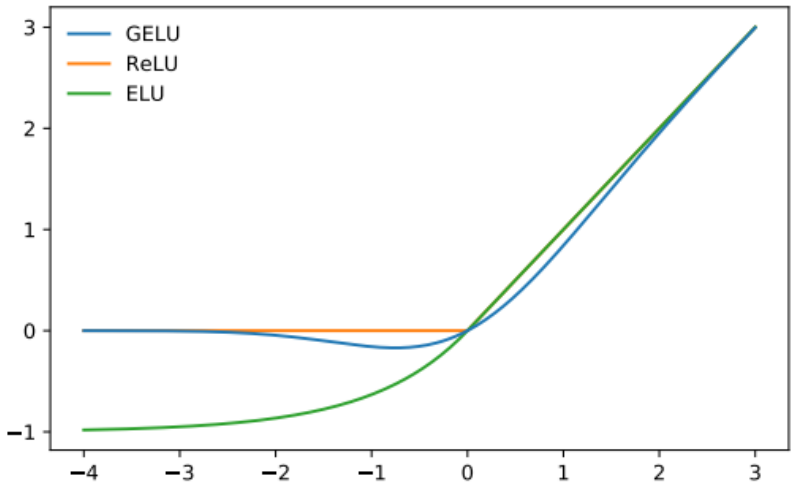
$$\mathbf{GELU}\,(x) = xP\,(X \leq x) = x\Phi\,(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}(x/\sqrt{2})\right] \tag{12}$$

If $X \sim \mathcal{N}(0,1)$.

Where x is the input value, which usually corresponds to a neuron activation potential, $\mathcal{N}(0,1)$ is the normal distribution with a mean of 0 and standard deviation of 1, $\Phi\,(x)$ is the standard Gaussian cumulative distribution function and erf is an error function. Here, $P\,(X \leq x)$

means the probability that a sample from the input x variable distribution (if, X is normally distributed) is smaller than the input x.

Figure 10 – Gaussian Error Linear Unit comparison



Source: (HENDRYCKS; GIMPEL, 2020).

GELU is not monotonic, which means gradients may vary its sign, regularizing the model implicitly (HENDRYCKS; GIMPEL, 2020). GELU can also be approximated to optimize efficiency, as demonstrated in Equations 13 and 14:

$$0.5x\left(1+\tanh\left[\sqrt{2/\pi}\left(x+0.044715x^3\right)\right]\right) \tag{13}$$

or

$$x\sigma\left(1.702x\right) \tag{14}$$

Where σ is the sigmoid activation and x is the input value. Different Cumulative Distribution Function (CDF) could be used, as the proposed Sigmoid Linear Unit (SiLU) (HENDRYCKS; GIMPEL, 2020), but they currently have no major relevance in literature.

2.2.4 Softmax

The Softmax operation outputs a probability distribution, being useful for classification tasks (where discrete classes are default) and for measuring class probabilities or general feature importances (VASWANI *et al.*, 2017; TRIGGS *et al.*, 2021).

An output $\hat{y}_j$ belong to a given class j, and the one which has the highest output value $\operatorname{argmax} y_j$ is the most probable class (TRIGGS *et al.*, 2021). For example, if there are 3 classes $\hat{y}_1$, $\hat{y}_2$ and $\hat{y}_3$, with 0.1, 0.3 and 0.6 as probabilities, respectively, the output class would be $\hat{y}_3$.

The Softmax function transforms the logits of a model in a way that they all become non-negative and sum 1, while still being a differentiable function. In order to achieve its output,

Softmax applies exponentiation to each logit to ensure non-negativity and divides by their sum, ensuring that they sum to 1, as in Equation 15.

$$\hat{\mathbf{y}} = \mathrm{Softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}. \tag{15}$$

Each term in the output is the percentage of the total logit sum that is composed by the class neuron output logit.

2.3 Loss Functions

It is possible to view a Neural Network as a function $NN(X)$, where X is the input data, and the objective is to adjust its parameters in order to fit the function $NN(X)$ to a desired vector y. Taking this statement into consideration, it is possible to say that, with the goal of adjusting model parameters to output a desired output, one must minimize a loss function that demonstrates the difference between the model output vector $\hat{y}$ and the correct output vector y (NIELSEN, 2018).

In order to adjust model parameters and minimize the loss function, it is first necessary to initialize and predict labels for the used dataset with a pseudo-random model (TRIGGS *et al.*, 2021). To compare model outputs ($NN(X) = \hat{y}$) with target outputs (y), applying quadratic loss function is possible, as it compares $\hat{y}$ and y. This is demonstrated in equations 16 and 17.

$$\mathbf{L}(\mathbf{w},b) \equiv \frac{1}{2n} \sum_x \|NN(x) - y\|^2 \tag{16}$$

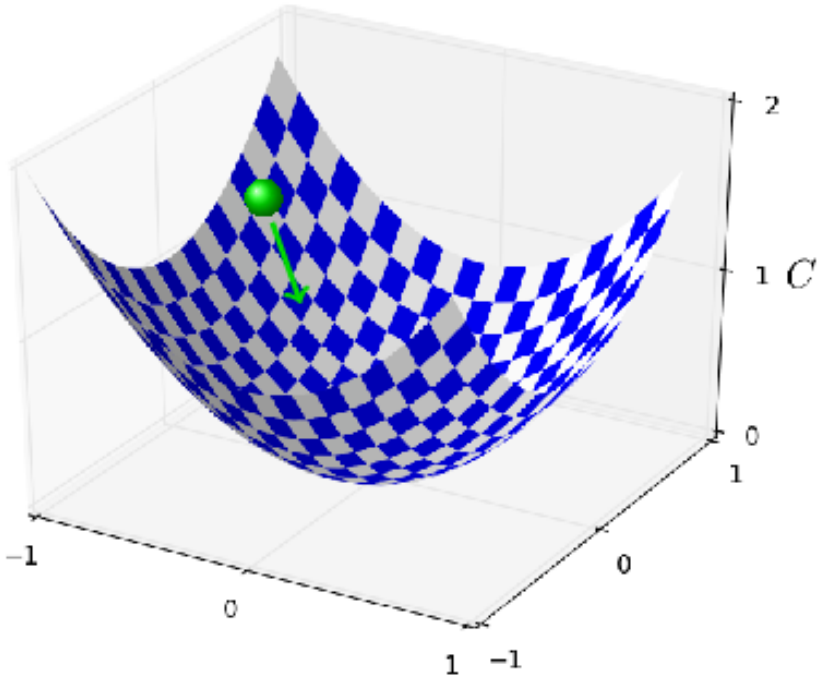
or

$$\mathbf{L}(\mathbf{w},b) \equiv \frac{1}{2n} \sum_x \|\hat{y} - y\|^2 \tag{17}$$

Where n is the total number of instances in the current batch and x is a single instance in the batch.

It is noticeable that the loss function is written in terms of weights W and biases b, which are the model parameters. It is possible to calculate the partial derivatives with respect to model parameters, as a way to measure each parameter importance, and use the loss function average gradient (a vector composed of each parameter derivative [importance]) of the current data batch to slightly move all of the model parameters in the calculated direction, iteratively (NIELSEN, 2018). The gradients are calculated in a backward pass by the Backpropagation algorithm, and parameters are optimized based on the chosen loss function gradient direction. This theoretically leads to a local minimum when following the gradient opposite direction (GOODFELLOW; BENGIO; COURVILLE, 2016), with the Gradient Descent algorithm. A geometric intuitive interpretation for a Gradient Descent variation, Stochastic Gradient Descent, is shown in Figure 11.

Figure 11 – Gradient Descent Intuition



Source: (NIELSEN, 2018).

2.4 Natural Language Processing

Natural Language Processing, or NLP, refers to an active area of research that aims at extracting valuable insight and, more generally, understanding human language from the computer point of view. Words are the most fundamental way to arrange letters in a way to communicate an idea in modern languages, and it is not rare to have exactly the same words meaning completely unrelated things. In Lexical Semantics, a Lemma or Citation Form is the base form of a word, while its many different variations are its Wordforms. For example, a verb Lemma corresponds to its infinitive form, while all conjugations are different Wordforms (JURAFSKY, 2020).

The study of words reveals some recurrent patterns in language, which may be seem as features for machine learning models. The modelling of a language in terms of its lexical statistical properties is a foundation concept to understand Language Modelling, which is the area that gave birth to modern NLP models (TRIGGS *et al.*, 2021).

This section exposes an overview of Lexical Semantics, followed by an explanation of Language Modelling fundamentals through the Distributional Hypothesis.

2.4.1 Lexical Semantics

Sometimes, different words mean exactly the same thing, and those are called Synonyms. Formally, two words are synonyms if they are interchangeable for one another in any sentence without altering the sentence truth conditions, which are situations in which the sentence is true. In this cases, words have the same Propositional Meaning (PARADIS, 2012).

Also, the same meaning may be associated to different words in varying contexts. The Principle of Contrast states that a difference in linguistic form is always associated with some difference in meaning: in Computer Science, when people talk about “C”, the context is usually clear that they actually refer to “C Programming Language”, whereas among chemistry researchers, “C” would more objectively mean “Chemical Element Carbon”.

Word Similarity can help in comparing the similarity between sentences, which can be useful in question answering, summarizations and other language understanding tasks (TRIGGS *et al.*, 2021). A number of datasets where build by humans judging the similarity between words. Words may also be measured in terms of their relatedness or association. One common type of association is the semantic field, that is the domain which a word belongs to. For example, the semantic field for the word “stores” can be “sellers”, “products”, “cashier” and all other “stores” related words. Other associations between words are possible, as hypernymy (IS-A, as in “apple is a fruit”), and meronymy (IS-PART, which comprehend part-whole relations, as in “piece of cake”) (JURAFSKY, 2020). Semantic Frames illustrate the situation participants are in, which can give different meaning to some words (“channel” when you are watching TV and when you are crossing a water channel has different meanings, because they are in different semantic frames). Also, when in a determined situation or event, semantic frames may have roles taken on by words (e.g. in the semantic frame of a university classroom, there are students, teaching assistants and a professor as roles, generally).

Connotation, in NLP, is interpreted as the meaning of words given the writer’s or reader’s emotions, knowledge, beliefs and evaluations (GOUWS, 1996). Some words may have good connotation, like “awesome”, and some may have bad connotation, like “terrible”. It is even possible to have different connotations for the same thing, like “innocent” and “naive”, which are positive and negative, respectively. Positive or negative evaluation is referred to as Sentiment in the literature, being classified in different dimensions in an attempt to capture word meanings. The task of classifying these sentiments in text is known as sentiment analysis.

2.4.2 Language Modelling

Words may have a different meaning depending on the context (semantic frame) they are into. Words that appear in similar contexts tend to have a similar meaning, and this connection between distribution and meaning constitutes the distributional hypothesis. Whether it is to develop spelling correction tools or translate text, it is possible to develop a model that

assign probabilities for next words given current context. This is called Language Modelling (JURAFSKY, 2020). Machine learning, in special neural networks, leveraged language modelling by introducing neural language models and embeddings, which are dense representations of words, first with Static Embedding models with Logistic Regression in Word2Vec, which lead to neural language models like ELMo (PETERS *et al.*, 2018) and Transformer (VASWANI *et al.*, 2017).

2.4.3 Vector Semantics and Word Embeddings

Vector Representations of words is the standard way to encode word meaning in NLP. In the past, before the Neural Network boom in NLP around 2013 (MIKOLOV *et al.*, 2013a; MIKOLOV *et al.*, 2013b; MIKOLOV; YIH; ZWEIG, 2013), sparse word matrix representations through algorithms like Term Frequency Inverse Document Frequency (TF-IDF) and Positive Pointwise Mutual Information (PPMI) were State-of-the-Art (SOTA) for NLP tasks. These algorithms follow the idea of using word frequencies along many different documents in order to compare and discriminate between different topics. Generally, these algorithms would penalize words that are common to every document and classify according to occurrences in some particular documents.

Sparse word matrix representation algorithms used matrix with the size of document vocabulary, in order to achieve a numerical representation of whole texts. But techniques like that could not consider nuances in the context, as they did not capture word order, and thus, would not be enough to capture insights from individual phrases and intra-text relationships. But they were great in order to segment and find articles about specific topics and were relevant for a long time, being able to capture general topics discussed in specific excerpts.

From the idea of representing words with matrixes, the Neural Network boom leveraged NLP by converging to Word2Vec, which advanced lexically semantic representation allowing hundreds of not explicitly interpretable features in exchange of few interpretable features. Word2Vec optimizes an embedding matrix from a large text corpus, producing static Embeddings for every word in the vocabulary from a training process. Then, utilizes a synthetically generated corpus by a skipgram with negative sampling algorithm (MIKOLOV *et al.*, 2013a). This algorithm splits the phrase in tokens and considers a central word to predict if it is a common word given the current context or context window (e.g. “Brazil” relates to “country” just as “France” does). By doing this, the optimization byproducts a embedding matrix for the context and another one for central words. These embeddings holds relations between some words, being able to infer analogies like “King” is to “Man” as “Queen” is to “Woman”. This algorithm also maps embeddings from word frequencies and co-occurences, and successfully extracts some useful embeddings for classification algorithms. Word2Vec is a static dense embedding algorithm, and thus it does not vary due to the context a word is inserted into in prediction time, as it will not have different embeddings for different contexts. Here, the embeddings are only updated considering context

(co-occurence) with the training data, during training time, and the same words used in many scenarios will have always the same embedding, as they are assumed to have independent probabilities (TRIGGS *et al.*, 2021).

In order to measure the similarity of words and, more generally, whole documents, a way is to calculate the cosine similarity between the embedding of two different words, as described by Equation 18.

$$cos(\theta) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} \tag{18}$$

where:

$$\frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{N} a_i b_i}{\sqrt{\sum_{i=1}^{N} a_i^2}\sqrt{\sum_{i=1}^{N} b_i^2}} \tag{19}$$

Where a and b are vectors. This can be seen as a normalized dot product, able to measure vector similarity diminishing the problem of long vector representations (JURAFSKY, 2020).

In section 2.4.4, contextualized word vector representation models will be presented, which produces different embeddings for different contexts.

2.4.4 Contextualized Embeddings

Researchers proposed dynamic embedding models with the advent of dense vector representations now consolidated in the field of NLP. The most relevant for this work are Transformer (VASWANI *et al.*, 2017) and Bidirectional Encoder Representations from Transformers (BERT) (DEVLIN *et al.*, 2019), but they were preceded by the already discussed Word2Vec (MIKOLOV *et al.*, 2013a) and, secondly, by ELMo (PETERS *et al.*, 2018).

ELMo is a deep contextualized dense word representation that models word and character level characteristics, considering bidirectional context. As this model produces embeddings varying based on given word context, they can be considered dynamic or contextualized embeddings. ELMo consists of a deep BiLSTM Neural Network (HOCHREITER; SCHMIDHUBER, 1997; SCHUSTER; PALIWAL, 1997), which is a variant of the RNNs. It considers context both from left and right, for all characters, as it is character-level and bidirectional (PETERS *et al.*, 2018). This model advanced NLP SOTA as it is capable of extracting dense embeddings sensitively to context while also considering morphological[1] features of words.

Transformer model was released in 2017, after further developments in the field of NLP, using the Transformer Attention Mechanism (referred as Scaled Dot-Product Attention Mechanism), that considers a sequence of words represented as tokens and is defined by Equation 20.

[1] Morphology is the branch of linguistics that deals with words, their internal structure, and how they are formed (ARONOFF; FUDEMAN, 2011)

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{20}$$

In Equation 20, $Q, K, V \in \mathbb{R}^{n \times d}$, where n is the sequence length and d is the dense embedding reduced dimension (VASWANI *et al.*, 2017). These three matrixes are obtained from multiplying the current embeddings for each token by trainable matrixes $W_Q, W_K, W_V \in \mathbb{R}^{d \times h}$ where $h < d$, as it is being trained to encode information in a more compact way and d is the original embedding size. The intuition behind Equation 20 is that the product QK^T will measure the similarity between the Q (query) and K (key) embeddings by multiplying every query embedding by every key embedding (essentially relating every token to every other token), scaling by a term defined by the query/key/value embedding dimension square root, in order to maintain numerical stability (large numbers can lead to higher magnitude gradients, that can lead to unstable training) and faster computing. After dividing QK^T by the scaling term, the resulting matrix is used as input to a softmax function, which calculates the importance of every key (of each token in the sequence/sentence, including itself) for every query (essentially relating every token to every other token). The resulting value is multiplied by the matrix V in order to obtain the final attention heads output, which is concatenated with every other attention head output and multiplied by matrix $W \in \mathbb{R}^{h \times d}$, resulting in embeddings of the original size $E \in \mathbb{R}^{n \times d}$. The full Transformer is shown in Figure 12.

The whole attention submodule is formed by the attention mechanism described in Figure 12. It is followed by normalization and skip connection (which is an addition of original input information, before the previous transformation), that is feed to a token-wise MLP, or Fully-Connected Layer. Then, one more normalization step is applied after the skip-connection addition (VASWANI *et al.*, 2017).

Before passing words to the described encoder attention submodule, they are tokenized and transformed into ids, to later project them in the appropriate tensor $E \in \mathbb{R}^{n \times d}$ through sine and cosine functions of different frequencies, as in Equations 21 and 22.

$$\sin(\omega_k.t), \text{if } i = 2k \tag{21}$$

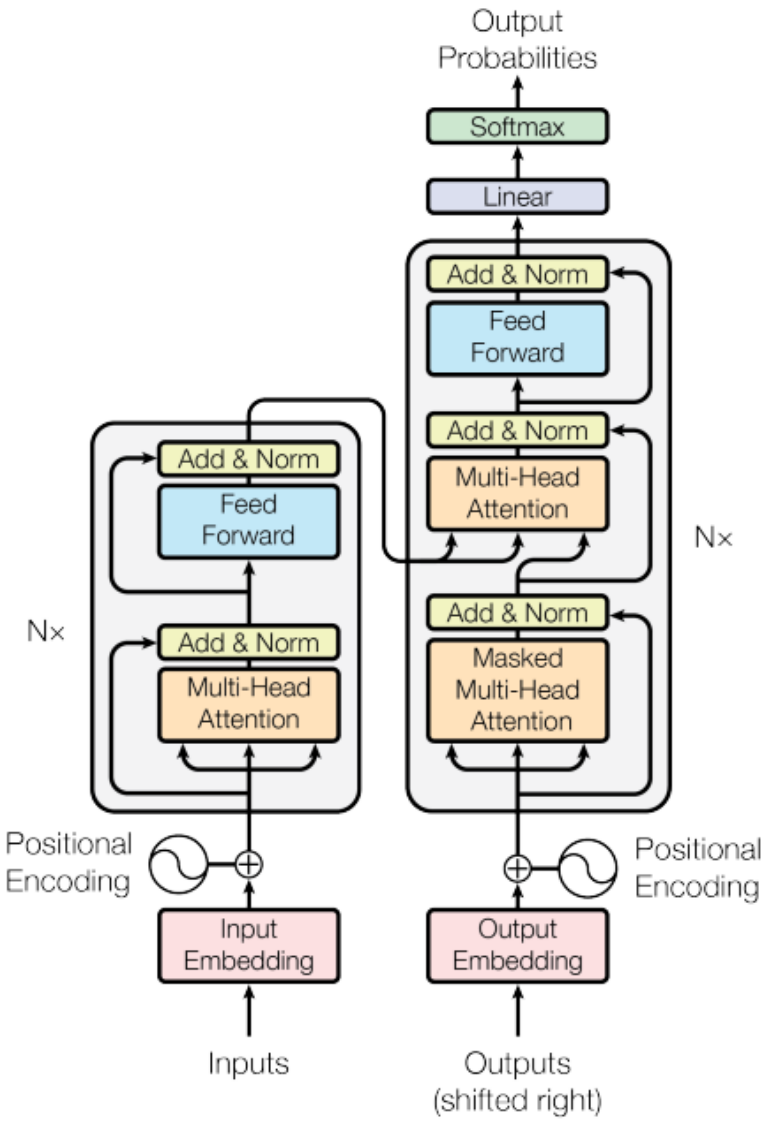
$$\cos(\omega_k.t), \text{if } i = 2k + 1 \tag{22}$$

Where

$$\omega_k = \frac{1}{10000^{2k/d}} \tag{23}$$

Which corresponds to a signal that encodes some positional information for any sequence length.

Figure 12 – Transformer Architecture



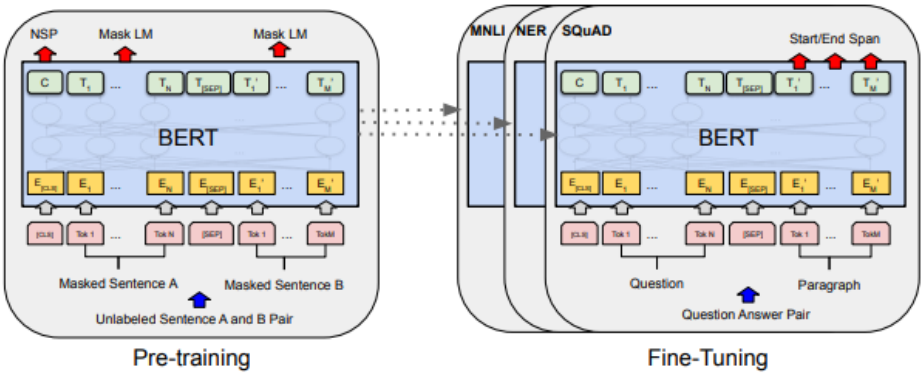
Source: (VASWANI *et al.*, 2017).

Finally, there is the transformer decoder, that generates output using a self-attention sub-module, which performs the Scaled Dot-Product Attention operation on every token, including the previous outputs of the model. As the output is generated recurrently, similar to RNNs, Transformer encodes information in parallel, like CNNs, while generates outputs recurrently, achieving SOTA performance. The model performs next-word prediction for training.

BERT introduced a new training method for Transformer encoders, where it leveraged mlm and nsp in order to direct learning towards high level language features, as can be seen in Figure 13 (DEVLIN *et al.*, 2019). MLM is used to mask 15 out of 100 input words, with 80% chance of the mask being a special token “[MASK]”, 10% chance of being substituted by a random word and 10% chance to be kept as it is. NSP consists of two sentence inputs which BERT predicts whether one follows the other or not. This forces BERT to learn general language

high-level features, as well as low-level features up to the point of word pieces. Those are parts of a word that may contain meaningful information (prefixes, suffixes, conjugation, inflexions, among others) (DEVLIN *et al.*, 2019). More importantly, BERT introduces a feature that was already incorporated by ELMo but wasn't yet seen in Transformer Architecture: Bidirectionality. BERT pre-training procedure enables it to perform well on many downstream (derived) tasks with a bit of fine-tuning.

Figure 13 – BERT Pre-Training Process



Source: (DEVLIN *et al.*, 2019).

2.4.5 Parameter Efficient Language-model Tuning

As LMs got bigger, its training computational cost became increasingly higher, a problem which Transfer Learning tries to solve (HOWARD; RUDER, 2018). Through the usage of pre-trained models, the community was able to reuse huge models without training on an industry-grade Graphical Processing Unit (GPU), a computer component capable of multiplying matrices very efficiently (OH; JUNG, 2004), or GPU clusters, and only fine tuning it for specific use cases, as in (DEVLIN *et al.*, 2019) downstream tasks, which enables researchers and practitioners without much resources to use big LMs.

In years prior to this work, large LMs with billions of parameters (RAFFEL *et al.*, 2020; RADFORD *et al.*,) are being trained, and they tend to get even larger. With that, fine-tuning scales as well, and can sometimes prove to be too expensive, specially with even bigger models on the way considering the trend set by NVidia and OpenAI with 100+ billion parameters LMs (SMITH *et al.*, 2022; BROWN *et al.*, 2020). Those models are also trained on massive amounts of textual data, which is not available in all languages.

In this scenario, storing these models start to be a burden. Fine-tuningand storing a different model for each task is an even bigger burden. This lead us to PELT methods, which optimizes just a subset of model parameters for each fine-tuning task, making it possible to more efficiently store them, as if we optimize solemnly 0.08% of model parameters, as a PELT

method named BitFit proposes (ZAKEN; RAVFOGEL; GOLDBERG, 2021), we store 99.02% of the model once, and the 0.08% subset many times, making it a 99.02% reduction in memory usage for each additional task the pre-trained model is fine-tuned on (MAO *et al.*, 2021).

One of such methods, and the one explored in this work, is BitFit (ZAKEN; RAVFOGEL; GOLDBERG, 2021). Although there are other PELT methods (MAO *et al.*, 2021), this work applied BitFit due to its simplicity. If it proves successful, future work in this project might include experiments with alternative PELT methods.

There are two main types of PELT methods: with and without additional parameters (MAO *et al.*, 2021), and BitFit is classified as a "PELT method without additional parameters". In this category, there is also fine-tuning using only the top layers or prediction head, which generally lead to worse model performance compared to complete fine-tuning (LEE; TANG; LIN, 2019; PFEIFFER *et al.*, 2020). BitFit achieves better performance by fine-tuning bias across the whole model (MAO *et al.*, 2021)

On the other hand, there are PELT methods with additional parameters. Examples are: Adapter (HOULSBY *et al.*, 2019), Prefix-Tuning (LI; LIANG, 2021) and LoRA (HU *et al.*, 2021), which is further classified as an Additive Method. Adapter works by adding a trainable bottleneck (down+up projection pair) layer after the feedforward network in each transformer layer of the LM, which reduce dimensionality and recovers the original input from its representation in reduced dimensions, extracting important features in order to properly recover the input. Prefix-tuning prepends two task-specific trainable prefix matrices to Query and Key values, which acts as virtual tokens and can be attended to, working as task-specific tokens. Lastly, Additive Methods interpret model parameters after fine-tuning as an addition of original pre-trained model parameters $\theta_{pre-trained}$ and task-specific variations δ_{task}. There are various ways to parameterize δ_{task}, which leads to different additive methods; LoRA introduces trainable low-rank matrices and pair them with the origianl matrices in the multi-head attention (MAO *et al.*, 2021).

3 MATERIALS AND METHODS

In this section, software and hardware requirements (sections 3.1 and 3.2) for the present work is explained, as well as the methods followed (Section 3.3).

3.1 Hardware

Hardware requirements for this work were a single computer for model training, analysis and writing/reading, also making heavy use of Google Collaboratory [1] and Google Collaboratory Pro:

- Lenovo Legion laptop, with 1 TB HD and 128 GB SSD and a Intel i7 2.80 GHz CPU, using Windows 10 as Operational System (OS). It has 16 GB RAM and a GTX 1060 GPU with 6 GB memory, in order to realize small scale experiments, prepare data and develop model training/inference pipeline;
- Google Collaboratory cloud environment is hosted on a different machine each time a session is started. The two most common settings included Nvidia K80, T4 or (on collab PRO) P100, with 12 GB, 16 GB and 16 GB of GPU memory, respectively. As for RAM, the environment has 12 GB and 2 CPU Cores, and also has access to the Google Drive environment, allowing for unlimited cloud storage as Federal University of Technology - Paraná sponsors it.

3.2 Software

The software requirements were the programming language (Section 3.2.1.1), frameworks and libraries (Section 3.2.2.2) and datasets (Section 3.2.3.3).

3.2.1 Programming Language

Python was used for the development of this work. Python [2] is a well-known programming language for scientific computation, considered intuitive and flexible, being fit for most scientific problems as it simplifies the implementation process, allowing researchers to focus on the modelling (OLIPHANT, 2007).

Python offers a wide range of advantages, other than ease of use:

- Useful built-in objects: everything in Python is an object, and the language is highly flexible with object types;

[1] https://colab.research.google.com/
[2] https://www.python.org/

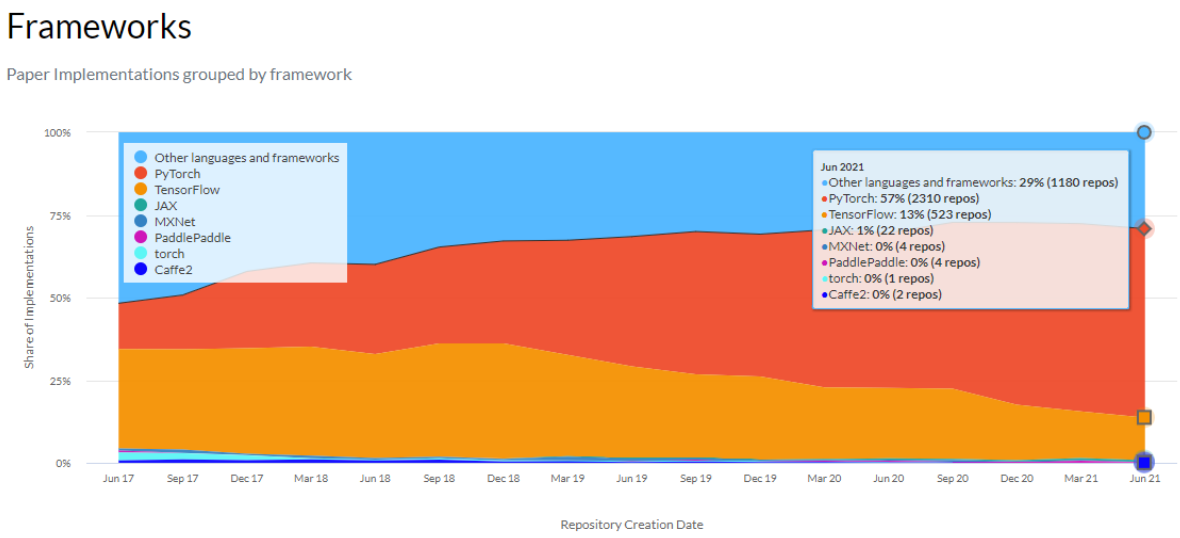
- Functions and classes: Pythons allows for both Procedural Programming as well as Object Oriented Programming, besides other programming paradigms;
- Standard Library: Python's standard library has a plentora of built-in functions and classes, enabling more efficient development;
- Ease of Extension: C or C++ code can be easily integrated with Python pipelines as a way to optimize code;
- Python has, apart from its standard library, a suite of efficient and useful libraries, which include GPU optimized code for Machine Learning.

The most relevant advantage of Python for Data Science tasks are its external libraries, which is described in Section 3.2.2.2.

3.2.2 Libraries

The PyTorch [3] (PASZKE *et al.*, 2019) frameworkd was used for Neural Network modelling, as it surpasses similar frameworks, having more active pull requests than Tensorflow [4] (ABADI *et al.*, 2016) on GitHub at the time of this work, and according to Papers With Code (2021) GitHub framework trends for Deep Learning papers shown on Figure 14.

Figure 14 – GitHub Data for Deep Learning Framework Adoption in Paper Implementations Through Time



Source: (Papers With Code, 2021).

[3] https://pytorch.org/
[4] https://www.tensorflow.org/

In conjunction with PyTorch, Huggingface transformers (WOLF *et al.*, 2020) library was used [5]. It is an open-source library which aims at enabling broader access to transformer models through an unified API, that facilitates machine learning practitioners to leverage Transformer models transfer-learning capabilities. This enables faster experimenting, contributing greatly to the advancement of the field.

For experiments comparison and visualization, both Plotly (INC., 2015) and Weights and Biases (WB) (BIEWALD, 2020) were used. Plotly is a Python data visualization library with customizable and interactive graphs, while WB is used to track and compare multiple experiments as well as to easily monitor energy consumption metrics.

Finally, for the models, legal text fine-tuned BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020) model weights were used, imported from the HuggingFace (WOLF *et al.*, 2019) repository [6]: “Luciano/bert-base-portuguese-cased-finetuned-peticoes”, which had no published paper advertised on the HuggingFace repository page at the time of writing.

3.2.3 Dataset

A dataset was built for the purpose of this project, although it is not the emphasis of this work. The main stakeholder of this project had many unlabeled (unclassified) legal documents, which their clients handed over in order to also be benefited by the project. In order to build the dataset, the following steps were executed:

- Extracted text from PDF as plain text using Optical Character Recognition (OCR);
- Submitted extracted text splitted into paragraphs to the annotation platform;
- Paragraphs were annotated (labeled/classified) by human annotators, hired for this project;
- Annotator agreement was measured through Cohen’s Kappa statistic, after each annotation round, in order to adjust for the next round;
- Annotated data was released as a dataset version, so models could be trained on it.

Those documents were not always in plain text, sometimes being even impossible to select text in some PDF file formats. The responsible team experimented with OCR techniques such as Smith (2007) and Dosovitskiy *et al.* (2021) in order to achieve the most robust/efficient text extraction engine from images, as it impacts in the quality/latency of all data/processing in the project pipeline. An OCR model output example can be seen on Figure 15.

The OCR model works by extracting paragraphs from documents (as full-page texts were giving unstable results), which were annotated in the developed annotating platform. In order to

[5] https://huggingface.co/

[6] https://huggingface.co/Luciano/bert-base-portuguese-cased-finetuned-peticoes

Figure 15 – OCR model output example

Source: (DU *et al.*, 2020).

maintain and monitor dataset quality, Cohen's Kappa statistic was used (FALOTICO; QUATTO, 2015). This dataset statistic is a widely used statistic to verify annotator agreement, and thus annotation consistency, which helps to minimize bias-related data problems.

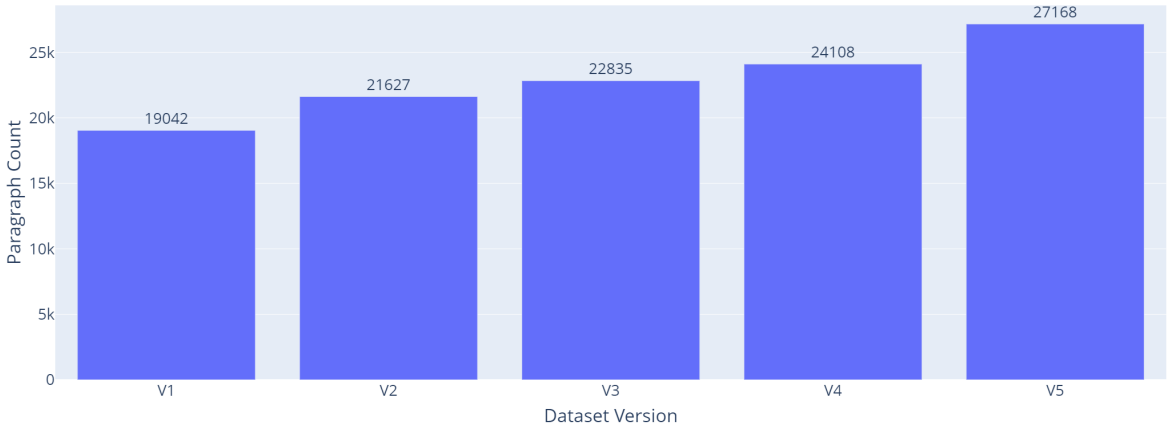
Cohen's Kappa measures annotators answers correlation. It generates a number ranging from -1 (negatively correlated: annotator A answers 0/1 while annotator B answers 1/0 in every sample) to 1 (positively correlated: annotator A and B agree in every sample) for every annotator (WARRENS, 2015). For this work, the perfect scenario consists of annotations that are highly positively correlated (1) for every annotator, meaning that class definitions are perfectly defined among every human annotator, and that the same features are being recognized as the same classes.

Annotators labeled paragraphs along 16 classes: "Dano moral", "Inexistência de relação jurídica/débito", "Inversão do ônus da prova", "Jurisprudência", "Fraude", "Tutela antecipada", "Canais Internos", "Restituição em dobro", "Justiça gratuita", "Inscrição indevida", "Desvio produtivo", "Dano material", "Produtos bancários", "Financiamento veículo", "Multa diária" e "Notificação de inscrição". Those were defined jointly by data scientists and lawyers participating in this project. The dataset has a mean Cohen's Kappa of 0.88, which means that annotators have good agreement rate for data samples.

The dataset was then exported as consecutive Comma Separated Files (CSV) files as annotation time goes on, with each version having more samples (and eventually fixing past mistakes caused by bugs or misconceptions among annotators). Versions were released weekly during the duration of this project. Dataset is not balanced, so all metrics used are weighted. This work don't disclose class distribution information as it used a private dataset, and leaking this information could lead to competitive disadvantages for this work sponsor.

A graph depicting dataset growth through versions is shown in Figure 16.

Figure 16 – Dataset growth over versions



Source: Own authorship (2022).

3.3 Method

This section describes the methodology used in this work. It is organized in the following steps:

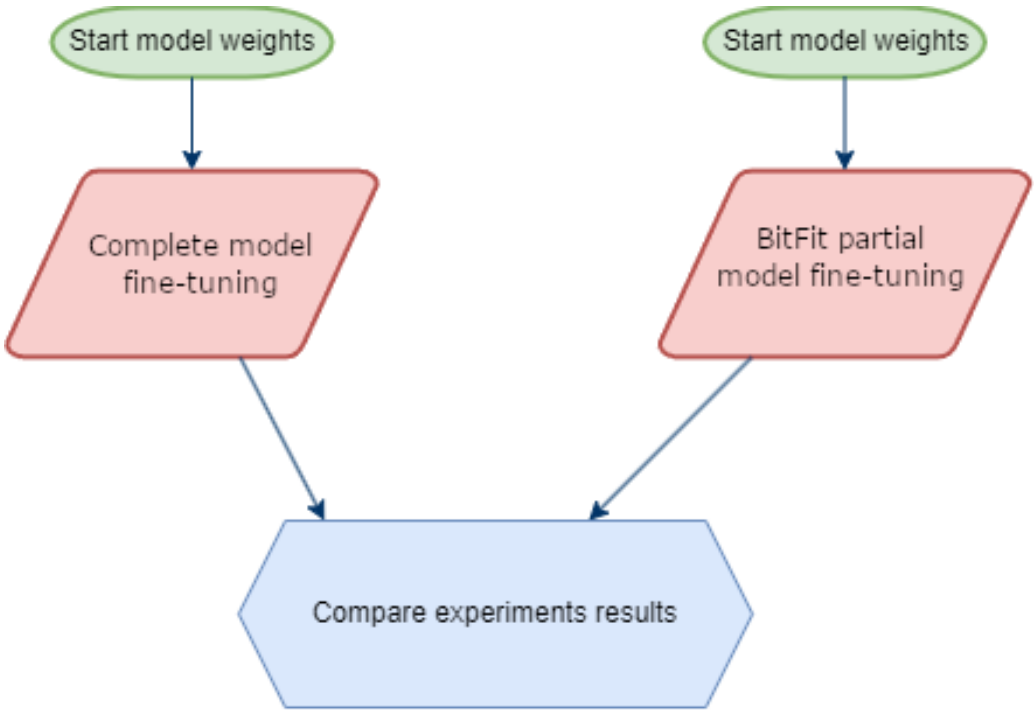
- Step 1: Initialized model weights using HuggingFace pre-trained model weights;
- Step 2: Completely fine-tuned model on created dataset;
- Step 3: Partially fine-tuned model using BitFit on created dataset;
- Step 4: Compared model results among dataset versions, plotted results and discussed model performance on both fine-tuning settings as sample size has grown with each dataset version;

The experiments flowchart can be observed in Figure 17.

3.3.1 Planned Experiments Flow

Methodology steps are further described as the following:

Figure 17 – Experiments Overview Flowchart



Source: Own Authorship (2022).

Step 1: For this work, both BERT models (fully fine-tuned and partially fine-tuned with BitFit (ZAKEN; RAVFOGEL; GOLDBERG, 2021)) were initialized with pre-trained model weights. Model weights came from a fine-tuned version of BERTimbau (SOUZA; NOGUEIRA; LOTUFO, 2020), "bert-base-portuguese-cased-finetuned-peticoes", which is fine-tuned in petitions, with training hyperparameters:

- Learning rate: 2.10^{-5};
- Train batch size: 8;
- Evaluation batch size: 8;
- Optimizer: Adam with betas=(0.9, 0.999) and epsilon=(1.10^{-8});
- Learning rate scheduler type: linear;
- Number of epochs: 3;
- Seed: 42.

In the HuggingFace repository [7], it is also possible to check fine-tuning hyperparameters:

- Attention probabilities dropout probability: 0.1;

[7] https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run$_glue.py$

- Classifier dropout probability: null;
- Directionality: bidirectional;
- Activation function: GELU;
- Hidden dropout probability: 0.1;
- Embedding size/Hidden size: 768;
- Max length: 512;
- Number of attention heads 12;
- Number of hidden layers: 12;
- Pooler fully-connected layer size: 768;
- Vocabulary size: 29794.

Step 2: The first experiment was to perform a complete fine-tuning on the created dataset. Training followed default settings for HuggingFace sequence classification scripts:

- Max text length: 128;
- Learning rate: 5.10^5;
- Number of epochs: 3;
- Gradient accumulation steps: 1;
- Learning rate scheduler type: linear;
- Seed: 42.

Step 3: The second experiment was to partially fine-tune on the created dataset, using BitFit (ZAKEN; RAVFOGEL; GOLDBERG, 2021). Training followed listed settings:

- Max text length: 128;
- Learning rate: 5.10^5;
- Number of epochs: 3;
- Gradient accumulation steps: 1;
- Learning rate scheduler type: linear;
- Seed: 42.

Step 4: After running both experiments, their weighted precision, recall and f1-score were compared. These have been shown to be good performance metrics for model comparison and evaluation (GOUTTE; GAUSSIER, 2005). A high precision model has less false positives, while a high recall model has less false negatives. The f1-score is the harmonic mean over precision and recall, which is punishing if one of the values is much lower than the other.

Comparing each model result for each dataset version made possible to analyse how performance varies along dataset sizes. It was expected that BitFit would perform better on low dataset size scenarios, as it optimizes less parameters in order to train more efficiently. Although this fastens convergence, it was also possible that less information would be learned by the model from training data, as shown in BitFit experiments (ZAKEN; RAVFOGEL; GOLDBERG, 2021), which avoids overfitting in low dataset size scenarios but also prevents the model from learning everything it can from available data.

With this methodology, this work compared BitFit performance with regular language model fine-tuning when data is scarce, which proved useful for experimenting with Transformer models at the start of projects in lower resource languages, where annotated texts may not be as abundantly available, helpíng researches to develop faster.

4 RESULTS

This chapter describes all experiments and their results, as well as discussing those outcomes, following steps described in 3. Firstly, complete model fine-tuning results will be presented and discussed in Section 4.1, followed by partial fine-tuning with bitfit in Section 4.2 and, for conclusion, it is shown a comparison among both of them.

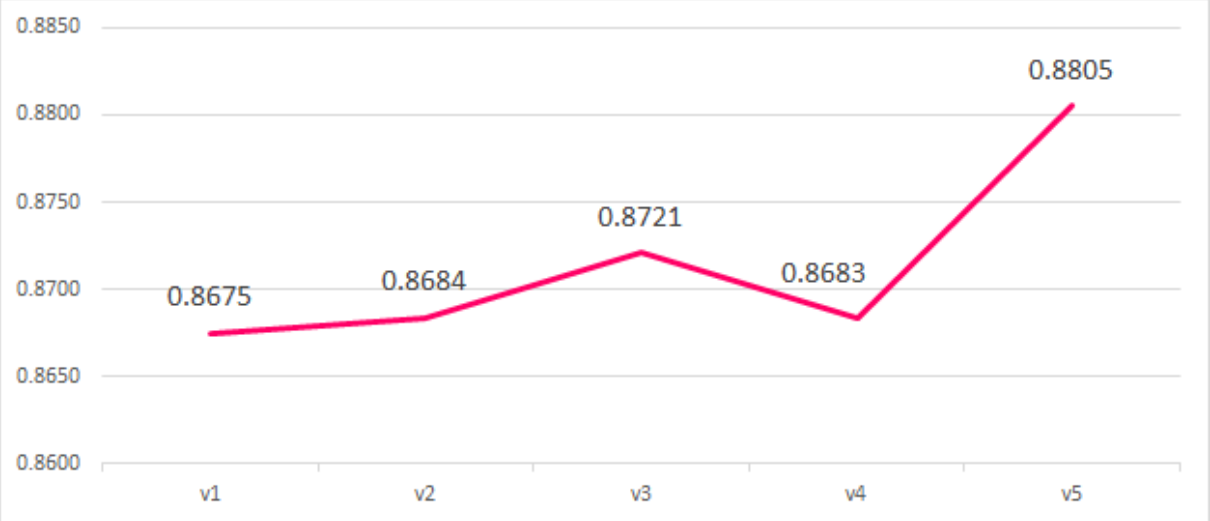
4.1 Complete Fine-tuning

This section presents the first set of experiments, which consists in benchmarking complete fine-tuning performance when training on 5 different versions of our dataset. Each subsequent version has more samples than the previous one, as they are weekly releases from the project annotation team. Results obtained in this step will be further used to compare with partial fine-tuning using BitFit.

Pre-trained parameters from the Huggingface repository named “Luciano/bert-base-portuguese-cased-finetuned-peticoes” were used for the model initialization. Training parameters are set as described in Section 3.3, through the Huggingface library.

For each dataset version, 3 experiments were run. Each had identical parameters and hyperparameters, except for the seeds: 40, 41 and 42. Figure 18 shows mean model F1-Score along dataset versions.

Figure 18 – Complete Fine-tuning F1-Score



Source: Own Authorship (2022).

In general, the model performs better as the dataset grows in size, except from version 3 to 4, the only occurrence of performance degradation. This happenned for a loss in annotation quality (less annotator concordance) during the project, a problem that was investigated and solved for version 5, which has continued model performance improvement.

This model improvement along dataset versions is expected. Following (OLTHOF; OOIJEN; CORNELISSEN, 2021), model performance tends to improve as dataset size grows, with diminishing returns as size increases, up to a plateau.

A table containing all training metric details can be seen in Table 1.

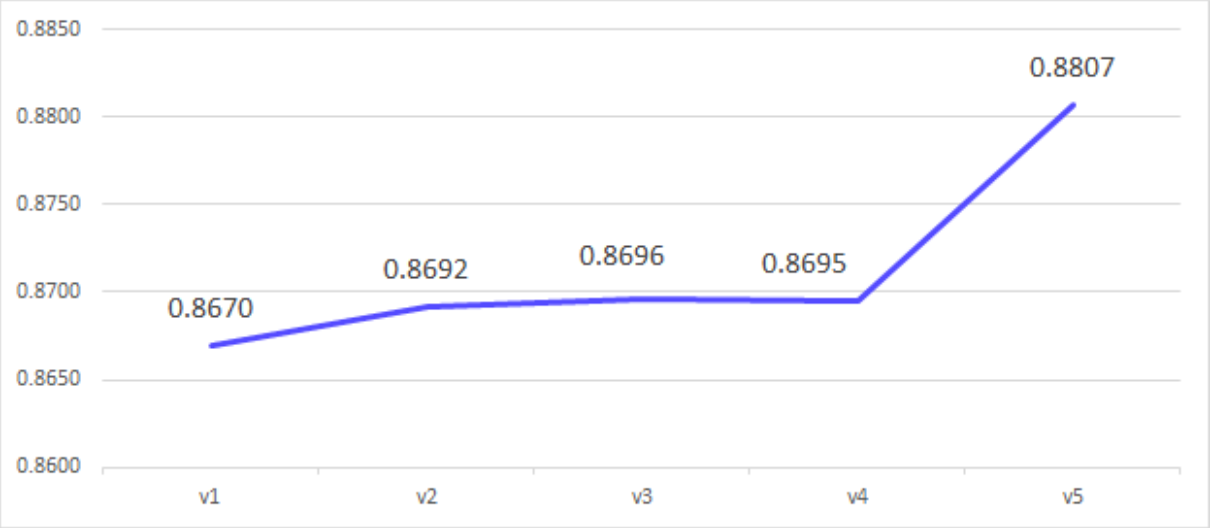
4.2 BitFit - Partial Fine/-tuning

For the second set of experiments, BitFit was employed in order to train the model.

BitFit optimizes just a subset of model parameters, and the method shows results that even surpass complete fine-tuning for small datasets (ZAKEN; RAVFOGEL; GOLDBERG, 2021), although they aim solely to be competitive while expending less computational resources through partially adjusting parameters.

It is important to have this in mind when looking at results for BitFit in Figure 19, as performance is expected to be competitive, but not necessarily better, when compared to complete fine-tuning.

Figure 19 – Complete Fine-tuning F1-Score



Source: Own Authorship (2022).

It is possible to observe that BitFit perfomance was also impacted in the fourth version of the dataset, although with less intensity than in full fine-tuning. This goes along with the fact that less parameters are being optimized, and thus the model is less affected by noise in the fine-tuning dataset, preserving more knowledge learned in pre-training.

BitFit authors mention that results obtained corroborates the theory which claims that fine-tuning exposes the model to knowledge induced by pre-training, essentially using pre-trained linguistic features to understand the new task, which is reinforced by a lower performance degradation in version 4.

This analysis is specially important as it also hints at the possibility of BitFit being more robust to Catastrophic Forgetting, while also never creating the problem of Catastrophic Remembering.

Catastrophic Forgetting is a well known problem in fine-tuning (and Continual Learning in general), where a model forgets knowledge acquired during pre-training during the process of learning a new task (fine-tuning) (KAUSHIK *et al.*, 2021). Some authors proposed data replay method in order to deal with this problem, where you expose the model again to older tasks data, and although this deals with Catastrophic Forgetting, models have a penalty in the ability to discriminate between new and old tasks (SHARKEY; SHARKEY, 1995). If BitFit is able to overcome catastrophic forgetting, it solves it without ever facing catastrophic remembering.

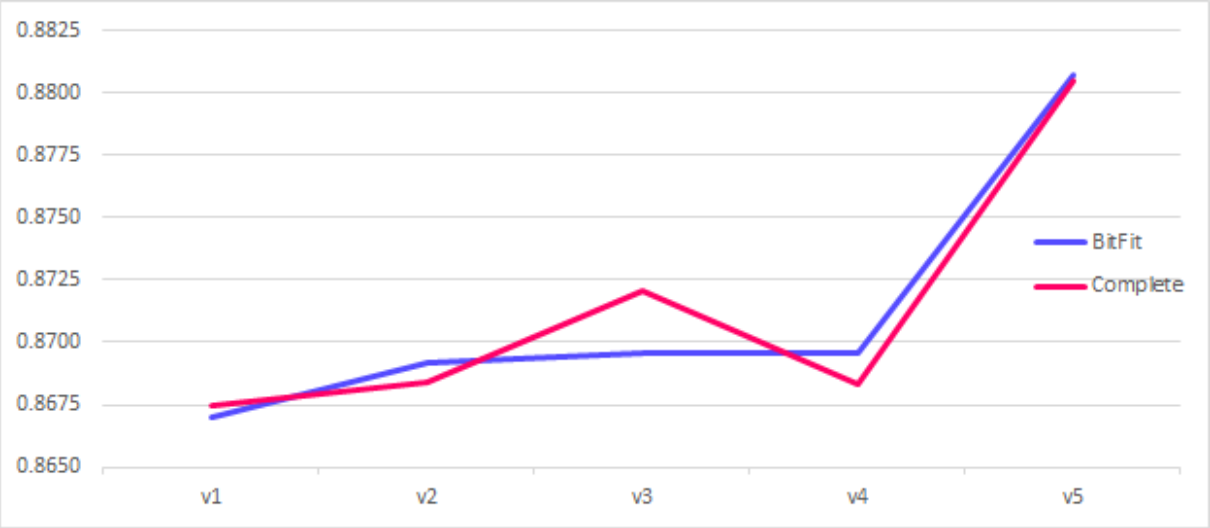
Keeping 95% of model parameters unchanged further supports BitFit avoiding Catastrophic Forgetting. This is because more knowledge acquired from pre-training is retained in comparison to complete fine-tuning, although feature representations usually aren't independent along layers inside deep learning models, as layer n usually depends on information coming from layer $n-1$, being skip connections and similar methodologies some exceptions. Thus, empirical evidence indicates BitFit method robustness to Catastrophic Forgetting and problems alike, although it is not trivial to formally prove it.

4.3 Results comparison

Following what was expected from the BitFit paper, it performed, on average, better for smaller dataset versions, although it performed worse in early (smaller) dataset versions and slightly better than complete fine-tuning on later (bigger) versions.

Results discussed can be visually compared in Figure 20.

Figure 20 – BitFit Fine-tuning F1-Score



Source: Own Authorship (2022).

As already stated, BitFit outperforming complete fine-tuning in dataset version 4 indicates that it is more resistant to fine-tuning data noise, and corroborates that fine-tuning is about leveraging pre-trained knowledge to solve fine-tuning tasks.

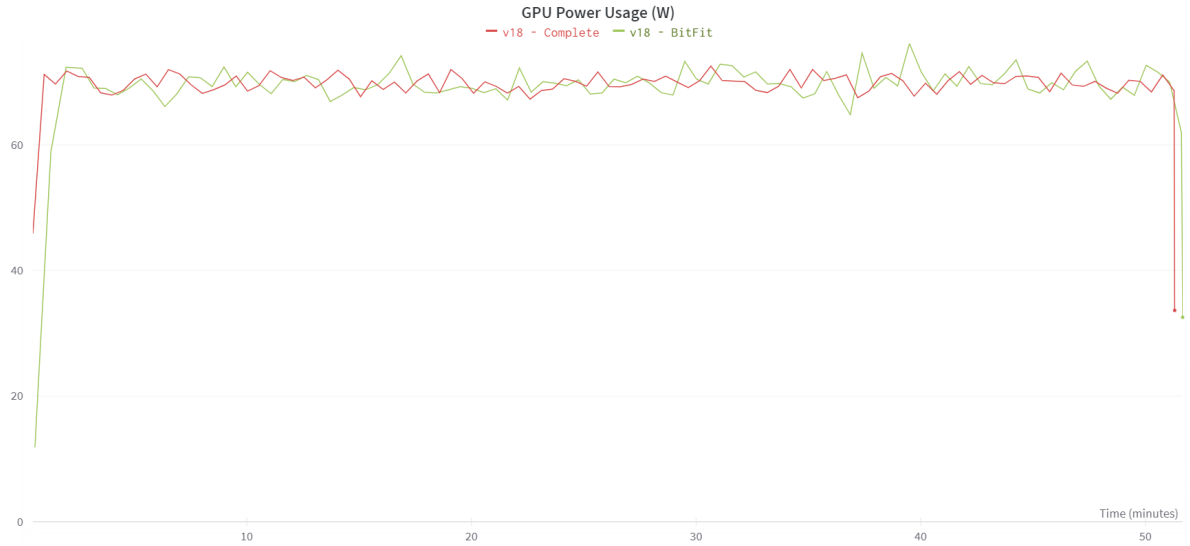
It is possible to lookup results for every single experiment in Table 1. Precision and recall are strongly correlated with the f1-score, as it is their harmonic mean. BitFit optimized model seems to keep the good precisions X recall tradeoff shown by the completely fine-tuned model, as they don't diverge much from the f1-score. This is ideal as it isn't inflating one over the other, which would cause a low f1-score or a non-reliable model, as it would either 1) commit to many false negatives (high precision, low recall) or 2) commit to many false positives (low precision, high recall).

Also, the correlation between f1-score and dataset size was calculated. F1-score is 82% correlated with size for Complete Fine-tuning and 88% correlated for BitFit fine-tuning. This reinforces the fact that BitFit usually performs better on smaller datasets.

In terms of energy consumption, BitFit doesn't seen to have major gains when compared to other fine-tuning strategies as backpropagation has a dependency on the gradient calculation of previous (or following, as it is a backward pass) parameters, thus requiring that they are calculated even though they aren't being updated (NIELSEN, 2018).

Energy consumption results for complete and BitFit fine-tuning experiments with seed 42 in dataset version 5 can be observed in Figure 21.

Figure 21 – GPU Energy Consumption



Source: Own Authorship (2022).

BitFit, then, slightly improves model performance, specially in cases of fine-tuning data noise. Also, it sometimes (3 out of 5, considering average f1-score) can be better than complete fine-tuning for small datasets.

Table 1 – All Experiments Results

fine-tune	dataset version	seed	precision	recall	f1
Full Fine-tuning	14	40	0.8669	0.8673	0.8669
Full Fine-tuning	14	41	0.8679	0.8667	0.8667
Full Fine-tuning	14	42	0.8692	0.8693	0.8689
Full Fine-tuning	15	40	0.8706	0.8723	0.8704
Full Fine-tuning	15	41	0.8666	0.8679	0.8663
Full Fine-tuning	15	42	0.8683	0.8696	0.8685
Full Fine-tuning	16	40	0.8699	0.8707	0.87
Full Fine-tuning	16	41	0.8701	0.8702	0.8698
Full Fine-tuning	16	42	0.877	0.8772	0.8765
Full Fine-tuning	17	40	0.8646	0.8647	0.8644
Full Fine-tuning	17	41	0.8723	0.8734	0.8726
Full Fine-tuning	17	42	0.8677	0.8691	0.8679
Full Fine-tuning	18	40	0.8811	0.8813	0.881
Full Fine-tuning	18	41	0.8787	0.8792	0.8787
Full Fine-tuning	18	42	0.8819	0.8819	0.8818
BitFit	14	40	0.8694	0.8689	0.8688
BitFit	14	41	0.8668	0.8673	0.8665
BitFit	14	42	0.8659	0.8658	0.8656
BitFit	15	40	0.8685	0.8695	0.8685
BitFit	15	41	0.8689	0.87	0.8688
BitFit	15	42	0.8703	0.8717	0.8703
BitFit	16	40	0.87	0.8704	0.8699
BitFit	16	41	0.8691	0.8702	0.8692
BitFit	16	42	0.8712	0.8699	0.8696
BitFit	17	40	0.8696	0.8696	0.8691
BitFit	17	41	0.8716	0.8725	0.8716
BitFit	17	42	0.8679	0.8686	0.8679
BitFit	18	40	0.8789	0.8793	0.8789
BitFit	18	41	0.8817	0.8815	0.8814
BitFit	18	42	0.8818	0.8822	0.8818

Source: Own Authourship (2022).

The Complete fine-tuned model was also validated by the Legal community, as it was presented by this research contractor's company to the Federal Court of Audits (Tribunal de Contas da União, from Portuguese), and they were interested in following this project researches. It is possible that that will become one of this model users, along with a large brazillian bank which name cannot be disclosed.

5 CONCLUSION

This work resulted from the efforts to classify legal documents in order to serve a big demand of text analysis, which would need many hours of work to be done by humans. More specifically, this work is an attempt to improve results obtained with complete fine-tuning, considering the smaller-than-usual Portuguese dataset for this work fine-tuning purposes.

For thus, a PELT method was leveraged, namely BitFit. It was used in order to achieve better performance in the dataset built in parallel by another team in the project research group. PELT methods optimize just a subset of model parameters in order to more efficiently store model parameters for each task. BitFit, for example, optimizes 0.08% of total model parameters and keeps 99.02% untouched. Although only a subset of parameters is optimized, BitFit energy consumption is the same as complete fine-tuning, as backpropagation must calculate the partial derivatives for all parameters, and not only those being optimized.

Although this property is useful for deploying LLMs, for the purpose of this work, the most important BitFit feature is that it surpassed complete fine-tuning results in some cases, for small datasets. This paves road to experiment with it in order to improve complete fine-tuning performance in small datasets.

Experiments results went accordingly to BitFit paper results, maintaining competitive results and even surpassing complete fine-tuning for a small dataset. Dataset size was more correlated to BitFit average metrics than with full fine-tuning, as well.

An unexpected relation came up during experiment runs: BitFit is much more resistant to fine-tuning data noise than complete fine-tuning. This went further, and it is perhaps capable of mitigating effects of Catastrophic Learning. That is a hypothesis that may be worth further investigation, in future work.

BitFit is surprisingly good at optimizing a very small subset of model parameters and achieving great performance. Applying it to achieve better performance allowed this work to more effectively classify legal documents given annotated data constraints, as well as validating this strategy to further improve model performance when only smaller datasets are available. If any company is going to annotate its own data, models trained on early dataset versions might usually benefit from using BitFit, earning the company some performance points.

Other PELT methods exist, which are not as trivial as BitFit, which achieves very good results given its simplicity. Some of them have solid results, and some preliminary experiments using unipelt (MAO *et al.*, 2021) were done during this work, although they weren't promising. In further work, it might be worth to give a look at other PELT strategies.

It is important to disclose that the company sponsoring this work team presented this model to the Federal Court of Audits, earning their attention to this project, and thus the legal community initial approval. Also, lawyers and engineers working at the company sponsoring this work are already discussing deployment and integration into their automatic legal document generation product.

The project goals were achieved. With just a very small subset of parameters (0.08%), BitFit achieved results not only competitive, but which surpassed those of fine-tuning. Also, it has shown to be robust to fine-tuning data noise, which makes it a great resort for future dataset versions that might present problems.

Lastly, for future works, the first suggestion is to further improve model performance with other pelt methods, such as those cited by and proposed on Mao *et al.* (2021), and the second is to formalize and further investigate whether BitFit solves catastrophic forgetting or not.

BIBLIOGRAPHY

ABADI, M. *et al.* Tensorflow: A system for large-scale machine learning. *In*: **12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)**. [*S.l.*: *s.n.*], 2016. p. 265–283.

ABONIZIO, H. Q. *et al.* Language-independent fake news detection: English, portuguese, and spanish mutual features. **Future Internet**, MDPI, v. 12, n. 5, p. 87, 2020.

ARONOFF, M.; FUDEMAN, K. **What is morphology?** [*S.l.*]: John Wiley & Sons, 2011. v. 8.

BAHDANAU, D.; CHO, K.; BENGIO, Y. **Neural Machine Translation by Jointly Learning to Align and Translate**. 2016.

BIEWALD, L. **Experiment Tracking with Weights and Biases**. 2020. Software available from wandb.com. Disponível em: https://www.wandb.com/.

BROWN, T. *et al.* Language models are few-shot learners. **Advances in neural information processing systems**, v. 33, p. 1877–1901, 2020.

CHEN, H. *et al.* A comparative study of automated legal text classification using random forests and deep learning. **Information Processing & Management**, Elsevier, v. 59, n. 2, p. 102798, 2022.

DEVLIN, J. *et al.* **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. 2019.

DOSOVITSKIY, A. *et al.* **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale**. 2021.

DU, Y. *et al.* Pp-ocr: A practical ultra lightweight ocr system. **arXiv preprint arXiv:2009.09941**, 2020.

FALOTICO, R.; QUATTO, P. Fleiss' kappa statistic without paradoxes. **Quality & Quantity**, Springer, v. 49, n. 2, p. 463–470, 2015.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S.l.*]: MIT Press, 2016. http://www.deeplearningbook.org.

GOUTTE, C.; GAUSSIER, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. *In*: SPRINGER. **European conference on information retrieval**. [*S.l.*], 2005. p. 345–359.

GOUWS, R. Aspects of lexical semantics. **Solving Language Problems: From General to Applied Linguistics**, University of Exeter Press, v. 20, p. 98, 1996.

HENDRYCKS, D.; GIMPEL, K. **Gaussian Error Linear Units (GELUs)**. 2020.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural computation**, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HOULSBY, N. *et al.* Parameter-efficient transfer learning for nlp. *In*: PMLR. **International Conference on Machine Learning**. [*S.l.*], 2019. p. 2790–2799.

HOWARD, J.; RUDER, S. Universal language model fine-tuning for text classification. **arXiv preprint arXiv:1801.06146**, 2018.

HU, E. J. *et al.* Lora: Low-rank adaptation of large language models. **arXiv preprint arXiv:2106.09685**, 2021.

HYNDMAN, R.; ATHANASOPOULOS, G. **Forecasting: Principlesnbsp;andnbsp;Practice (3rd ed)**. 2021. Disponível em: https://otexts.com/fpp3/.

INC., P. T. **Collaborative data science**. Montreal, QC: Plotly Technologies Inc., 2015. Disponível em: https://plot.ly.

JURAFSKY, J. H. M. D. **Speech and Language Processing**. 3th. ed. [*s.n.*], 2020. Disponível em: http://web.stanford.edu/~jurafsky/slp3/.

KADHIM, A. I. Survey on supervised machine learning techniques for automatic text classification. **Artificial Intelligence Review**, Springer, v. 52, n. 1, p. 273–292, 2019.

KAUSHIK, P. *et al.* Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping. **arXiv preprint arXiv:2102.11343**, 2021.

LEE, J.; TANG, R.; LIN, J. What would elsa do? freezing layers during transformer fine-tuning. **arXiv preprint arXiv:1911.03090**, 2019.

LI, X. L.; LIANG, P. Prefix-tuning: Optimizing continuous prompts for generation. **arXiv preprint arXiv:2101.00190**, 2021.

LIU, Y. *et al.* **RoBERTa: A Robustly Optimized BERT Pretraining Approach**. 2019.

LUONG, M.-T.; PHAM, H.; MANNING, C. D. **Effective Approaches to Attention-based Neural Machine Translation**. 2015.

MAO, Y. *et al.* Unipelt: A unified framework for parameter-efficient language model tuning. **arXiv preprint arXiv:2110.07577**, 2021.

MIKOLOV, T. *et al.* **Efficient Estimation of Word Representations in Vector Space**. 2013.

MIKOLOV, T. *et al.* **Distributed Representations of Words and Phrases and their Compositionality**. 2013.

MIKOLOV, T.; YIH, W. tau; ZWEIG, G. Linguistic regularities in continuous space word representations. *In*: **Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies**. [*S.l.*: *s.n.*], 2013. p. 746–751.

NIELSEN, M. A. misc. **Neural Networks and Deep Learning**. Determination Press, 2018. Disponível em: http://neuralnetworksanddeeplearning.com/.

NOGUTI, M. Y.; VELLASQUES, E.; OLIVEIRA, L. S. Legal document classification: An application to law area prediction of petitions to public prosecution service. *In*: IEEE. **2020 International Joint Conference on Neural Networks (IJCNN)**. [*S.l.*], 2020. p. 1–8.

OH, K.-S.; JUNG, K. Gpu implementation of neural networks. **Pattern Recognition**, Elsevier, v. 37, n. 6, p. 1311–1314, 2004.

OLIPHANT, T. E. Python for scientific computing. **Computing in science & engineering**, IEEE, v. 9, n. 3, p. 10–20, 2007.

OLTHOF, A. W.; OOIJEN, P. M. van; CORNELISSEN, L. J. Deep learning-based natural language processing in radiology: The impact of report complexity, disease prevalence, dataset size, and algorithm type on model performance. **Journal of medical systems**, Springer, v. 45, n. 10, p. 1–16, 2021.

Papers With Code. **Github Data - Framework Trends**. 2021. Disponível em: https://paperswithcode.com/trends. Acesso em: 02 ago. 2021.

PARADIS, C. Lexical semantics. **The encyclopedia of applied linguistics**, Wiley-Blackwell Oxford, p. 3357–3356, 2012.

PASZKE, A. *et al.* Pytorch: An imperative style, high-performance deep learning library. **Advances in neural information processing systems**, v. 32, p. 8026–8037, 2019.

PETERS, M. E. *et al.* **Deep contextualized word representations**. 2018.

PFEIFFER, J. *et al.* Adapterfusion: Non-destructive task composition for transfer learning. **arXiv preprint arXiv:2005.00247**, 2020.

RADFORD, A. *et al.* Language models are unsupervised multitask learners. **OpenAI log**, v. 1.

RAFFEL, C. *et al.* Exploring the limits of transfer learning with a unified text-to-text transformer. **J. Mach. Learn. Res.**, v. 21, n. 140, p. 1–67, 2020.

SANTHANAM, S. Context based text-generation using lstm networks. **arXiv preprint arXiv:2005.00048**, 2020.

SCHUSTER, M.; PALIWAL, K. K. Bidirectional recurrent neural networks. **IEEE transactions on Signal Processing**, Ieee, v. 45, n. 11, p. 2673–2681, 1997.

SHARKEY, N. E.; SHARKEY, A. J. An analysis of catastrophic interference. **Connection Science**, Taylor & Francis, 1995.

SMITH, R. An overview of the tesseract ocr engine. *In*: IEEE. **Ninth international conference on document analysis and recognition (ICDAR 2007)**. [*S.l.*], 2007. v. 2, p. 629–633.

SMITH, S. *et al.* Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. **arXiv preprint arXiv:2201.11990**, 2022.

SOUZA, F.; NOGUEIRA, R.; LOTUFO, R. Bertimbau: Pretrained bert models for brazilian portuguese. *In*: SPRINGER. **Brazilian Conference on Intelligent Systems**. [*S.l.*], 2020. p. 403–417.

SUN, Y. *et al.* **ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation**. 2021.

TRIGGS, A. *et al.* Dive into deep learning. **arXiv preprint arXiv:2106.11342**, 2021.

VASWANI, A. *et al.* **Attention Is All You Need**. 2017.

WANG, J. *et al.* Dimensional sentiment analysis using a regional cnn-lstm model. *In*: **Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)**. [*S.l.*: *s.n.*], 2016. p. 225–230.

WARRENS, M. J. Five ways to look at cohen's kappa. **Journal of Psychology & Psychotherapy**, OMICS Publishing Group, v. 5, n. 4, p. 1, 2015.

WOLF, T. *et al.* Huggingface's transformers: State-of-the-art natural language processing. **arXiv preprint arXiv:1910.03771**, 2019.

WOLF, T. *et al.* Transformers: State-of-the-art natural language processing. *In*: **Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations**. [*S.l.*: *s.n.*], 2020. p. 38–45.

YANG, Z. *et al.* Hierarchical attention networks for document classification. *In*: **Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies**. [*S.l.*: *s.n.*], 2016. p. 1480–1489.

ZAKEN, E. B.; RAVFOGEL, S.; GOLDBERG, Y. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. **arXiv preprint arXiv:2106.10199**, 2021.