BABEŞ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION APPLIED COMPUTATIONAL INTELLIGENCE

DISSERTATION THESIS

COVID-19 fake news detection using deep learning

Supervisor
Lect. Dr. Lupşa Dana

Author
NASTE Teodora Ioana

2021

UNIVERSITATEA BABEŞ-BOLYAI CLUJ-NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

LUCRARE DE DISERTAŢIE

Detectarea ştirilor false în context COVID-19 folosind deep learning

Conducător ştiinţific
Lect. Dr. Lupşa Dana

Absolvent
NASTE Teodora Ioana

2021

ABSTRACT

Society has been heavily impacted by the effects of fake news, especially considering recent world events, like the COVID-19 pandemic. While a first step would be the education of the population in this matter and aiding the user with manual fact-checking already done by experts from various organisations, the volume and speed at which it spreads makes it so that it is impossible to keep up. This is why research into automatic fake news detection has been of a great interest these past years. In this paper, I approach the fake news detection on a hand crafted dataset which contains claims regarding Coronavirus, using rather simple deep learning methods and similarity. Moreover, I present the CoviralCheck application which is meant for research in the field, as well as actual fact-checking using the models previously trained by the researchers.

Contents

Chapter 1

Introduction

Fake news have become a real issue in modern society when mostly everything has moved online where information is spread faster and cheaper than traditional news media such as newspapers and television. There have also been studies which have demonstrated that humans are only able to differentiate between legit and deceptive information with a mean accuracy around 55%-58% [35]. This was also proven by real-life events, like the COVID-19 pandemic, in which rumours of miraculous cures or potential non-existent threats have only managed to scare and confuse people, inducing chaos in a time that was already unstable and uncertain. For example, in Europe, patients refused to take ibuprofen after some viral misinformation claimed it worsened the COVID-19 symptoms.[23] Of course, the fields in which fake news circulate are not reduced only to health conspiracies, but also in politics, economics and entertainment, so I believe it should come as a reflex to study this phenomena and try to combat it.

As one can tell, this is not an easy task and one of the issues is deciding what exactly are fake news. The term can be defined as information that it's not true and it's spread with a malicious intention. Considering this, some key words that are associated with fake news are: deception, rumours, clickbait or even satires and memes. Minyi Zhou and Reza Zafarani gather substantial information in their survey [35] on fake news and manage to discover 3 characteristics that unites these concepts: authenticity, intention and whether the information is news. It is important to mention that even if a piece of information is not necessarily considered actually news that does not mean that it cannot manipulate the public's opinion and that it doesn't need fact-checking as much as any other piece of news. On the contrary, the difference is just in the method in which fact-checking is done, because different types of text have different features.

With this motivation, I present the CoviralCheck application which is an application to aid debunking fake news. It has 2 main user cases: first one designed for researchers that want to process data and train models for fake news detection and

the second one is designed for everybody who wants to fact-check a claim. Moreover, I gather my own data from the web and manual work. On this data, I train 3 models (multilayer perceptron, multilayer perceptron with similarity and LSTM) using the CoviralCheck Application.

The paper is structured in 6 chapters as follows. The first chapter is the introduction. The second chapter presents detailed theoretical aspects of machine learning models, explaining the way they functionate and how they learn. In the third chapters, I present some approaches that have been used to tacke misinformation in different scenarios and domains. The forth chapter presents the CoviralCheck applications and its features, while the fifth presents the approach I attempted to solve the task; the generation method of the dataset is also detailed here. Finally, the last chapter contains observations for improvement and future reasearch.

Chapter 2

Machine Learning Models

Machine learning is a subdomain of artificial intelligence which is preoccupied to create programs or machines that are able to learn. But what does learning mean in this context? Generally, we can say a program is learning when it is able to modify its behaviour based on new information. In a similar way how people learn from experience to become better people, machine learning aims for machines to be able to optimize themselves for the purpose of reaching a goal,for example the goal of minimizing an error function. In this chapter, I will present the logic and way of working for some machine learning models that have been used for natural language processing and , especially, fake news detection. This will be only a theoretical presentation, as the practical analysis will be presented in the next chapters.

2.1 Supervised learning

Machine learning algorithms can be separated into two big categories: supervised and unsupervised. The difference between them, except the algorithms themselves, is the data they take as input. Supervised algorithms work on labelled data, so they have an idea of what the expected result should be in order to adjust themselves. In a general manner, for each input data, the result of the model is compared to the actual known result and it gets a feedback, similar to what experience does to humans, to do better in future.

Even with the feedback that supervised algorithms benefit of, there are cases in which the need of labelled data can be an issue, because it is not always available; especially when we need many samples to give our model to learn. One such example can even be detecting misinformation in covid related news. Everything happened so fast that there was no time to actually put together a well defined dataset, as of the time this paper was written.

There are many supervised algorithms used in machine learning, some examples

are: linear regression, decision trees, artificial neural networks etc.

2.2 Artificial neural networks

The idea of a neural networks has first been introduced in 1943 by McCulloch and Pitts who have introduced the concept of a neuron as a conceptual unit capable of simple operations. However, as it is mentioned by Kröse, Ben et al. [**?**] , only in the 80s the interest towards artificial neural networks has begun to grow.

As the name suggests, the model of the artificial neural networks (ANN) is inspired by the structure of the human brain. It's formed by a big number of artificial neurons, also called units or nodes, united by edges. Each unit j has an associated activation function and each edge has a weight $\omega_{jj'}$, which represents the cost from unit *j* to the unit *j'*. So, the value of a node, v_j, is computed applying the activation function, like in the formula below:

$$v_j = l_j(\sum_{j'} \omega_{jj'} \cdot v_{j'})$$

The activation function l_j is a simple linear function. The most common used ones are the sigmoid function: $\sigma(z) = \frac{1}{(1+e)^{-z}}$ and the tanh function: $\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. These are especially used for feed-forward neural networks, but they have been applied to recurrent neural networks too. Another possible activation function is the rectified linear unit (ReLU) l_j(z)=max(0,z) which is mostly used for deep neural networks and for the task of image detection.

The architecture of the neural network can be observed in Figure 2.1 [12]. On the leftmost side we have the input layer and on the rightmost side we have the output layer. The layers in the middle are called hidden layers. A neural network has only one input / output layer, but it can have however many hidden layers.

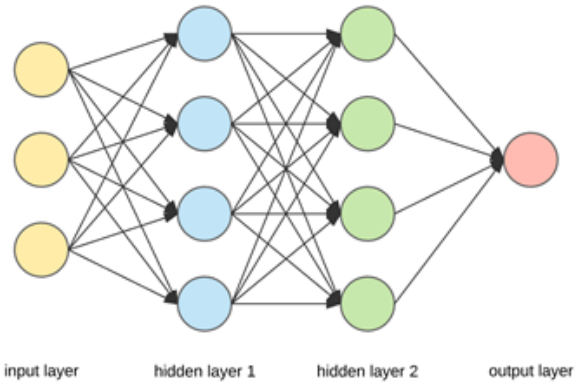

Figure 2.1: Neural network architecture

The artificial neural network is capable of approximating a non-linear function.

For example, we have a function $f(x) = y$ which maps a value x with a value y. A neural network constructs a mapping function $y = f^*(x,)$, where are the values of the parameters of the best approximation. The next question would be how the model chooses the best approximation and what exactly means best in this case. To answer, these networks learn using **error propagation**. The values calculated by the nodes are sent to the output nodes which compare them with the expected results. Each output node then returns the difference between the two , which is the error. The purpose is to minimize this error, so the best approximation is the one in which the error is at a minimum. The simplest method is to distribute the error from one output node to all the other nodes connected with it , proportionally with the nodes' weights, till the input nodes and each edge will update its weight and a new error will be calculated. The error is calculated using a **cost function**. For that a smooth function like the quadratic cost or mean square error is used to easily determined the small adjustments of the weights to achieve the best approximation.

Some of the particular properties of artificial networks are their ability to to learn and adapt, to generalize a problem or to organize data. However, there are also some limitations to this model. Firstly, because we are talking about a supervised model, neural networks need a set of meaningful examples. Also, it can be hard to determinate how big the network should be, especially since for bigger networks (with many more layers) the training time can grow very much. Another thing that should be considered is that the training depends a lot on the processing power of the computer.

2.2.1 Multilayer Perceptron

The perceptron is an early type of neural network for binary classification with no hidden layers that consists of only one neuron. The concept was first proposed by Frank Rosenblatt [27] in 1958, based on the neuron proposed by McCulloch and Pitts. However, this type of neural network can only learn linear functions.

The multilayer perceptron is a special type of what was described earlier as a neural network. It uses multiple neurons and so, it has multiple (hidden) layers. The difference is that in an artificial neural network, the neurons from a layer are not necesarily connected to all neurons from the next or previous layer, while for amultilayer perceptrion the hidden layers are fully connected. Figure 2.2 better exemplifies the difference between the two types of neural networks.

A multilayer perceptron may also be defined as having the same number of neurons on the hidden layers or the same activation function across the layers. Also, they can refferenced as deep neural networks or feed-forward neural networks. The termn feed-forward comes from the fact that the information flow from inone way,

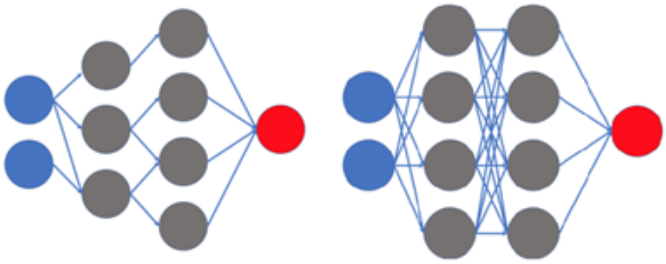

Figure 2.2: Left: graph representation of an artificial neural network; Right: graph representation of a multilayer perceptron

from the input layer to the output layer (from left to right). There are neural networks in which information can be exchanged between the neurons of the same hidden layer too. These will be discussed in the next section.

2.2.2 Recurrent neural networks

In contrast to traditional neural networks, the recurrent ones can have cycles, introducing the notion of time to the model. Thus, at a time t , the nodes with a recurrent edge get information from the current input x_t and from the previous states $h_{(t-1)}$. A simple recurrent neural network (RNN) is presented in Figure 2.3. [19] Moreover, RNNs are able to handle variable-length input, making them suitable candidates for processing sequences like texts.

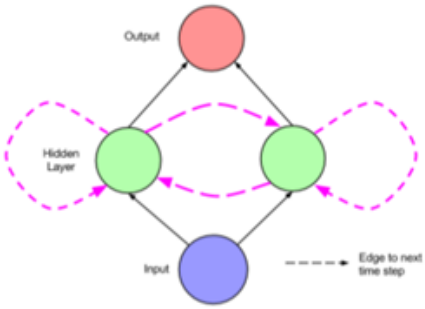

Figure 2.3: Recurrent neural network

The training for a RNN can be difficult. The main problems that can appear are vanishing and exploding gradients which can occur during error propagation. Which of these two events may occur depends on the weight of the edges and on the activation function. For example, if the activation function is a sigmoid one, the values risk to get too close to 0 and the network won't learn. In this case, we deal with vanishing gradient. If the activation function is a ReLU max(1,x), then it's easy to imagine the values growing too high and the exploding gradient might appear.

2.2.3 Long Short-Term Memory Neural Networks

To solve the problems of the vanishing and exploding gradients that appear during training of a standard RNN, the Long Short-Term Memory neural network was proposed in 1997 [16] with an innovative gating system that introduces an intermediate memory space via the memory cell.

The gating system consists of 3 gates: the input, forget and output gates. The input gate, i^t, decides how much of the information processed by the input node, g^t, should be modified. The forget gate, f^t, helps to sort out the irrelevant information and the internal state, s^t , is calculated based on the results from the previous 2 gates. Then, the final activation is composed with the help of the output gate, o^t, which decides how much of the information should be passed to the next units. This process can be viewed in the next equations [19]:

$$g^t = \phi(W^q x^t + U^g h^{t-1} + b_g)$$

$$i^t = \sigma(W^i x^t + U^i h^{t-1} + b_i)$$

$$f^t = \sigma(W^f x^t + U^f h^{t-1} + b_f)$$

$$o^t = \sigma(W^o x^t + U^o h^{t-1} + b_o)$$

$$s^t = g^t \bigodot i^t + s^{t-1} \bigodot f^t$$

$$h^t = \phi(s^t) \bigodot o^t$$

Firstly, the hidden state computed by the previous nodes, $h^{(t-1)}$, is concatenated with the input x^t , then the result is passed through the forget gate, which uses a sigmod activation to *"forget"* information that's irrelevant for the current task. Next, the values go throught he input gate which also sues a sigmoid activation. At the same time, the candidate value is composed using a tanh activation. The candidates are then concatenated with results from the input gate. The next step is computing the internal state s^t and the hidden statte that will be forwared to the rest of the nodes h^t. This last step is done by the output gate. The internal state is normalized using a sigmoid function then multiplied with the candidate values o^t. The whole flow can be also seen in Figure 2.4 [19], which presents the structure of a LSTM neural network with its gates and activations.

LSTM neural networks have been successfully applied in many different domains and are currently the most popular recurrent neural networks, even if they were firstly proposed over 20 years ago. Some of the fields in which they have been used are: text classification, text generation, automated translation, image captioning or predictions.

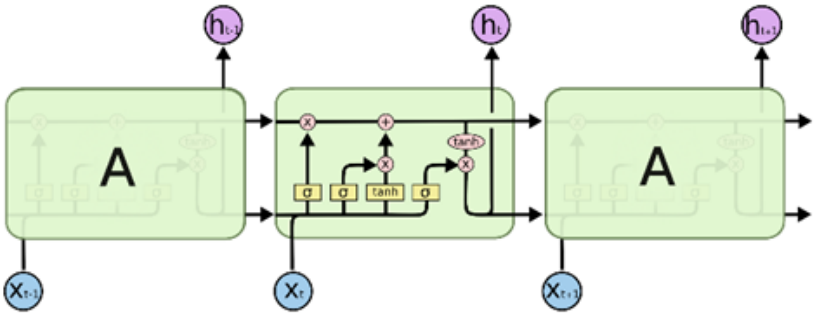

Figure 2.4: Structure of a LSTM neural network

2.3 Regularization

As mentioned in the previous chapters, recurrent neural networks are prone to overfitting. This is a common problem when training a neural network, however there are methods that help a model to avoid overfitting.

Firstly, what is overfitting? During training, it can happen that the model shows very good accuracy on the training data, but a poor one on the test data. This is because the model learnt too well the details in the edge cases from the training data. This can often happen especially when a model is too complex. Figure **??** [29] shows an example of how overfitting looks compared to underfitting. It can be observed how the model fails to catch the nuances of the training data for underfitting and this leads to poor training accuracy. Comparatively, the mapping representation touches the edge points , creating a very specific curve. Because this curve is so specific to the training data, the model will fail on test data.

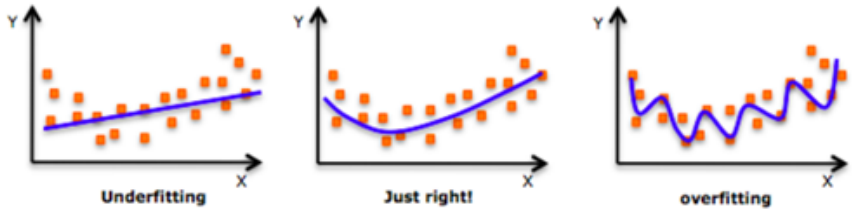

Figure 2.5: Graphic representation of underfitting and overfitting

While underfitting can be solved using a larger dataset or adjusting the parameters, overfitting can be avoided using regularization. Regularization is a technique that penalizes weights of the nodes in order to obtain small, but impactful changes for the model to generalize better.

There are different techniques used for regularization, but probably the most used ones are: L1 , L2 regularization and dropout.

L1 and L2 regularization are probably the most common ones. As their name suggests they are related to the L1 and L2 norms of a vector w. In this context, these

will be the regularization terms.

$$L1\,norm\,or\,1-norm: \|W\|_1 = |w_1| + |w_2| + ... + |w_N|$$

$$L2\,norm\,or\,Euclidean\,norm: \|W\|_2 = (|w_1|^2 + |w_2|^2 + ... + |w_N|^2)^{\frac{1}{2}}$$

The regularization terms are added when computing the cost function. Assuming y is the function we want to approximate, y' is the computed approximation and λ is the regularization parameter who is optimized for better results, the loss would look like this:

$$Loss = Error(y, y') + \lambda \sum_{i=1}^{N} |w_i|$$

$$Loss = Error(y, y') + \lambda \sum_{i=1}^{N} |w_i|^2$$

Adding the regularization terms to the loss, we force the weights to shrink to values close to 0 (even 0 in case of L1). This is done because it is assumed that neural networks with smaller weights lead to simpler models. Now, to explain why adding these terms help in dealing with overfitting. Let's assume that y' generated a perfect match for an edge case in our model. This leads to overfitting, but with the help of the regularization, the value is slightly shifted from the noise data. Also, we don't want perfect matches in our model, because, without regularization, this will lead to no real change in the weights and so the model won't be able to predict well on other datasets, so adding the regularization also helps with the generalization.

Dropout is a different type of regularization, but is also the one that produces the best results and is the most frequently used in deep learning, however still very simple and approachable. The idea behind this technique lays in the concept of ensembles. Ensembles are a group of models which can better cover different aspects of the data and so they can be used to reduce overfitting and enhance generalization. However, there is a cost in computational power and maintenance for multiple models. So, through dropout an ensemble can be simulated using only one single model. This is done by randomly dropping some nodes and their edges during each training iteration. So, with each iteration we can get a different neural network structure and ,in effect, each layer will update the weights differently for the nodes. In this way it adds more randomness to the model, encouraging a better generalization This technique is especially good when dealing with large and complex neural networks.

Dropout is implemented on each layer and it can be used on any types of layers (hidden, input/output). A hyperparameter is also given that indicates the proba-

bility of dropping, or inversely, keeping a node, also known as dropout rate. This probability is usually 0.5 for a hidden layer and somewhere close to 1 for a visible layer (input or output) [30]. Of course, because of the dropout , the weights would be larger than normal, so before finishing, they should be scaled down with the chose dropout rate.

2.4 Transformer models

Transformers are a quite recent addition to machine learning and especially to the field of natural language processing. This field has been dominated for quite a while by recurrent neural networks based models, like Long Short-Term Memory and Gated Recurrent Unit, because of their powerful ability to process sequences and remember important details. However, transformers offered a breakthrough , proving to surpass the old models for certain tasks like machine translation [21], without using any recurrent neural networks at all. The architecture of the transformer was first proposed by Vaswani Ashish et al. in 2017 [32] and the idea was based on 2 other concepts: the sequence-to-sequence models and the attention mechanism, so before going further into describing the transformer model, I'll first explain these concepts.

2.4.1 Sequence-to-sequence (Seq2Seq) model

A Seq2Seq model is actually a neural network that is given a sequence of elements, such as a sentence, transforms it into another sequence, as the name also suggests. These models are based on the Encoder-Decoder architectures such as the one proposed by Cho et al. [7] and Sutskever Ilya et al. [31]. Both of these papers focus on machine translation and express their motivation for such a system because the neural networks, even though are powerful tools in learning difficult tasks with large labeled data, they still lack in mapping a sequence to another sequence. These models are also achieving spectacular results for other tasks like image captioning, question answering etc. In fact, Google has started using a seq2seq model for translation since 2016. The architecture of the Seq2Seq can be observed in Figure 2.6 [22].

As mentioned, the model is formed by 2 parts: an encoder and a decoder. These are usually some type of a recurrent neural network, like GRU [7] or LSTM [31]. The encoder is given an input sequence and it translates it into a context vector. The purpose of the context vector is to encapsulate the important information that need to be then decoded to make accurate predictions. For this step the actual outputs of the encoder are ignored and only the context vector or internal states are considered. Then, the decoder gets the final state of the encoder and it translates into the desired

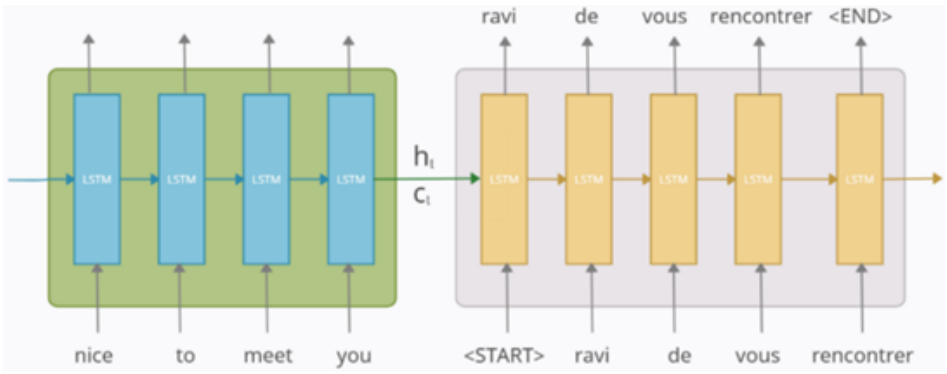

Figure 2.6: Architecture of the Encoder-Decoder model

sequence output. So, while the encoder is trained on the input sequence, the decoder is actually trained on the information received from the encoder. To better explain how the decoder works, we should look at the Figure 2.6, at the decoder side. The first input is a START symbol which is expected to generate the first word in the translation. Then the output is passed as the next input until an END symbol.

There are however disadvantages to a simple seq2seq model and these are mostly related to the fixed length of the context vector. Due to this, for long sentences, the context might forget important information from the beginning of the sentence till it reaches the end. Moreover, it was proven by Cho et al. [6] that the performance of the encoder-decoder models decreases rapidly as the length of an input sentence increases.

2.4.2 Attention mechanism

The attention mechanism was first introduced by Dzmitry Bahdanau, et al. [4] and it offers a way for a neural network to distinguish between what is relevant from the input and what is not. This is extremely useful because given a sentence, not all words from the sentence are actually important to the meaning, but in fact it can be reduced to a few important words. Intuitively, if we take as an example the task of automatic translation, the Seq2Seq explained in 2.4.1 would first memorize the whole sentence and then give it to the decoder for translation. A human translator, however, would probably look at the first part of the sentence translate the words and move to a different part. The attention mechanism is looking to simulate a human translator.

First, attention was added as an extension of the seq2seq models. What was happening is that on the encoder part some annotations or attention weights were calculated for both the preceding and following inputs.Then the context vectors for each input is computed as the weighted sum of the annotations. The decoder gets the context vectors which will contain information about the whole sentence, but

will have a strong focus on the parts surrounding the current input. So,for each input the decoder will decide to what parts of the sentence to pay attention to when making the prediction.

2.4.3 Transformers

Similar to the seq2seq model, transformers are model that transform one sequence into another sequence using two parts, an encoder and a decoder, plus, the most important component, attention. Actually, the difference between the previously encoder-decoder and the transformer is that the later has dropped the recurrent neural networks from the model and instead focuses only on attention, as the creative title of the initial paper suggests *"Attention is all you need"* [32].

The architecture of the Transformer can be observed in Figure 2.7 [32]. We can see that the encoder and decoder are composed by modules that can be stacked and each module has 2 or 3 sub-layers. The encoder's modules are formed by a multi-head attention layer and a position-wise fully connected feed-forward network layer. The decoder has one extra multi-head attention layer which makes sure that the predictions respect the sequence order and that one output depends only on the known previous outputs. This was not needed before because recurrent neural networks are made to be able to process sequences and to remember the order of the elements, otherwise we wouldn't have a sequence anymore. However, feed-forward neural networks don't know how to keep order,so there is a need to encode a relative position of the elements, hence the "position-wise" in the name.

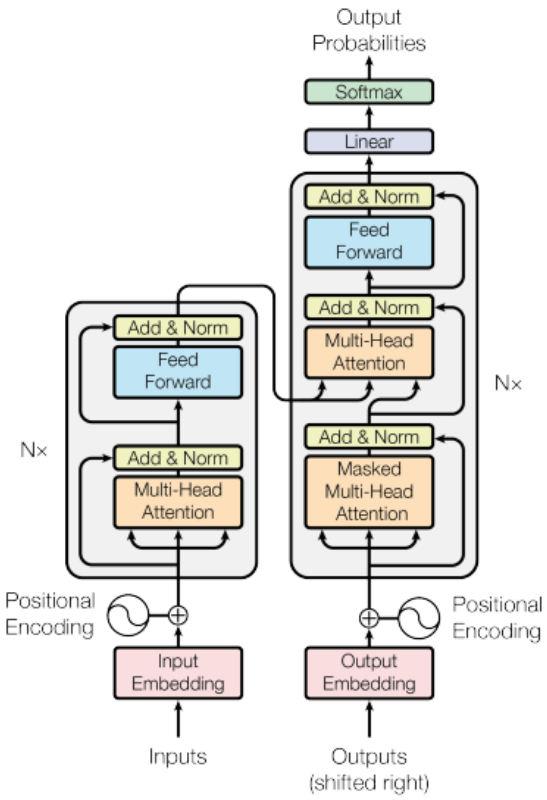

Figure 2.7: Architecture of the Transformer model

The attention is the most important part of the model. In the paper, the authors describe 2 types of attention: Scaled Dot-Product Attention and Multi-Head Attention. The first one computes the attention weights based on the dot product of the

"query" and "keys". In this case, the query is each word of the sentence and the keys are all the other words. So, in the end each word will be influenced by all the other words. The later attention is a paralleled version of the attention mechanism. Using this version allows much faster training than previous models.

Based on the general idea of the transformer model, several other models have been implemented. The most popular and well-known out of them would be BERT (Bidirectional Encoder Representations from Transformers) [10], which achieved state of the art results in many natural language processing tasks.

Chapter 3

Approaches to fake news detection

The raise in popularity of social media has massively increased the amount of user-generated information that can spread uncontrollably throughout the web. This heavy amount of data generated in real-time is impossible to be filtered and checked manually for veracity. So, there has been a more interest to research automatic ways of detecting false information. However, considering the complexity of what can be considered fake news and the various forms in which this can be spread in, the task of simply determining the authenticity of the information can be very hard and complex. In the last years, there has been a massive interest to combat the spread of fake news, especially those caused during the COVID-19 pandemic, which put people's health at risk. In this chapter I will present some of the methods used to approach the task and sub tasks, not just in the context of the pandemic, but also other topics.

3.1 Deep learning approaches

Even if the task of fake news is rather new, there have been attempts to tackle it before. In 2017 the Fake News Challenge released acontest for stance detection. Given two pieces of text, the purpose of the task was to estimate the relative perspective or stance of a text to the other on a given topic, claim or issue. The dataset for the challenge consisted in pairs of news article headline and its body and the contestants had to estimate the stance of the body towards the headline. The stance was represented by one of the following labels: *agree*, *disagree*, *discuss* or *unrelated*. There were many entries for this challenge, each with different types of models, however the ones that got into top 3 used rather simple methods. For example, Hanselowski Andreas et al. [15] and Riedel Benjamin et al. [26] have used simple multilayer perceptron models and were able to get the second and third best results.

The former used a MLP with one hidden layer and as features: term frequency

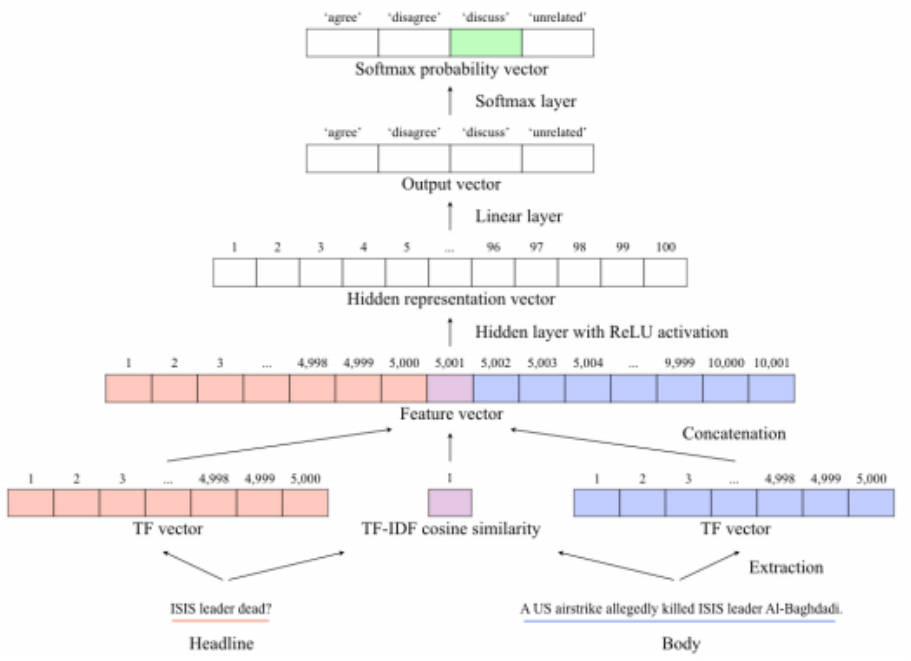

Figure 3.1: Schematic diagram of stance detection system by Riedel et al.

and similarity between headline and article body. With this setup they managed an overall accuracy of 88.46%. Figure 3.1 shows a schema of their approach. This paper has heavily inspired my research, the model used and the input structure used,but this will be discussed in detail in chapter 5. The other team used an ensemble of 5 MLP models,together with semantic features, paraphrase detection and non-negative matrix factorization. The final prediction was made with a voting system. The overall accuracy was of 89.5%. Other teams have used more complex models, but these two have caught my attention because of their simplicity and really strong results.

Ensembles have been used a few times to enhance the power of the models for such a difficult task. An ensemble helps to get higher accuracy as more models are trained and this introduces more randomness in the overall model, which helps generalization and error minimization. Ahmad et al. [2] have used an ensemble of different classifiers to label news as *True* or *False*. The chosen algorithms were: logistic regression, support vector machine, multilayer perceptron, K-Nearest Neighbors and decision trees. Using a tool, they extracted different textual features, such as words emotions implied by words, quantity of stop words or of different grammar parts. Their ensemble methods has shown an accuracy greater than 90% on all datasets used. The models that have stood out are random forests, the bagging ensemble with decision trees and the boosting ensemble with XGBoost algorithm.

Similar to ensembles, Ghosh and Shah [11] have used a modular approach to classify news as *Fake*, *Suspicious* and *Legit*. They have used several known fake news

detection datasets which contained news from various domains. A first module was using information retrieval to retrieve k related articles for a news statement and feed them to a feed forward neural network. The second module was collecting information about the writing style of the text which used a bidirectional LSTM as classifier. Finally, the results from the two modules pass through a voting system using a weighted average to get the final prediction. The model has achieved an accuracy of 82.4%.

Another sub field to fake news detection is satirical news detection. Satirical news bring another challenge because their purpose is for entertainment and thus, it is harder to recognize the intention of the creator. Also, the satire can be subtle and so, users could easily misunderstood it as true news. In this case, the style of writing makes a big difference. Fan Yang et al. [34] have investigated satirical news and proposed a paragraph-level detection model using a hierarchical neural network. This was composed of a convolutional neural network, a gated recurrent unit (GRU), a bidirectional GRU and ,finally, a MLP. They managed to obtain an accuracy of 98.39%. The importance of the linguistic features used was also analysed which gives a very interesting insight into what features are more relevant when trying to detect satire.

Social media is the main environment in which fake news are being created and spread at an alarming rate. However, this makes space to research different aspects of fake news, like the propagation pattern and the users' response via comments or reposts, with the intention to have an automatic method of spotting misinformation in its early stages of spreading. Ma et al. [20] have implemented a tree-structured recursive neural network that analyses how a rumour is spread and what's the users' stance relative to it. They used tweets and their propagation trees as the dataset and were classifiyng the tweets into the following categories: *non-rumour*, *false rumour*, *true rumour* and *unverified rumour*. They have used 2 RNN that were analyzing a tree from both directions (root to leaves and vice versa) to get a better insight into the complex patterns. They obtained an accuracy of 73% which was the best on the datasets at the time. Moreover, further experiments were done to see the capabilities to detect rumours in early stages. They concluded that their method needs approximately 8 hours or about 90 tweets to correctly spot a fake rumours which is superior to the best baseline method which needs 36 hours and about 300 posts.

3.2 Transformer based approaches

Transformers based models, like BERT, have shown great results in many complex NLP tasks and they have been used quite a lot for fake news detection too, especially lately.

Jwa Heejung et al. [18] have lately revised the stance detection task and proposed a new model called exBAKE, based on BERT. They have added new data (from CNN and Daily Mail) to the already pre-trained BERT, so that their model, BAKE , is also trained on fake news articles. Using it, the data was classified into the known labels for stance detection. Their revised BERT has managed a F1-score of 0.746, which is higher than previous BERT experiments.

Coming into the present crisis that has led to a scary spread of fake news, transformers have been used in many cases for COVID-19 fake detection, showing very good results. Vijjali et al. [33] have used a two stage transformer on data gathered from Poynter. They first formed a knowledge base using (claim,explanation) pairs that they generate using a transformer trained for entailment task. Then, the most relevant explanations are fetched and fed into the second transformer model which classifies if the given claim is true or false. It's interesting how they look at this problem as a textual entailment task. They evaluated 3 transformers models and they obtained an accuracy of 85.5% using a combination of BERT and ALBERT.

Another very recent research with BERT on COVID-19 data has been done by Chen et al. [5] that was published just earlier this year. This proves how relevant the topic is to this day and time. The researchers have improved BERT by training it on COVID-19 specific sentences. Then, they combined two versions of BERT: CT-BERT and RoBERTa. The results from the two transformers were then fused with a multilayer perceptron. Moreover, the BERT parameters have been highly optimized. In the end, they obtained a F1-score of 0.99 for a public COVID-19 fake news dataset.

For social media posts, Gundapu and Mamidi [14] have developed an ensemble of 3 transformer models: BERT, ALBERT and XLNet, that computes an average of the outputs from the transformers. They trained the ensemble on a dataset with various media posts from different platforms like Facebook, Twitter, Instagram etc. Overall, they obtained an accuracy of 98.55%.

3.3 Other approaches

There have been some approaches presented above,however the research does not stop here. There has been research using word-based N-gram analysis, information retrieval and linear support vector machine [3]. After pre-processing, the N-gram analysis was used to extract features. The model was trained on a dataset made from online articles and they have obtained an accuracy of 92%.

There has been research going on in our country too for fake news detection. One such paper was written by Cuşmaiul et al. [9] which have tried different machine learning models on a database from Twitter. The best results were obtained using a Random Forest with an accuracy of 95.93%. Ontologies and reasoning have been

also used for the task by Groza Adrian [13]. He has used description logics to find inconsistencies between trusted and not trusted medical sources. For this, a Covid-19 ontology of facts and myths has been designed. Then, for each input entry, a tool was used to convert it into an ontology. The last step was to look for conflicts the freshly converted ontology and the one previously mentioned with facts.

Chapter 4

CoviralCheck Application

4.1 Motivation

The spread of fake news has caused situations in which people's health has been put at risk. Today there are several fact-checking sites that have specialist who verify suspicious statements. However, the speed at which misinformation is spread and created surpasses the speed at which people can verify them. For this reason, finding an automated fake news detection solution is essential. In this scope, there has been extended research in the field.

The goal of my research is to find a way to implement an application that can aid scientists to further research ways to automatically detect fake news, as well as to offer the general public a trusted fact-checking that can verify new claims fast and in an accurate manner, with minimal need for human intervention.

4.2 Specifications and requirements

The main requirement for the application is to support two cases: to be used for research and for actual fact-checking. To satisfy this, it offers one interface for each case with different features.

The fact-checking interface has to offer the user the possibility to input a claim of their choice which they want to verify. By clicking a button, the claim will be processed by the chosen model that the application runs and the user will receive a response in the form of a True or False tag.

The research interface is designed for professional scientists and it has to offer the next features: possibility of preprocessing a given dataset, adjusting the hyperparameters and training a machine learning model, possibility to save a trained model and to view the results of the training. The user can input a csv file of their choice as the dataset. The file has to respect the structure defined in chapter 5.2. Then by

clicking a button, the application will apply the techniques defined and save the new dataset. For the training part, the user is required to choose from a dropdown a model that has been predefined. Then, they can choose to go with the default parameters or adjust them accordingly. After pressing a button, the application will start training the chosen model using the parameters given by the user. At the end of the training, the user can see the model name that was last trained and the accuracy and loss it obtained on the test data (the application automatically splits the given dataset 80/20). Next to the last results, the user can see a list of the best performing models, so the results can be compared. The list should update as more models are being trained.

4.3 Architecture and diagrams

CoviralCheck (Covid+Viral) is a desktop application which is implementing the Model View Controller (MVC) design pattern with adjustments to incorporate the machine learning part; that is in fact the main focus of this work. That's why not all object oriented programming paradigms have been necessarily been respected entirely. Figure 4.1 shows the class diagram.

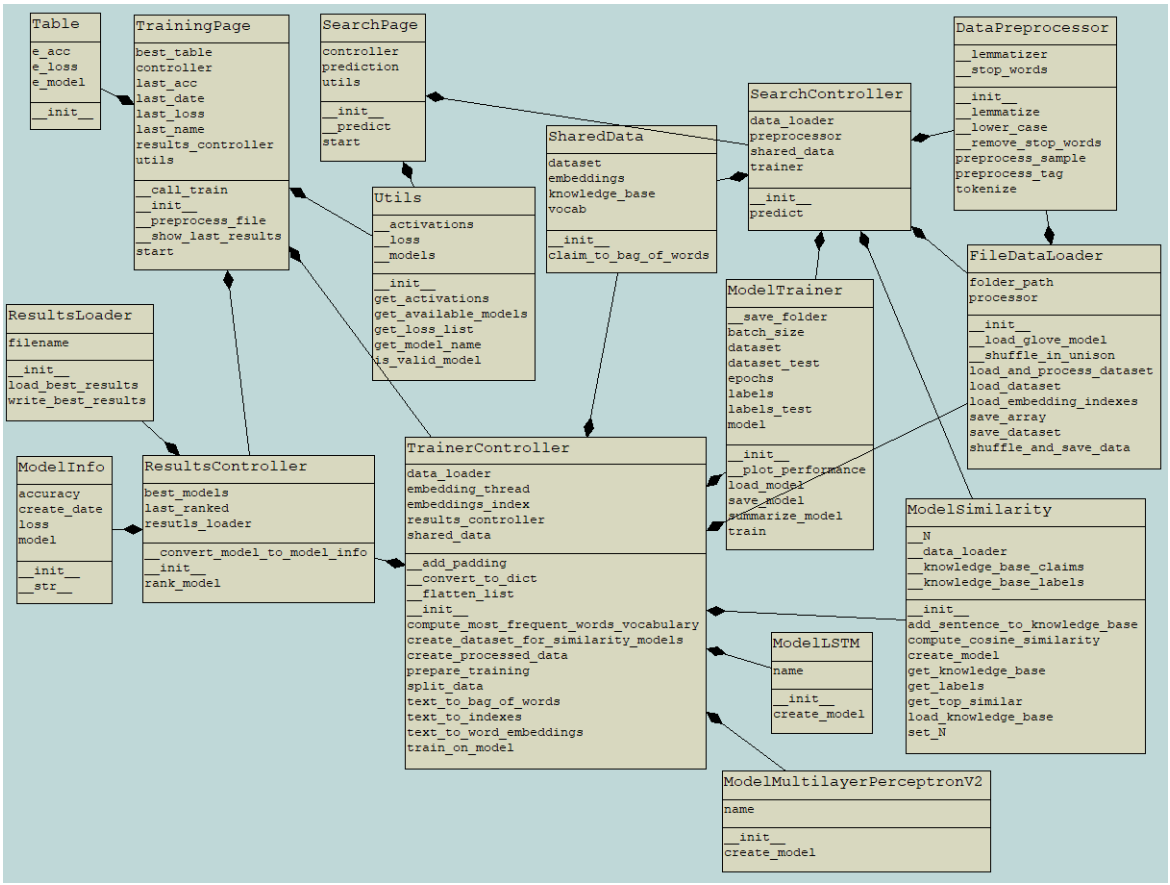

Figure 4.1: Class diagram for CoviralCheck App

Package	Version
TensorFlow	2.0.0
Keras	2.3.1
Tkinter	8.6.10
Numpy	1.19.2
NLTK	3.5
Matplotlib	3.3.2

Table 4.1: Packages and versions used

The application consists of 7 packages: *dataLoaders*, *preprocessing*, *models*, *trainers*, *controllers*, *gui* and *utils*. Operations of loading and saving datasets to and from files are done by the ”FileDataLoader” with the help of ”DataPreprocessor” which applies the cleaning data techniques. The class ”ResultsLoader” has the sole responsibility of dealing with the files that save the best models trained and their parameters.

The ”TrainingController” manages everything that is done for training mode. The different modes of the application will be further detailed in section 4.5. It creates a model and a the ”ModelTrainer”. All models are saved in the *models* package, for example: ”ModelLstm”, ”ModelMultiLayerPerceptron” etc. The *utils* pakage contains classes that hold different useful information for the GUI interface and data that needs to be shared between the application modes. The views are found in the *gui* packages.There are two views: one for the training (research) mode and one for the search (fact checking) mode.

4.4 Technology and frameworks

Since the purpose of the application was to present something practical and not to propose a new model for solving the task, I decided to leave the mathematical details behind the implementations to specialized libraries. For this reason, the application has been developed in **Python 3.6**, since it offers very good support for working with machine learning and the most popular ML libraries are in fact implemented in Python. As for version control, git was the chosen system.

The main libraries that helped implement this project were **TensorFlow** [1] and **Keras** [8]. Tensorflow is a free open-source software library developed by the Google Brain team for machine learning, with a special focus on training of deep neural networks. Keras is an API built on top of TensorFlow and it was developed with a focus on enabling fast experimentation. It offers a more intuitive way to write and understand deep learning, using **layers** and **models** at its core.

Other libraries that have been used are: **NumPy** (used for operations with high

dimensional arrays), **NLTK** (offers multitude of natural language processing operations), **matplotlib** (used for creating plots to visualize the model results), **unittest** (used for creating unit tests to be able to test isolated components of the application), **Tkinter** (used to create the GUI interface). Table 4.4 presents all the most important libraries used and the versions used.

4.5 Features

The goal of this application was to offer a solution to ease the integration of machine learning research into normal fact checking applications, so that the manual fact checking could be easily assisted or replaced by machine learning models. For that reason, the application runs into 2 modes: training for researchers and fact checking for the normal users. Switching between the two modes is done by passing the command line argument "**–research**" when running the application.

The fact checking mode can be used by any users to fact check a claim, an example is presented in Figure 4.2 . When starting the application in this mode, the "SearchController" loads the model saved during the training phase, by default it uses the MLP with similarity model. After the user introduces a claim and presses the **Search** button, the controller cleans the claim, gets the most similar claim, forms the vector representations of the two. Then, it concatenates the input and it uses the pre-trained model to predict a label. Finally, the said label is presented to the user. Figure 4.3 depicts how a prediction looks for the user.

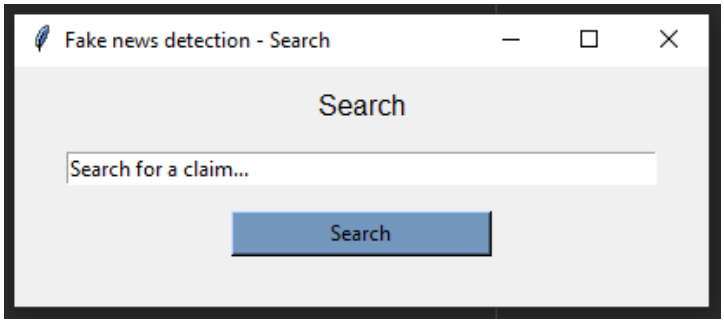

Figure 4.2: Example of the fact checking interface

The training mode is intended to be used by professional research, to help them train and validate different models for fake news detection that could be used for the fact checking interface. The interface is presented in Figure . This mode offers 3 main features: preprocessing of a dataset, adjust parameters and train a model and visualization of best and most recent results. The user can give the filename of a dataset and press the **Preprocess**. The application will apply the techniques enumerated in chapter 5.2 and save the new dataset to the file **datasets/processed_data.csv**. Figure

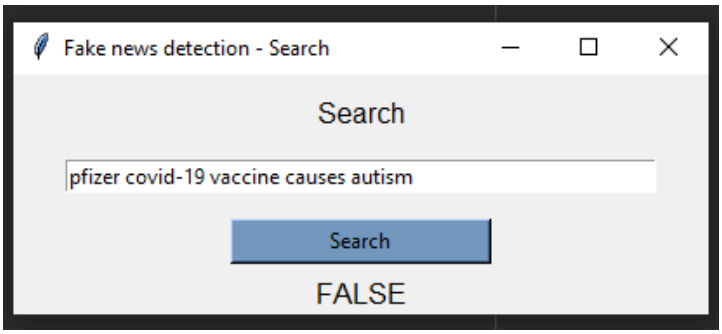

Figure 4.3: Example of prediction done

4.4 depicts how the preprocess being done is annoucned to the user. The preprocessing step is optional, a already processed dataset can also be given as input.

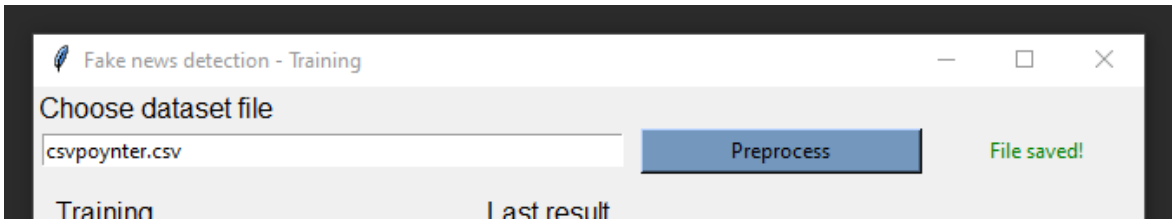

Figure 4.4: Example of the preprocessing feature

The main feature of the this mode is the training of a model. Here, the researchers can pick a model from a dropdown and adjust the main hyperparameteres. The chosen default values are explained in chapter 5.3. The application requires for a model to be chosen, otherwise an error message is shown. Then, when the preparation is ready,by pressing the **Train** button, the application starts training the chosen model and parameters in the background. Figure 4.5 depicts the way that the finalization of the training is announced to the user.

The results of the training appear on the results side (see Figure 4.6). Here the user can see the results of the most recent training session, as well as the best performing models so far. Only the best 5 models are shown in the best models list. The list is updated with every run of a new algorithm.

However, all models are saved after training in the ”savedModels” folder, each model into its own separate folder. The name of the folders have the next naming structure, so it’s easy to distinguish between them:

short model name + run date (dd-mm-yyyy HHMM)

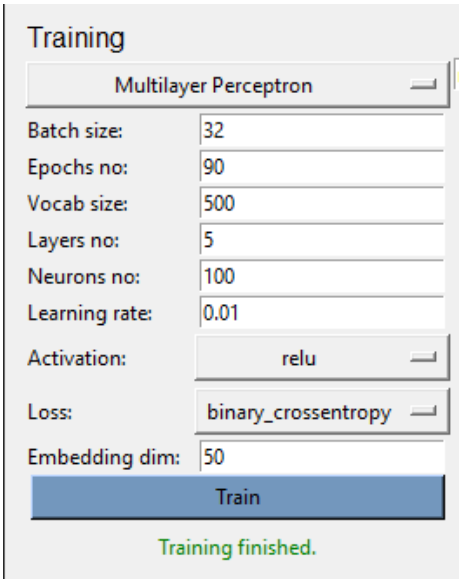

Figure 4.5: Example of the training feature

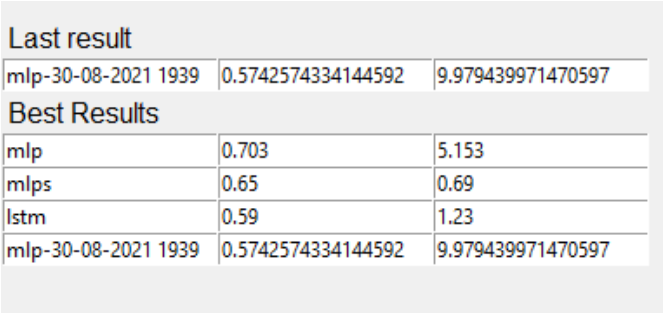

Figure 4.6: Example of the results view

Chapter 5

Experiments and results

5.1 Chosen approach

The chosen approach for the task of fake news detection was to tackle it as a binary classification problem. Given a sentence or claim the model implemented has to classify it into two categories: true or false. In real life, such a restrictive classification is not really possible, because there is much more nuance to the meaning of a text than just this. Truthful information can be taken out of context for the purpose of misleading the public to believe something else which is not corresponding to the reality. Stating that such information is either true or false is debatable. Still, I chose to only use two classes because it simplifies the task a bit and it's much easier to gather data, since not all sources use the same labelling logic.

Another aspect in which I was interested in is how a system based on similarity and machine learning would work. Moreover, I wanted to experiment if using a knowledge base, that can easily be update from trusted sources on the internet, can improve the detection accuracy. This was inspired by the idea proposed by Dr Scott A. Hale during a webinar at the University of Oxford. [28] His idea was to have a knowledge base of articles and when a new claim comes from the user, to first check if that claim has already been verified before or if there are similar claims. Starting from this, I tried to go one step forward and also combine this with a machine learning model that can learn to classify based on the claim itself and other similar claims.

5.2 Dataset

In the past years there has been interest in offering access to researchers to public sources of labelled data. However, for COVID-19 related data, there is a lack of state of the art datasets as of the time this paper is written. So, many researchers have

been gathering their own data from the web, from verified news sites, from fact checking websites or from social media.

I have chosen a similar approach of forming my dataset as Vijjali Rutvik et al. [33] and I used data provided on the fact checking site offered by Poynter. The experts from this organisation (which is also responsible with PolitiFact, another well-known fact checking website) have fact checked and labelled thousands of claims regarding COVID-19 from all over the world. Their annotation system consist of many labels such as: *True, False, Mostly-true, Mostly-False, Misleading, Manipulated* etc. However, it is very easy to filter between these labels and the data is very easy to collect using web scrapping. Besides the claim and the label, each piece of data comes with an explanation and also a link to a verified article that confirms the tag. One more thing that should be taken into account is that it mainly focuses on claims that are inaccurate and there is a gap between the number of claims tagged as *True* and *False*.

I extracted 577 claims and their explanation from Poynter which included the tags: *True, False, Mostly-true, Mostly-False, Misleading*,but I re-labelled them to only classify as True or False. One row of the data input has the format:

$$[id, claim, explanation, tag]$$

Out of the all claims, only 24 were True claims. So, I manually rephrased the False claims into True statements. While doing that I have observed there are a few claims that are repeating, maybe with slight differences, so I decided to only keep one claim of each, but because of the heavy manual work that had to be done, there's is no guarantee that there are no more duplicates, since it is hard to properly check. In total I have obtained a dataset of 1554 labelled claims. This is a significant smaller dataset than the one made by Vijjali Rutvik et al. [33], which contains over 5500 claims. This aspect will be considered when analyzing the training results.

5.2.1 Preprocessing

Data preprocessing is vital in NLP to help the models get the relevant information, especially because text is very malleable and the same word can be written in different ways. For this task, I have applied the next preprocessing methods:

- Transformed all letters to lower case
- All punctuation was removed
- Tokenization
- Stop words were removed

- Lemmatization was applied on the remaining words

Because machine learning models can only process numbers, all claims are converted into vectors of numbers using either a simple bag-of-words or word embeddings. Both options were chosen for comparison reasons, since each can extract different information from text. I will also mention that I have tried the GloVe pre-trained word embeddings created by the researchers from Stanford University [24]. I chose the smallest embeddings (50-dimensional), but in fact it didn't help the model and I observed than training my own embeddings shows better results. I think this is because the dataset is centered around COVID-19 and the key terms are missing from the pre-trained embeddings.

Finally, data is cross-validated and split 80/20 for training and testing.

5.2.2 Knowledge base

As mentioned in 5.1, one of the implemented models is using a knowledge base from which it extract similar claims. To construct this, I used 10% of the whole data to initialize the knowledge base. Then, with each new claim being labelled by the model, it will be added to the knowledge base. The main idea behind this knowledge base is that it could also be updated directly from Poynter regularly, to always have the most recent information.

5.3 Models used

In chapter 3 I presented a few state of the art models for fake news detection and transformers have been preferred in many papers. However, I was impressed by Riedel Benjamin et al. [26] paper that got the 3rd place in the stance detection task. The main reason was because they chose a fairly simple model that managed to produce unexpected good results for such a complex task. With this in mind, I have also decided to go for 2 neural network models: the multilayer perceptron, the one used in [26], and a Long Short-Term Memory neural network, because of its known capacities to process sequences and remember relevant information over time.

5.3.1 Multilayer perceptron

The first model chosen was a simple multilayer perceptron (implementation from keras [8]). This was also the model used in [26] for stance detection task. I started with the parameters that got the best results as a starting point and tried to see how the model handles the data. Moreover, I tried different both bag-of-words and word embeddings as representation (in the original paper only bag-of-words

were used), also different activation functions. Because of the training interface (see chapter 4) implemented in the application, I was able to easily try different parameters.

It was observed that the model was showing initially a very high loss. This was a bit adjusted by adding a Dropout layer between each hidden layer with a dropout rate of 0.4. The loss is still showing a fast growth, but there are a few reasons why this is happening. First, the dataset available is quite small to be used for training. The more examples a neural network has, the better it learns. Secondly, the MLP might be too simple and not able to actually capture the needed information from the data to be able to detect fake news. The structure of the MLP model is presented in Figure 5.2.

The best overall results were an accuracy of 65%, with a loss of 0.6. The parameters used for this run are presented in the Table 5.3.1. Besides these parameters, the model was using a softmax on the output layer because it's easier to interpret for a binary classification and it has a l2 regularization, compared to the dropout from the other layers. These were conventions kept from the original paper by Riedel Benjamin et al. [26]. The sigmoid function was used as activation on the hidden layers and binary cross entropy for loss computation. As for the word representation, the highest accuracy was achieved using word embedding instead of term frequency.

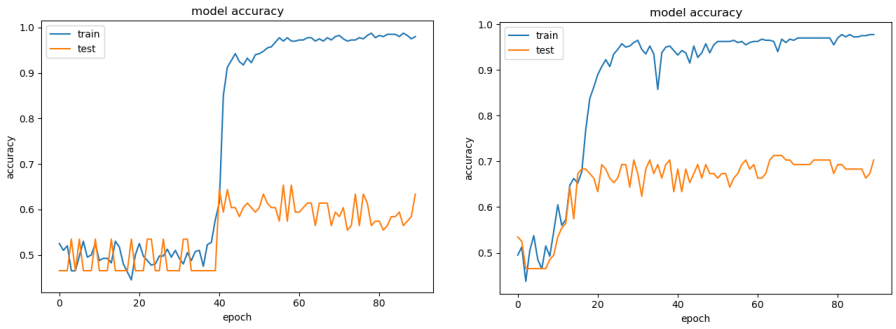

Figure 5.1: Accuracy for best performing MLP models

However, I was able to get an accuracy of 70% using Relu, a batch size of 100 and the same parameters,but the loss using Relu was growing very fast. This is also a consequence of the fact that while sigmoid has values between 0 and 1, Relu doesn't have an upper limit, but since the labels are 0 or 1, the loss is significant. Trying to analyze other metrics might give a better insight in the performance, still the accuracies are not very far apart. The graphs for both models are presented in Figure 5.1. The one on the left is using sigmoid, while the one on right ReLu. Here, we can see that the models are indeed overfitting,as the accuracy on test is much worse than the one on train.

Some interesting observations that were made about the model are that the best

Parameter	Value
Batch size	50
Epochs number	90
Vocabulary size	700
Hidden layers	5
Units per layer	100
Learning rate	0.01
Embedding dimension	100

Table 5.1: Description of parameters and best values

results are obtained with 4 or less hidden layers. More hidden layers made the neural network too complex and the model wasn't able to learn as well.

5.3.2 Multilayer perceptron with similarity

For the second model, I combined the multilayer perceptron with an information retrieval system based on similarity. The MLP is still the one from keras, but the process of training is a bit different and consists of 2 steps.

In the first phase of training, the knowledge base is loaded in the similarity model. Then, for every training input data, I form the cosine similarity matrix with the knowledge base and I get the most similar claim and its known tag. With this, I create a new dataset with a structure also inspired by the work of Riedel Benjamin et al. [26]. In this dataset, each row will have the following structure:

$$[TF\ vector\ claim, TF\ vector\ most\ similar, cosine\ similarity, known\ tag]$$

The next phase is creating the multilayer perceptron model and training it with the new dataset. The structure of the neural network is presented in Figure 5.2.

The best accuracy was of 63%. However, at this point the lack of data is showing. Not just because for the training data, but also for the knowledge base too. It's only natural that with the growth pf the knowledge base, the accuracy will also grow, since better matches will be made. Same parameters as the ones from Table 5.3.1 were used to achieve this performance, together with the sigmoid activation. My expectations were that there will be at least a slight improvement adding the similarity feature, since it offers more information. Even for humans, it's much easier to classify something you don't know by comparing it with something similar. However, I don't think that this result can be considered final for the research, not unless it is tested on a better dataset.

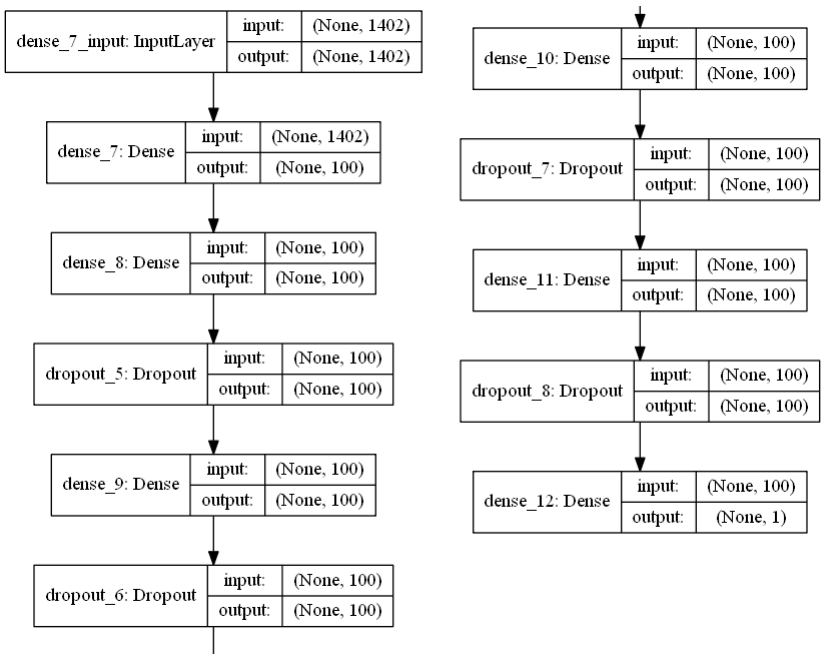

Figure 5.2: Accuracy for best performing MLP models

5.3.3 Long Short-Term Memory

For the sake of comparison,I also tried a classic Long Short-Term Memory model (implementation from keras [8]), because LSTM based models have been used to solve NLP tasks for a very long time. These neural networks are known for being able to capture order dependencies in sequences like text.

The approach for this model is the same as for the multilayer perceptron from section 5.3.1. Same default values were used and from there, through trial and error, I analyzed the best combination of parameters.

It was interesting to see that the LSTM actually performed worse than the MLP. It managed to reach an accuracy of 53% , however the model wasn't actually learning anything. The accuracy graph was flat, stuck somewhere between 0.45 and 0.55. It's true that usually LSTM need much more training data than other neural networks, so it is very possible that because of the very tiny dataset, the model is not able to distinguish any patterns. Also, because the LSTM is counting on a temporal dependency between the inputs, this model cannot be paired with the proposed similarity model, at least not with that dataset structure.

5.4 Comparison

Table 5.4 presents a comparison of the accuracy obtained by the 3 models tried. I also included a comparison with results obtained by Vijjali Rutvik et al. [33], which contain their proposed model and a few other models they have tried. I am com-

paring my results with this paper because, even if the datasets are not identical, the same source was used (poynter) and the same problem was solved: binary text classification using a knowledge base and similarity. The models in bold represent my models, while all the rest are from Vijjali's research.

Model	Accuracy
TF-IDF	0.525
LSTM	0.53
GloVe	0.579
MLP+similarity	0.63
MLP	0.70
MobileBERT	0.710
BERT	0.810
ALBERT	0.825
BERT+ALBERT	0.855

Table 5.2: Results comparison for different models

It's no surprise that transformer based models show the best results. They are a revolutionary method for natural language processing and it has been show that they are preferred for fake news detection (see chapter 3). Still, I think it is impressive to see the results of the MLP models. Even if they are not the best, we can see that they are close to the worst performing transformer model. Plus, the MLP is a much simpler model than a transformer. We should not forget that there is a huge difference in the amount of data that was available for training, so there is a chance that some ranks might change if the dataset from [33] would be made public. I believe that, overall, the results are not disappointing and that they are opening an interesting path for research in the field of fake news detection: can this task be actually solved with simpler methods?

Chapter 6

Future work and improvements

There is always space for improvements and I believe this research paper represents a start point for investigating simpler models than transformer based. Of course, one much needed improvement right now should be brought to the dataset. It would be very helpful if the data gathering would be automated. Besides getting the false claim pretty easily from the web, it would be interesting if there would be a solution to automatically generate the true claims based on the fake ones and their explanation.

There have been many approaches in literature (see chapter 3) that have used ensembles to boost the accuracy of their models. Since here I also have trained multiple models, it would be an interesting experiment to combine them in an ensemble. Of course, experimenting with other more complex neural networks models might also show improvements. For example, a CNN-BiLSTM model has been successfully used for rumor detection [25].

The problem presented in this paper has been centered on classifying claims. This could be extended to be able to verify entire articles. There has been research done to extract claims from text [17], so I would propose a model that is still trained on claims, which can lead to faster training because of the small size of a claim compared to an entire article. Then, this model can predict on the claims extracted from the article. This way, it could highlight parts that are ambiguous or suspicious. Fake news are trying to be convincing, so not the article will contain true news too, but this method can raise the awareness of the reader towards certain segments.

Application wise, there are a few features that I would like to introduce. Firstly, for the research mode, it would be nice to have a live monitoring screen while training, for example live plots and charts. This could give more insight into how well the model is working and how it can be improved. Also, right now the user can only choose from a set of predefined models. There should be more flexibility to easily construct new models by being able to combine the predefined models. As for the fact checking mode, the user definitely needs more information regarding an

input claim than just the label. Otherwise, the prediction doesn't seem trustworthy. Poynter is providing, for their fact checked claims, explanations and links to trusted sources. These information can be linked to the knowledge base and when a prediction is made, the extra information is also returned to the user. This would help the user check that the information is correct and would make the application much more trustworthy.

Chapter 7

Conclusions

Fake news have a big impact on society, especially in the era of social media where information can spread easy and fast. For the past few years, there has been a growing interest to combat this phenomenon using machine learning techniques. In this paper, I approach this problem on a highly current and specific domain: COVID-19. Because as of today there are not many datasets for the task, I attempt to create my own dataset composed by fact-checked claims from the known fact-chacking site Poynter. I also do manual work into balancing the True/False examples. On this new dataset, I train 3 models inspired by previous research done in the field, which are later put into practice to predict new claims. All this is done through a novel application: CoviralCheck, which supports both experimental and practical cases.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Iftikhar Ahmad, Muhammad Yousaf, Suhail Yousaf, and Muhammad Ovais Ahmad. Fake news detection using machine learning ensemble methods. *Complexity*, 2020, 2020.

[3] Hadeer Ahmed, Issa Traore, and Sherif Saad. Detection of online fake news using n-gram analysis and machine learning techniques. In *International conference on intelligent, secure, and dependable systems in distributed and cloud environments*, pages 127–138. Springer, 2017.

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

[5] Ben Chen, Bin Chen, Dehong Gao, Qijin Chen, Chengfu Huo, Xiaonan Meng, Weijun Ren, and Yang Zhou. Transformer-based language model fine-tuning methods for covid-19 fake news detection. In *International Workshop on Combating On line Ho st ile Posts in Regional Languages dur ing Emerge ncy Si tuation*, pages 83–92. Springer, 2021.

[6] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

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

[8] François Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[9] CIPRIAN-GABRIEL Cușmaliuc, LUCIA-GEORGIANA Coca, and ADRIAN Iftene. Identifying fake news on twitter using naive bayes, svm and random forest distributed algorithms. In *Proceedings of The 13th Edition of the International Conference on Linguistic Resources and Tools for Processing Romanian Language (ConsILR-2018) pp*, pages 177–188, 2018.

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

[11] Souvick Ghosh and Chirag Shah. Towards automatic fake news classification. *Proceedings of the Association for Information Science and Technology*, 55(1):805–807, 2018.

[12] Gong Steven. How does a Neural Network work intuitively in code? `https://gongster.medium.com/how-does-a-neural-network-work-intuitively-in-code-f51f7b2c1e3f`. Online; accessed 30 August 2019.

[13] Adrian Groza. Detecting fake news for the new coronavirus by reasoning on the covid-19 ontology. *arXiv preprint arXiv:2004.12330*, 2020.

[14] Sunil Gundapu and Radhika Mamidi. Transformer based automatic covid-19 fake news detection system. *arXiv preprint arXiv:2101.00180*, 2021.

[15] Andreas Hanselowski, PVS Avinesh, Benjamin Schiller, and Felix Caspelherr. Description of the system developed by team athene in the fnc-1. *Fake News Challenge*, 2017.

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

[17] Tom Jansen and Tobias Kuhn. Extracting core claims from scientific articles. In *Benelux Conference on Artificial Intelligence*, pages 32–46. Springer, 2016.

[18] Heejung Jwa, Dongsuk Oh, Kinam Park, Jang Mook Kang, and Heuiseok Lim. exbake: Automatic fake news detection model based on bidirectional encoder representations from transformers (bert). *Applied Sciences*, 9(19):4062, 2019.

[19] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[20] Jing Ma, Wei Gao, and Kam-Fai Wong. Rumor detection on twitter with tree-structured recursive neural networks. Association for Computational Linguistics, 2018.

[21] Maxime. What is a Transformer? `https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04`. Online; accessed 4 January 2019.

[22] Moses Kriz. Encoder-Decoder Seq2Seq Models, Clearly Explained. `https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b`. Online; accessed 12 March 2021.

[23] Cathal O'Connor and Michelle Murphy. Going viral: doctors must tackle fake news in the covid-19 pandemic. *bmj*, 369(10.1136), 2020.

[24] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[25] Neetu Rani, Praseniit Das, and Amit Kumar Bhardwaj. A hybrid deep learning model based on cnn-bilstm for rumor detection. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, pages 1423–1427, 2021.

[26] Benjamin Riedel, Isabelle Augenstein, Georgios P. Spithourakis, and Sebastian Riedel. A simple but tough-to-beat baseline for the fake news challenge stance detection task, 2018.

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

[28] Scott A.Hale. Countering the COVID-19 Misinfodemic with Text Similarity and Social Data Science. `https://www.oii.ox.ac.uk/videos/countering-the-covid-19-misinfodemic-with-text-similarity-and-socia` Online; accessed 19 June 2020.

[29] Shubham Jain. An overview of regularization techniques in deep learning. ://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

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

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[33] Rutvik Vijjali, Prathyush Potluri, Siddharth Kumar, and Sundeep Teki. Two stage transformer model for covid-19 fake news detection and fact checking, 2020.

[34] Fan Yang, Arjun Mukherjee, and Eduard Dragut. Satirical news detection and analysis using attention mechanism and linguistic features. *arXiv preprint arXiv:1709.01189*, 2017.

[35] Xinyi Zhou and Reza Zafarani. A survey of fake news: Fundamental theories, detection methods, and opportunities. *ACM Computing Surveys (CSUR)*, 53(5):1–40, 2020.