Predicting Forex Rates using Sentiment Analysis on Financial Articles

Simon Danielsson
si7660da-s@student.lu.se

Arvid Gramer
ar8384gr-s@student.lu.se

January 27, 2023

Bachelor's thesis work carried out at
the Department of Economics, Lund University.

Supervisor: Peter Jochumzen, peter.jochumzen@nek.lu.se

Abstract

We develop a deep learning model that uses financial news articles and historical exchange rates to predict the rate for the Euro/US Dollar currency pair one hour ahead. From the articles' titles and bodies we extract sentiment scores using two different transformer based classifiers – one for short text and one for long text. The long-text classifier is built and trained by us. These sentiment scores are used as a proxy for the market participants' mood. We combine the mood with technical indicators derived from historical exchange rates to make the future prediction. Our long-text sentiment classifier obtains 72% test accuracy, and our predictive Forex model achieves 44% improved test loss compared to a similar model not leveraging sentiment from article bodies. Moreover, the Forex model heavily outperforms a model not using financial articles at all. Using these results we discuss the efficiency of the EUR/USD currency market. It hints at some inefficient tendencies in the market, but more data and further experiments are needed before any conclusion can be drawn.

Keywords: Forex, efficient market hypothesis, sentiment analysis, transformer, long-short term memory

Contents

1 Introduction

Finding a way to accurately predict market prices has been a dream for speculators for as long as the modern markets have existed. However, it is in the nature of speculation that it is not perfectly possible. If it was, it would allow for possibly unbounded returns in zero time, given sufficient leverage. The prized economist Eugene Fama has contributed a lot to the view of an unbeatable market through his Efficient Market Hypothesis (EMH) (Fama 1970). Due to the amount of gold waiting at the end of the rainbow, substantial efforts have been made to find the best way of predicting markets, utilising different kinds of information as exogenous signals. One factor that strongly influences the today's price is expectations about future price movements (Adam and Nagel 2022). However, quantifying these expectations can be tricky. Ideally, one would want to read all market participant's minds. This is not yet possible. Another approach for capturing expectations and moods is to instead probe information sources where the expectations are expressed. This could be financial news articles, trending Google searches, financial blogs, Twitter feeds, to state a few.

Over the past years, advances in machine learning, natural language processing and computer hardware has enabled the academia and industry to process huge amounts of multimodal data (Kalyanathaya, Akila, and Rajesh 2019). Specifically, the development of large language models based on the transformer architecture (such as GPT3 and BERT) has shown huge potential in several natural language related tasks (Devlin et al. 2019). Tasks related to sentiment analysis, machine translation, question answering, dialogue systems, and text classification are now solved with greater accuracy (Kalyanathaya, Akila, and Rajesh 2019). Anbaee Farimani et al. 2022 utilizes this language technology to try to extract the general mood of the currency market participants through the article titles of financial news. From the article titles, they extract the mood using state-of-the-art machine learning sentiment classifiers. However, they do not leverage the article bodies to extract the mood. They combine these mood signals with more conventional technical indicators to create a predictive machine learning model, *FinBERT-SIMF*. The model predicts the future closing price on the Foreign Exchange (Forex) market for a number of currency pairs. We further develop this idea, and use it to investigate the validity of the Efficient Market Hypothesis.

Our contribution is as follows.

- We further develop FinBERT-SIMF by including the sentiment obtained from *article bodies* in the model, creating our own Forex model "Financial Bidirectional Encoder Representations from Transformers based Long Sentiment and Informative Market Feature", or *FinBERT-LSIMF*. It predicts the closing price of the EUR/USD currency pair the coming hour using financial news and historical Forex prices. We do this to see if the addition of article bodies

enhances prediction.

- To build FinBERT-LSIMF, we additionally build, train, and evaluate a long-text sentiment classifier for analysing the article bodies. It is a Longformer-based model capable of handling long sequences of text, fine-tuned on a financial sentiment task.
- We publish as open source our implementation of FinBERT-LSIMF[1] and the long-text sentiment classifier[2].

We evaluate the performance our model to see if there is an improvement to be found when including the article bodies. The result is then used to discuss whether there is reason to question the Efficient Market Hypothesis, particularly of the semi-strong form, for the EUR/USD currency exchange market between October 2020 to May 2021.

2 Related work

The study of the dynamics governing financial markets can be traced back to the 1960's (see for instance Fama 1965 or Davies and Canes 1978). A ubiquitous goal with this has been to make accurate predictions about the future state of the market.

Historically, the market dynamics have often been studied from a purely technical perspective, taking into account technical market indicators and conducting analyses using financial statistics (Alamatian, Jahan, and Fard 2021). Over the last couple of years, researchers have started to incorporate other types of information in their analyses, such as textual news data or Twitter posts, and use financial sentiment analysis for deriving additional signals used for making predictions. See Daudert 2021; Bollen, Mao, and Zeng 2011; and Nisar and Yeung 2018 for examples of such.

This processing of text can be done in numerous ways. For instance, Tetlock 2007 and Seifollahi and Shajari 2019 uses a rule-based approach and uses a fixed generic lexicon for assigning sentiment values to words, while Ferguson et al. 2015 use a domain-specific lexicon. These lexicons generally lack flexibility and are not fully capable of capturing the semantic content of the text. Antweiler and Frank 2004 and O'Hare et al. 2009, among others, improve these by taking a machine learning based approach to analysing the information content and sentiment of the texts. They use trained machine learning sentiment classifiers to infer the sentiment of texts.

[1] `https://github.com/simondanielsson/FinBERT-LSIMF`.
[2] `https://github.com/simondanielsson/longformer-financial-sentiment`

With the recent advances in natural language processing with the advent of large language models, see for instance Devlin et al. 2019, the performance of machine learning sentiment classifiers have improved significantly. Among these is the model developed by Araci 2019–a transformer-based sentiment model fine-tuned for financial articles. Lastly, Anbaee Farimani et al. 2022 combines this novel transformer-based sentiment analysis approach with classical technical analysis to build a deep learning model for predicting future prices on the Forex and cryptocurrency markets. They find an improvement in model performance when taking into account both sentiment and historical technical indicators, compared to using only the technical indicators. Specifically, they account for the sentiment derived from the titles of financial articles, but leave the article bodies unprocessed. They propose adding news bodies to see if it enhances prediction, which is what we do in our experiment.

3 Theory

We begin by briefly introducing some fundamental theory required to understand the problem we are trying to solve, and the means by which we solve it. It requires having an understanding of the Efficient Market Hypothesis, the Forex market, and how to build predictive machine learning models capable of processing both text and time series data. Note that this is by no means an extensive description of the field we are operating in; to get a proper understanding of the subject matter we refer to the sources referred to in the following sections. However, what is stated here should give enough insight into machine learning and natural language processing to follow the core concepts discussed in this paper.

The section is outlined as follows. We start by introducing the Efficient Market Hypothesis and its different forms in section 3.1. In section 3.2 we briefly cover the pricing of Forex assets and the difficulty in predicting the price trajectory. In section 3.3 we describe the core concepts in machine learning and deep learning, and explain the workflow for building and evaluating machine learning models. Finally, section 3.4 outlines the field of natural language processing and how to make a computer program understand human-language text.

3.1 The Efficient Market Hypothesis

The Efficient Market Hypothesis (EMH) states that the price of an asset on an *efficient market* reflects all available information regarding that asset (Fama 1970). When new information becomes available, the price is adjusted immediately.

Fama 1970 defines three different forms of the efficient market, depending on what information is to be regarded as available. The following list states what information is considered available for the different forms.

- *Weak form:* Historical prices only.
- *Semi-strong form:* All publicly available information. This additionally includes e.g. company statements and news articles.
- *Strong form:* All information, including private and insider information.

Under the assumption that a certain form of the EMH holds true, then it is not possible to systematically make risk-free returns using only the corresponding available information (Fama 1970).

For example, assume that the foreign exchange market is semi-strong form efficient. Then, one should not be able to make risk-free returns using neither publicly available information, nor historical prices alone. Similarly, a model using historical prices only to predict future prices should not systematically improve when additionally given public information (such as news articles).

3.2 Forex pricing

Forex trading refers to the act of buying an amount of one currency using a different currency (Sidehabi, Indrabayu, and Tandungan 2016). The price for such a transaction is the fraction between the amount of units you pay in one currency and the amount you obtain in the new currency. Thus, one always refer to a specific *currency pair* when considering Forex trading, for instance EUR/USD for buying Euros using US Dollars.

The set of factors affecting the price of a certain currency pair is vast. It spans inflation, interest rates, historical Forex prices, speculation, GDP, political stability, macroeconomic and geopolitical events, among others (Patel et al. 2014). This results in a noisy and non-stationary observed FX price time series (Deboeck 1994), tricky to effectively predict.

There are five fundamental measurements in a Forex time series; *opening price* and *closing price* measures the first and last price, respectively, in a certain time period; *high* and *low* measures the highest and lowest prices, respectively, in the same time period; *volume* measuring the total amount of units traded in the same time period (Kirkpatrick II II and Dahlquist 2010).

There exists numerous specialized technical indicators that are derived from the five fundamental quantities above (Kirkpatrick II II and Dahlquist 2010). These indicators are commonly used in

the literature to augment the Forex price series, and are commonly used as input to predictive models along with the five fundamental quantities. We do not cover in detail how to compute these technical indicators; they are included in many software packages, such as the *Technical Analysis*[3] Python distribution.

3.3 Machine Learning

Machine learning (ML) is all about distilling patterns and information from raw data (Goodfellow, Bengio, and Courville 2016). This is often referred to as 'learning from data'–hence the name. In practice, this often amounts to creating a mathematical function (or *model*) which accepts as inputs raw data (or so-called *features*), and outputs a value (*outputs* or *labels*) (Hastie, Tibshirani, and Friedman 2001). This function can subsequently be used to make *inferences*, or *predictions*, about a certain subject.

As a first example from the financial world, the features could be descriptive values of a stock (such as stock price volatility the past day), and the output could be the stock price in the future. The function that you create could then be used to predict future stock prices using historical price volatilities for that same stock.

Creating any function that maps features into outputs is not difficult. However, what is tricky is to find a function that is able to do these predictions *accurately* (Hastie, Tibshirani, and Friedman 2001). We also want the function to predict accurately on examples it has never seen before. Creating such accurate functions by hand is almost always infeasible. Instead, one could choose a family of functions with *tunable parameters* – whose values are unknown in the beginning – and use an algorithm that finds the set of parameters that make the function predict the data the best. This algorithm usually minimizes some type of error function – or *loss* – such that the model in some sense is as good as possible (Hastie, Tibshirani, and Friedman 2001). Minimizing the loss is often referred to as *training* the model.

The canonical example of such a model is linear regression. It has parameters that we can tune (the coefficients), and as an algorithm to find the set of optimal parameters we minimize the squared deviation between the model outputs and actual values (equivalently, the $L2$-loss). The final, trained, model is then a linear function with parameters determined by this minimization procedure.

What is described above, and also what will only be discussed in the following sections, is referred to as *supervised learning* for *parametric models*. This is merely a small subset of what the vast field of

[3] `https://pypi.org/project/ta-py/`

machine learning offers. Neither will we give a thorough treatment of the probabilistic nature of ML which is required to fully understand some of the concepts described.

There are several programmatic frameworks for building machine learning models. Among the most popular are TensorFlow and PyTorch, which have API's available for e.g. Python and C++.

This section is outlined as follows. In section 3.3.1 we describe the main components required to build a machine learning model. Section 3.3.2 outlines how to train the model on a certain task. In section 3.3.3 we introduce deep learning and how it connects to machine learning; section 3.3.4 shows the simplest type of deep learning layer and briefly cover the topic of activation functions. Following, section 3.3.5 introduces the LSTM layer. Section 3.3.6 covers how more complex models can be created out of LSTM layers. Lastly, in section 3.3.7, we display some common issues and pitfalls one might stumble upon when building a model, and what to take into account when addressing these.

3.3.1 Creating a model

There are four key components required to create a machine learning model.

First of all, one has to decide on a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ to train, parametrized by some $\theta \in \Theta \subseteq \mathbb{R}^k$ (Hastie, Tibshirani, and Friedman 2001). Here, $\mathcal{X}$ is the set of features and $\mathcal{Y}$ the set of possible outputs. For instance, simply defining the mapping as $f_\theta : x \mapsto \beta^\intercal x$ for $x \in \mathcal{X} = \mathbb{R}^k$ gives us linear regression. The parameters in this case are $\theta = \beta$. However, there is no restriction in how complex we can make f_θ. More complex models are usually required for capturing complex relations between the the input and output data (Hastie, Tibshirani, and Friedman 2001). Complexity, or *capacity*, can often be described in terms of the number of parameters of the model (Goodfellow, Bengio, and Courville 2016).

The second component is the data. In order to make the model capable of capturing the entire relation between $\mathcal{X}$ and $\mathcal{Y}$, we need to provide it with (many) examples of this relation (Hastie, Tibshirani, and Friedman 2001). Since the ultimate goal is to find a function which performs well on data it has *never seen before*, we also need data that is not fed into the model during training. This is often referred to as a *test set*.

Let us bring back the previous example of predicting stock prices using past volatilities to make this more clear. We want to create a model for predicting stock prices in the future using information about the stock that is available to us at that point in time. What we can do then is to train a model on *historical examples* of the relation between the stock price volatility the past hour and the

actual price in the future. Given that the relation between past price volatility and actual price is kept constant, training a model on a set of such examples should give a model that is accurate on predicting stock prices in the future as well. In this case, examples of past price volatilities would constitute examples in $\mathcal{X}$, and the actual stock prices (that we aim to predict) are in $\mathcal{Y}$.

Thirdly, we need a loss function $L = L(y, f_\theta(x))$. Its purpose is to penalize erroneous predictions, i.e. be large when the prediction $f_\theta(x)$ is far from the real value y (Hastie, Tibshirani, and Friedman 2001). This means that if we have a historical example $(x, y) \in \mathcal{X} \times \mathcal{Y}$ that we want f_θ to reflect (meaning $f_\theta(x) \approx y$) then $L(y, f_\theta(x))$ should be as small as possible. If we provide an entire dataset $X = (x_1, x_2, ..., x_n) \in \mathcal{X}^n$ and $Y = (y_1, y_2, ..., y_n) \in \mathcal{Y}^n$, then $L(y_i, f_\theta(x_i))$ should be small *for all* i. Equivalently, $\sum_{i=1}^{n} L(y_i, f_\theta(x_i))$ should be small. The loss can therefore indicate the "goodness" of our model when applied to our dataset. An example of a loss used in the real-valued output setting is the well known $L2$-loss (or residual sum of squares). In the setting where outputs y are in some finite set of categories, a popular choice of loss is the *categorical cross entropy* (Zhang and Sabuncu 2018).

Finally, we need an algorithm that finds the set of parameters that makes the model as good as possible in our relation to our loss. This amounts to finding the parameter $\theta = \hat{\theta}$ that minimizes the loss across the dataset (Hastie, Tibshirani, and Friedman 2001). If we judiciously choose a loss, and manage to minimize it, one would expect to get a good model. Unfortunately this is not generally the case: we will discuss this problem further in section 3.3.7. Also, the minimization problem

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \sum_i L(y_i, f_\theta(x_i)) \tag{1}$$

generally does not admit an algebraic solution (Sra, Nowozin, and Wright 2011). This means we resort to numerical methods for finding the optimal function parameters. We also have to specify what we mean by "finding the minimum"; in general the loss is non-convex and non-smooth, meaning that we cannot be certain that there exists a unique minimum, there might be several local minimas, or a minimum value might not exist at all (Sra, Nowozin, and Wright 2011). Figure 1 shows what a loss function landscape might look like in a low-dimensional setting. The field of mathematical optimization is thus highly relevant for machine learning, and has brought lots of techniques and strategies for finding $\hat{\theta}$ under different conditions (Hastie, Tibshirani, and Friedman 2001).

Next, we describe one of the most popular approaches to try to solve (1).

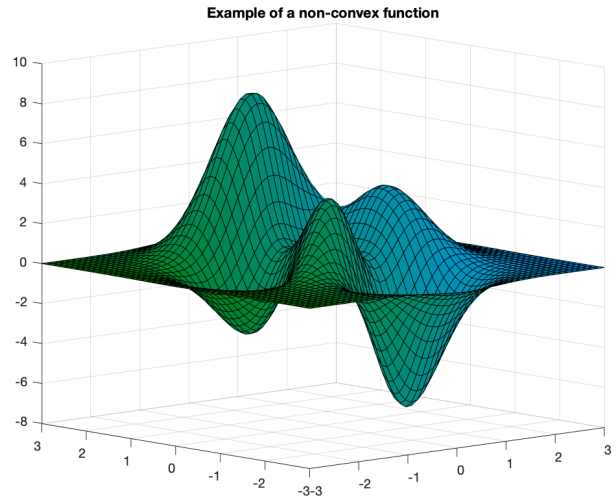

Figure 1 A low-dimensional example of a non-convex loss function. It may contain several stationary points which could be local or global optimums, or neither of them.

3.3.2 Training the model

The act of *training* a model essentially means finding the best set of parameters $\hat{\theta}$ for f_θ, done by minimizing some loss function. Numerical methods for doing this are often *iterative*, meaning we step by step try to make small progression toward finding $\hat{\theta}$. First we describe the method *gradient descent*, which forms the basis for many optimization techniques. We then introduce *stochastic gradient descent* which solves the problem of running gradient descent on huge datasets. Finally we briefly discuss the *Adam* optimizer, a popular adaptive optimizer with good track record in practical Machine Learning.

The motivation behind gradient descent is based on one fundamental observation; if we step in the loss function landscape in the direction of the *negative gradient*, we can expect the value of the loss to decrease (given that the *step size* is sufficiently small) (Sra, Nowozin, and Wright 2011). Refer to Figure 2 for intuition. If we repeat this stepping over and over, we should hopefully get to a smaller and smaller value of the loss function. In certain cases, this will make us end up at the optimal parameter value $\hat{\theta}$. Formally, we use the update rule

$$\theta \leftarrow \theta - \lambda g \tag{2}$$

where $g = \nabla_\theta \sum_i L(y_i, f_\theta(x_i))$ and $\lambda \in \mathbb{R}$ is the step size, or *learning rate* (Goodfellow, Bengio, and Courville 2016). There are two main issues with this schema. First, note that this gradient has to

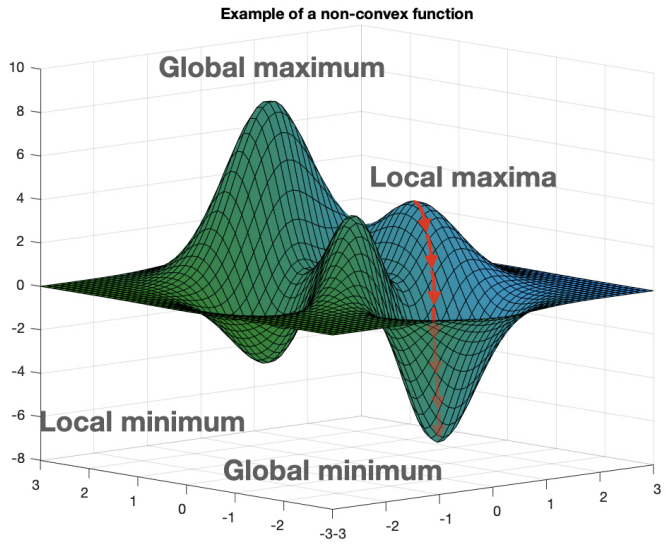

Figure 2 Illustration of how gradient descent is used to find minimum of the function. However, it is not guaranteed to find the global minima.

be computed over all points in the training dataset. To get good performance one usually needs a huge training dataset, meaning this will be a very expensive computation (Goodfellow, Bengio, and Courville 2016). Second, choosing a good value of the step size can also be tricky. If it is too small we will require more steps, but if it's too large we cannot be certain that we make progress in every step (Sra, Nowozin, and Wright 2011). Such a parameter is called a *hyperparameter* - one that is not tuned by the learning processed but rather has to be chosen judicously by the user of the algorithm.

Stochastic gradient descent (SGD) aims to solve the first issue addressed above regarding computational cost (Goodfellow, Bengio, and Courville 2016). Instead of computing the gradient of the loss over all data points, one randomly samples an index j from the dataset, and calculates the gradient of the loss on that single data point x_j (Goodfellow, Bengio, and Courville 2016). This requires substantially less computational effort. Formally, we pick $g_j = \nabla_\theta L(y_j, f_\theta(x_j))$. Although this is an unbiased estimate of the full gradient g, it's corresponding random variable might have quite some variance (Goodfellow, Bengio, and Courville 2016). This is often observed as quite an unstable progress when observing the loss function over the iterations. Also, note that we still compute the gradient over θ. If the model is very complex, meaning we have a large number of parameters, using SGD might still require expensive computations.

A variant of SGD is *mini-batch* SGD. Instead of calculating the gradients on only a single data point x_j, it is computed on a small set of m samples (called a *batch*) (Goodfellow, Bengio, and Courville 2016). The size of the batch is usually a trade-off between variance of the gradient estimates, the computational effort required, and the need to tune m based on the dynamic between m and the other hyperparameters (such as the learning rate) (Goodfellow, Bengio, and Courville 2016). We

say that an *epoch* has passed when we have iterated through our batches and calculated gradients using the entire dataset.

Partly to address the stability concern of SGD, the *Adam* algorithm has been developed (Kingma and Ba 2014). It can be regarded as SGD in addition to two new components. First of all it uses *momentum*, essentially meaning that the "gradient" at iteration i includes gradient computations from previous iterations. This improves stability. Next, it includes *adaptive* learning rate, meaning the learning rate λ is modified in each time step based on the previous gradient computations. This removes some effort from choosing a good, fixed, learning rate before training has even started (Kingma and Ba 2014). Luckily, Adam is readily available in the common ML Python frameworks such as TensorFlow and PyTorch.

To summarize, there exists numerous techniques to find optimal model parameters for a given training set. The strategies are often iterative, and computationally expensive if the model has high capacity. Also, there is often little to no theoretical foundation assuring we will actually converge to the optimal parameters $\hat{\theta}$. We have not yet described in detail any actual models f_θ. This is done in the following section.

3.3.3 What is Deep Learning?

Deep learning (DL) can be described as a framework for building machine learning models using compositions of *layers* (Goodfellow, Bengio, and Courville 2016). A layer can be seen as a single function that maps vectors to other vectors, with easily configurable capacity. These layers can be composed into into extremely complex systems of layers, called *artificial neural networks* (ANNs) (Goodfellow, Bengio, and Courville 2016). Depending on the task to be solved, one chooses different *architectures*, referring to different compositions of different types of layers (Goodfellow, Bengio, and Courville 2016). Deep learning can in some sense be seen as the act of building a very complex function using the composition of many simple, non-linear components. However, we note that building well-performing DL model architectures is no easy task: all the choices regarding architecture has to be regarded as hyperparameters that has to be chosen by the builder of the model (Goodfellow, Bengio, and Courville 2016). The performance of the model usually varies strongly with the architectural hyperparameters.

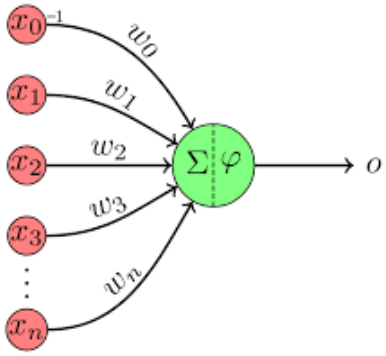

Figure 3 Graphical depiction of the perceptron. A linear combination is taken of the input vector components x_j and weights w_j, $j = 1, ..., n$, which is then fed through an activation function φ to produce the output. Image: *Perceptron-unit* by MartinThoma 2014 under CC0 license.

3.3.4 The simplest type of layer & activation functions

The simplest type of layer is called a *perceptron* (Goodfellow, Bengio, and Courville 2016). It passes a vector x to an output vector $y = f_\theta(x)$. The perceptron includes two operations; first a linear combination is formed between the components of the input vector x and some weights w. Then a function φ is applied to form the output

$$y = \varphi(w^\intercal x + b), \tag{3}$$

where $b \in \mathbb{R}$ is an additive bias (Goodfellow, Bengio, and Courville 2016). The weights w are the parameters to be optimized during training. The function φ is usually called the *activation function* (or simply *activation*), and is typically very simple but non-linear. We directly see that choosing the activation as the identity function, i.e. $\varphi : z \mapsto z$, recovers the familiar linear function. See Figure 3 for a graphical depiction of the perceptron.

The power of the perceptron however comes when composing multiple perceptrons. This creates what is called a *feedforward network* (Goodfellow, Bengio, and Courville 2016)–a flexible non-linear function made out of simple components. A typical feedforward network is displayed in Figure 4. It can be depicted as a graph, where the nodes (or *neurons*) are input or output values of each computation in (3), and where each connecting edge has the corresponding weights. The first layer through which the input x is fed is called the *input layer*; the intermediate layers are commonly referred to as *hidden layers*; the final layer, which produces the final output, is called the *output layer*.

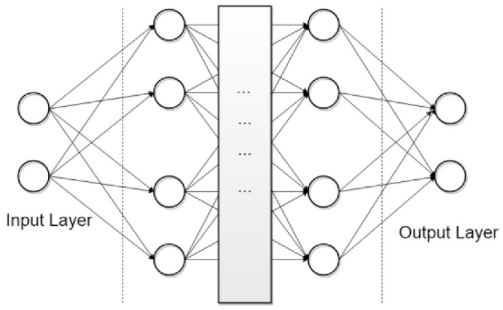

Figure 4 A typical feedforward network. It is created by composing multiple perceptrons. The nodes, or *neurons*, of the network represent the input or output of the computation in each perceptron. Image: *Neural Network* by Stetco et al. 2020 under CC0 license.

To make a prediction with a feedforward network (commonly referred to as making a *forward pass*), you

1. input the input vector x into the input layer,
2. run the computation in (3) for each neuron in every layer of the network,
3. consider as output y the values of the neurons in the output layer

Likewise, the power of the feedforward network comes from choosing a good, non-linear activation function φ. If one chooses a *linear* activation function it would simply results in a re-parametrized linear function. This might not fully capture the complex relation between the features and the output variable. Instead, it is customary to choose a simple but non-linear activation function for each layer. A commonly used activation function is the Rectified Linear Unit (or simply *ReLU*), defined as $\varphi(z) = \max(z, 0)$ (Agarap 2018). In classification tasks, one usually chooses the last activation as the *softmax function*. The softmax function maps the final layer's values to probabilities of the possible classes, meaning that we get an estimate of how probable each class is for a given input (c.f. multinomial logistic regression) (Hastie, Tibshirani, and Friedman 2001).

The feedforward network does not have the ideal design for all types of input data. Next, we describe a DL layer specialized for processing time series data.

3.3.5 The LSTM layer

The *Long Short-Term Memory* (LSTM) is a special type of neural network layer used for dealing with data of sequential nature (Goodfellow, Bengio, and Courville 2016). This data is could be

time series data. Its high level functionality is very similar to the feedforward network mentioned previously: it inputs some data vectors $x_1, x_2, ..., x_T$ and outputs a vector y. The main difference is that it not only accept a single vector as input, but rather a sequence of them (Goodfellow, Bengio, and Courville 2016). To produce its final output y the LSTM processes each of the inputs x_j sequentially (Goodfellow, Bengio, and Courville 2016). There exists several neural network layers specialized for sequential data; the LSTM is a popular one frequently used in academia and the industry (Sherstinsky 2020).

A canonical use-case of the LSTM is for building models using time series data, for instance a model predicting future stock price using historical stock prices. In that case, each x_j would be a vector of (historical) prices at time j, and the output y is the predicted future price. There are two fundamental properties of the LSTM that allows for easy processing of sequential data.

Firstly, the LSTM is capable of processing data of variable sequence length (Sherstinsky 2020). This is useful as the one does not have to train an entirely new neural network if the length T of the input data sequence changes. Compare this to the feedforward network in Figure 4. This network has two input neurons, meaning it can only handle inputs of length two. If one would want to process sequences of length three, a new network with three input neurons would have to be built and trained. There is no such restriction for the LSTM.

Second, the LSTM can learn to remember important aspects of past data to make a future prediction (Goodfellow, Bengio, and Courville 2016). Given that there is a relationship between historical and future values, it would be reasonable to consider it beneficial to remember important parts of the historical data also when processing future data.

Formally, the LSTM obtains these two properties using a *feedback loop*. In a forward pass at time t, there is one output o_t but also an output h_t (for *hidden state*) that is fed into the model during the forward pass at time step $t + 1$ (Zaccone, Karim, and Menshawy 2017). See Figure 5 for a high-level graphical depiction of an LSTM. The ambition is to have the hidden state h_t be some summary of the important information in the previous inputs up to time t (Goodfellow, Bengio, and Courville 2016). Particularly, we want it to capture information that could be important for the model to have when making future predictions. The more important the information in the hidden state is, the more likely the model is to perform well on the specific task. We omit a mathematical specification of how the model learns to remember past information–the interested reader should consult for instance Goodfellow, Bengio, and Courville 2016 or Sherstinsky 2020.

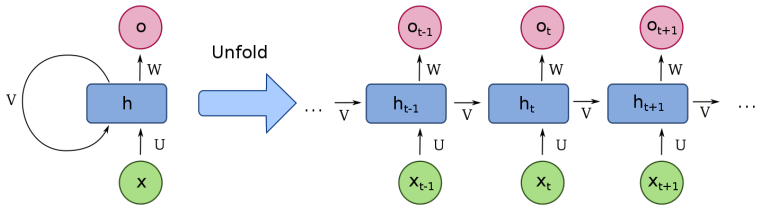

Figure 5 Overview of an LSTM. At each time step, an input x_t is fed to the network together with the previous hidden state h_t. This produces an output, and a new hidden state. The process is repeated for the length of the input sequence. The right subimage shows an unfolded version of the RNN where inputs and outputs are displayed over time. The purpose of the hidden state is to remember important information in the past inputs that could be leveraged in future forward passes. Image: *Recurrent neural network unfold* by fdeloche 2017 under Creative Commons Attribution-ShareAlike 4.0 International license.

3.3.6 Creating more complex models out of LSTM's

One can create a powerful model out of LSTM's by simply stacking multiple LSTM's on top of each other. Such a model is commonly referred to as a *stacked LSTM* (Goodfellow, Bengio, and Courville 2016). Its purpose is to increase the model's capacity and potentially allow for capturing more complex relationships between the input sequence and output variable (Goodfellow, Bengio, and Courville 2016). Figure 6 displays what such a model might look like.

In practice, creating a stacked LSTM amounts to considering the outputs of the LSTM layer i as the input to the next LSTM layer $i + 1$. The final output of the stacked LSTM model is the final output of the final layer. It is up to the model builder to decide on how many LSTM's you would like to stack.

It is straight forward to add even more complexity to this stacked LSTM model thanks to the fact that the format of its inputs and outputs are unchanged. It still inputs a sequence of vectors, and outputs a single vector. As is ubiquitous in deep learning, we can compose this stacked LSTM model with additional layers (Goodfellow, Bengio, and Courville 2016), for instance by connecting a feedforward network to its output layer.

More specifically, composing an LSTM with a feedforward layer with a single output neuron results in a regressor for sequential data. In contrast, composing the LSTM with a feedforward layer with a softmax activation function (a *classification head*) results in a multiclass classifier. We say that we *attach a head* when composing a model with a certain output layer. Attaching different heads typically results in completely different types of models.

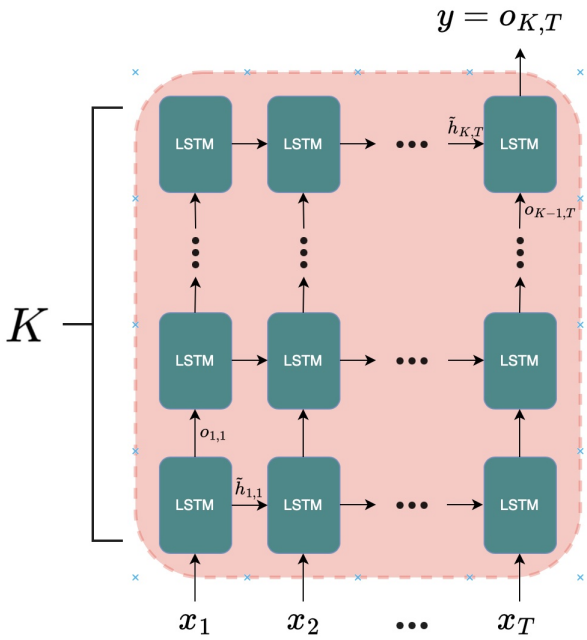

Figure 6 Overview of a stacked LSTM. It consists of $K \geq 1$ stacked LSTM units where the outputs of one layer is considered the input of the next. The output of the full architecture is the final output of the final layer.

To summarize, the LSTM layer is a popular and flexible layer for creating models that process sequential data to make predictions. They are generally able to more easily capture sequential relationships in data than pure feedforward networks, and are able to process sequences of arbitrary length. Furthermore, they can be easily combined with other layers to construct complex models of different types.

3.3.7 Things to consider when building a model

Unfortunately, obtaining a performant model is not as easy as simply minimizing the loss function. Here we talk about a few common issues one might stumble upon, and some ways to address these. We also provide some general concepts that are useful to consider when building a performant machine learning model.

The first issue is related to how one defines "performant". The ultimate goal of the model is to predict accurately (in some sense) *on completely new data*, i.e. data the model has not been trained on (Hastie, Tibshirani, and Friedman 2001). There is no real use in predicting well on the training set as we already have corresponding labels for that input data. Instead we want to be able to use the model in real-life scenarios where the data is likely completely disjoint from the data seen at training time. The way this is commonly solved is by dividing the available data into three separate

datasets, called a *training*, *validation*, and *test set* (Hastie, Tibshirani, and Friedman 2001). The purpose of this is essentially to create artificial "unseen" data: we train the model on the training set and choose the hyperparameters as those yielding the best model performance on the validation set. Then, one tests and evaluates the trained model on the test set. The performance on the test set is what we consider as the model performance, as this most accurately reflects the performance the model would have in a real-life scenario. Furthermore, there are several ways to get model performance estimates of lower variance (at the cost of additional compute resources): on of these is called cross-validation (Hastie, Tibshirani, and Friedman 2001).

Second, a common issue is directly related to the issue of a suboptimal optimization solution touched upon in section 3.3.2. If the model has a huge amount of parameters, it might be infeasible (from a time or cost perspective) to optimize the model parameters fully. The implication of this is that one would obtain a suboptimal model. The solution is might simply be to train the model for a longer period of time or using more compute resources. However, this might not always be possible.

Third, even though the optimizer converges it might not yield a global minimum (Sra, Nowozin, and Wright 2011). This means the model has converged to a local minimum in the loss landscape. There are numerous advanced techniques for addressing this issue. One common strategy to improve convergence is to use a so-called *learning rate scheduler*: a schema for how the learning rate $\lambda = \lambda(t)$ should change across epochs t (Kim et al. 2021). This alleviates the act of judiciously choosing a fixed learning rate. If the loss does not decrease over several iterations it often means that the learning rate is too large. In this case the learning rate should be decreased. An example of a learning rate scheduler is a procedure which automatically reduces the learning rate by a constant factor when the loss is not improving. Optimizers using adaptive learning rates, such as Adam, also help to solve the issue with choosing a good learning rate ahead of training time.

Fourth, it is often a good idea not to train a model until it fits the training data too well (Hastie, Tibshirani, and Friedman 2001). Otherwise, the model is said to be *overfitted*. The reason for this relates back to the way we define "performant": ultimately we are interested in the model's ability to generalize to unseen data. We say that we apply *regularization* when one intentionally does not completely fit the training data. One regularization technique is to stop the training before the loss function has converged. This is called *early stopping* (Hastie, Tibshirani, and Friedman 2001). Another common way to regularize the model is to use *dropout* (Srivastava et al. 2014). Dropout means that each node in the neural network has a certain probability to be "turned off" during each forward pass, which according to Srivastava et al. 2014 "prevents units from co-adapting too much". Lastly, *weight decay* is another popular way of applying regularization (Hastie, Tibshirani, and Friedman 2001). Weight decay penalizes large parameter values; the amount to penalize is a

chosen number considered as a hyperparameter.

Fifth, there are some statistical properties of the input data playing a role in how well the model can be trained. It can be shown that many models enjoy some nice properties when the input data is normalized (meaning having zero mean and unit variance) (Goodfellow, Bengio, and Courville 2016). Additionally, it can be shown that normalizing outputs of specific layers in deep learning models improve training. This is called *batch normalization*, which refers to the fact that this normalization is done with respect to the statistics of each batch in a forward pass (Ioffe and Szegedy 2015). Normalizing the inputs and employing batch normalization is frequently done in DL applications.

Sixth, there is another difficulty related to generalization but this time with respect to the *data quantity*. To generalize well, model's usually have to be trained on huge amounts of data, and adding more training data usually results in better models (Goodfellow, Bengio, and Courville 2016). This can be a very tricky problem to solve given the difficulty and cost of collecting high-quality labeled data.

Lastly, there is also the topic of *data quality*. To make sure that the model performs well over time, up-to-date data has to be provided for training such that the model is able to capture patterns that are relevant today (Goodfellow, Bengio, and Courville 2016). This is especially crucial if the underlying data distribution changes significantly over time, as that would otherwise be detrimental for model performance. It is also necessary to make sure that the data used for training is not biased towards certain groups (Gohel, Singh, and Mohanty 2021). In the worst case, this could make the model disfavour e.g. minorities and reinforce inequalities with respect to for instance gender and race.

To summarize, there are several aspects making it difficult to create a good machine learning model. However, the community has found good solutions to some of these issues - but many problems are still unsolved. In practice, one often has to try lots of different strategies and configurations to see what works best for the application in question. It is also imperative that one have a thorough understanding of the data source used for training, as this might otherwise unintentionally propagate assumptions and bias to the model output.

3.4 Natural Language Processing

Natural Language Processing (NLP) is a subfield of artificial intelligence, with the goal of enabling a computer to make sense of *natural language*. Natural language is what humans use to communicate with each other, as apposed to constructed languages used for describing logic or programming lan-

guages (Goodfellow, Bengio, and Courville 2016). The field's applications and research output has grown tremendously over the past years, mainly due to the advent of deep learning-style language processing (Otter, Medina, and Kalita 2018). Here we give a brief and narrow introduction to the field. This will help the reader make sense of coming sections - however the unacquainted reader is recommended to consult other sources on NLP such as Aggarwal 2022 or Goodfellow, Bengio, and Courville 2016.

First we explain some of the core problems that NLP aims to solve. Second, we describe some traditional approaches and methods to solve this problem. Lastly we proceed towards explaining some of the recent advances within the field of NLP and display some popular and efficient ways to make a computer understand text data.

3.4.1 Problems in NLP

If we want to leverage the machine learning and deep learning models we encountered in section 3.3 there are a few issues that first have to be dealt with. Firstly, computers work with numbers and not words, meaning that we need to map words to numbers. Secondly, there are a lot of ambiguity and contextual information in natural languages that has to be somehow captured by the model. The same combination of letters mean one thing in one context and something else in another context. Lastly, processing large amounts of text is often computationally intense (Chowdhary 2020).

3.4.2 A first approach

The main issue in natural language processing is that in order for a computer to handle and understand text, the text first has to be translated to numbers. The most straight forward approach is to simply count the number of occurrences of each word in every document in a *corpus* (the set of documents at interest). Aggarwal 2022 describes how normally very uninteresting words such as *is, a, the* are first filtered out. The author continues to describe how each remaining word is then assigned a dimension in a high dimensional vector space, where the length in each dimension is the frequency of that word. He states that this method is called the *bag-of-words*, since we view a text as a bag containing the words without respect to order and context. Each document in the corpus would then be represented and handled as a vector in this vector space. For example, the sentence "USD is appreciating against EUR due to Fed adopting a more hawkish interest rate policy than BoE" would be represented by a vector with ones at positions representing the words *USD, appreciating, against*, and so on.

However, merely the frequency of each word does not carry a lot of insight on its own, but it can if the importance of each word is extracted. For instance, this could be done by giving more weight to important words in the vector representation of the text. One way of measuring the importance of a word in a text is for example if an unusual word is frequent in the text. It then probably carries a lot of information regarding what the text is about. For example, the presence of "USD", "EUR", "Fed", "BoE" and "hawkish" in the above sentence would explain a lot about the subject. Another problem with the bag-of-words approach is that the number of words is very large, which results in sparse, very high-dimensional vector spaces, where most dimensions are zero. This makes handling the data very inefficient in terms of memory and compute usage (Aggarwal 2022). Another issue is that the context of the word is unaccounted for. For example, the sentence discussed above would have the exact same vector representation as "EUR is appreciating against USD due to BoE adopting a more hawkish interest rate policy than Fed", which would have the direct opposite meaning. A third is that for short pieces of text, word frequency seldom contains enough information.

3.4.3 Representing words with embeddings

A way around the aforementioned obstacles is to use *embeddings*, where each word is instead represented by a vector of numbers rather than one dimension in a vector. The aim is to create a mapping of words to vectors, such that words with similar meaning are represented by vectors that are close to each other – in some sense – in the vector space (Aggarwal 2022). A simplified illustration of this is found in Figure 7. With such a representation, there is much less sparsity in the vector space since there are much fewer, but more used, dimensions. One example of this are the algorithms *word2vec* and *continuous bag-of-words*, first published in 2013 (Mikolov et al. 2013). The author's approach is to train a single hidden layer neural network to predict each word in the corpus from the t words before and after it. With t being for example 4, they slide a window of length $2t + 1 = 9$ through the corpus, and let the weights in the hidden layer be the representation of that middle word. Word2vec generates a 300-dimensional vector space in contrast to bag-of-words which has a number of dimensions equal to the number of unique words (Mikolov et al. 2013). According to Aggarwal 2022, this can be up to hundreds of thousands of dimensions. It is called a continuous bag-of-words since it treats the surrounding words as an ordinary bag of words–without respect to order (Aggarwal 2022). This makes the algorithm take some contextual words into consideration, namely the words within the sliding window, but not more than that and not the order in which they are presented.

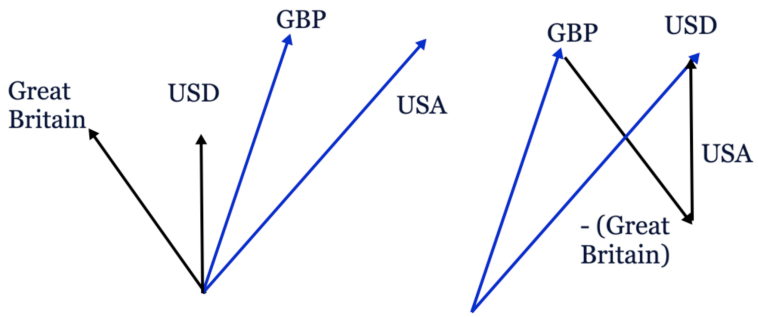

Figure 7 The main idea of embeddings: map each word to a vector in a vector space where similar words are represented by similar vectors. Left: one way of representing the words in two dimensions. Right: showing that Great British Pound - Great Britain + USA ≈ US Dollar

3.4.4 Transformers and BERT

A recent breakthrough in deep learning is the development of the *self-attention* mechanism (Wolf et al. 2020). A model employing self-attention is called a *transformer* (Vaswani et al. 2017). These models have been had a great influence on the performance of models on NLP tasks (Wolf et al. 2020).

Attention is, just as for the brain, a way to focus on the most important pieces of information. When it comes to NLP, the attention mechanism is a trained way of focusing on the most important words in a sequence. This makes the model being able to use information from a wider span of text and understand the relationship between words of a text. In contrast to continuous bag-of-words, where only surrounding words were taken into account for predicting a word, attention is capable of incorporating "global" information when inferring a word (Vaswani et al. 2017).

To be more precise, the transformer takes as input a sequence of text and outputs *contextualised embeddings* for each word in the sequence (Vaswani et al. 2017). A contextualised embedding is a vector representation of a word that might differ depending on the context in which the word is present. As such, words that have different meanings but same spelling are treated differently (Wolf et al. 2020). These embeddings can subsequently be consumed by a downstream layer to form a full deep learning model.

Let's consider an example. The expression "bull market" have a vastly different interpretation if it is found in a Financial Times article than it would in a tourist guide to Seville. Here, the context in which the word appears is crucial for determining its meaning. It is evident that having a model capable of discerning these semantic colors could be quite useful.

On a mathematical level, (scaled dot-product) self-attention is fundamentally based on matrix mul-

tiplication (Vaswani et al. 2017). Thanks to this, among other things[4], the computation can be efficiently parallelised. However, the time complexity of this operation (with respect to the length of the input text sequence) is quadratic, meaning the operation will be expensive when the sequence length is long. This issue is addressed in section 3.4.6.

A specific model implementation based on the transformer architecture is the *Bidirectional Encoder Representations from Transformers* (BERT), developed by Google (Devlin et al. 2019). It has shown great performance on NLP tasks over the last couple of years (Rogers, Kovaleva, and Rumshisky 2020). BERT is commonly used in deep learning models by attaching different heads to it depending on the objective task. It is trained on two massive corpora; a Google books corpus consisting of 800 million words and the entire English Wikipedia, 2.5 billion words. This is done on Google's own hardware once and can then be used for other applications through *fine-tuning* (Devlin et al. 2019).

3.4.5 Fine-tuning BERT models

Using the massive, trained BERT-model one can use *transfer learning* to fine-tune it towards a specific application (Goodfellow, Bengio, and Courville 2016). This is based on the following idea; if one task is already solved by a model, solving a similar task should not require us starting over from the beginning. One could compare it to this example from everyday life; every time you learn about a new topic, you do not have to re-learn how to read entirely. Instead, you fine-tune your reading ability to understand words and notions related to the new topic you wish to learn about. In that context, it evidently saves you lots of time and effort.

In practice, *fine-tuning* a BERT model usually means that we take the already-trained BERT model, attach a head specific to our application, and continue to train it on on specific dataset related to our task. In essence one could consider this as "clever" initialization of the model weights, where the initialized weights correspond to the best weights from the BERT's original training task. This often results in better model performance after shorter training time.

Transfer learning has been used countless of times in both academia and industry to produce specialized BERT models. These are not seldom named with prefixes relating to the specialization. As an example from financial applications, *FinBERT* is a BERT-based model fine-tuned on classifying the sentiment of a corpus of financial news (Araci 2019).

[4] C.f. for instance *multi-head* attention (Vaswani et al. 2017).

3.4.6 Long-text models

An inherent issue with the original self-attention mechanism is, as mentioned, its quadratic time complexity. This inhibits its use for processing long sequences of text (Beltagy, Peters, and Cohan 2020). Several authors have addressed this problem by inventing modified attention mechanisms with in some contexts preferable complexities. An example of transformer-type leveraging these modified self-attention layers is the *Longformer* (Beltagy, Peters, and Cohan 2020) with complexity $O(L)$ for sequence lengths L. The Longformer uses local windows of a certain length on which the attention operation is performed (Beltagy, Peters, and Cohan 2020). The window length is considered as a hyperparameter.

3.4.7 Open-source models

Fortunately, many of the pre-trained models and weights mentioned above are readily available online under open source license. A central hub for many models and related tooling is *HuggingFace*. Through their products and services it is possible to collaborate, develop, and use machine learning models, occasionally requiring little to no prior programming experience. For instance, one of the models available is FinBERT.

4 Data

In the following we introduce the data used to train and evaluate the two models needed for our analysis. First we present data used to fine-tune a sentiment classifier for long-sequence financial texts, particularly to be used downstream to score on financial article bodies. Following up, we present the data needed to train a regressor used to infer closing prices one hour in the future for the currency pair EUR/USD.

4.1 Data for training a long-sequence sentiment classifier

To train a new sentiment classifier used for scoring article bodies, we require a dataset of sentences with corresponding sentiment labels. Since we specifically want to target financial articles, we use the FinancialPhraseBank corpus for this (see Malo et al. 2014). It consists of some 5000 financial texts that have been labeled as one out of three sentiment classes, namely *negative*, *neutral*, or *positive*.

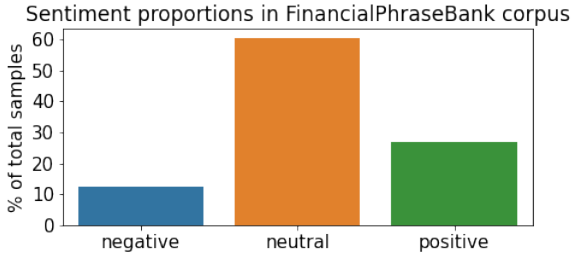

Figure 8 Proportions of each sentiment class for the texts in the FinancialPhraseBank corpus.

The topical content of the data is of general financial nature, but not necessarily related to the foreign exchange.

The dataset is quite unbalanced, see Figure 8. The most common sentiment type in this corpus is *neutral*. This is a crucial observation, and will have an impact on what constitutes a 'good' model later on. As these are a bit shy of 60% of the entire dataset, it means a naïve dummy-classifier always predicting *neutral* would achieve an accuracy of about 60%. Thus our model must be able to at least score better than 60% accuracy to yield an improvement.

To provide some intuition, consider the following example sentence from the corpus: "Nordea Group 's operating profit increased in 2010 by 18 percent year-on-year to 3.64 billion euros and total revenue by 3 percent to 9.33 billion euros.". This has been labeled with a positive sentiment.

4.2 Data for training a predictive Forex model

To be able to train a model predicting future closing prices on the Forex market using technical market information as well as news information, we require historical examples of exactly this. At our disposal we have a dataset coalesced from two sources, as acquired and provided by Anbaee Farimani et al. 2022.

Market data. On the one hand, the dataset consists of market data from September 2018 to May 2021 for the currency pair EUR/USD. Specifically the dataset contains the price measurements *Low*, *High*, *Open*, *Close*, as well as the *Trading Volume* for each hour the market has been open. This results in roughly 16 000 data points. From these five fundamental quantities an additional 12 other technical indicators are created using the *Technical Analysis* Python distribution. We choose the same technical indicators as Anbaee Farimani et al. 2022. Moreover, we follow Anbaee Farimani et al. 2022 and drop the columns *Low*, *High*, and *Open* in our final dataset, creating a total of $\tau = 14$ market-based features.

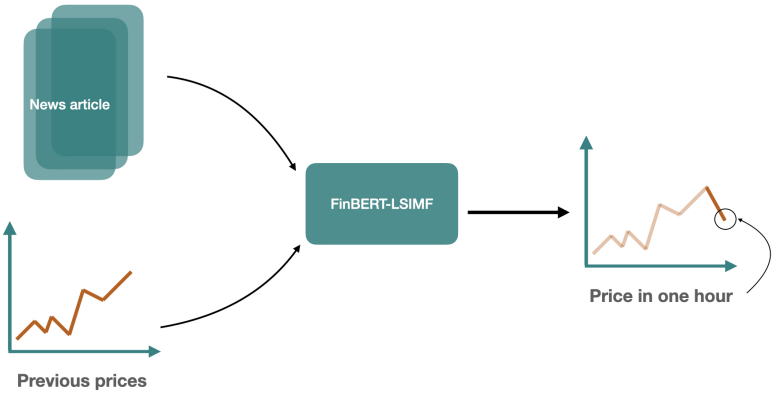

Figure 9 Overview of the inputs and outputs of FinBERT-LSIMF. It takes as input news articles and historical technical indicators, and outputs the future closing price.

News data. On the other hand, the dataset also consists of financial articles published within the same time period as the market data above. These articles have been scraped by Anbaee Farimani et al. 2022 from two financial news websites, namely `www.fxstreet.com` and `www.investing.com`. They consist of a title, an article body, and the time of publishing. In total we have 7 413 news articles. Article statistics can be found in Table 1. An important observation is that the article bodies are significantly longer than the titles. This is something we will account for in the upcoming model building.

Length (characters)	Title	Article Body
Mean	66	1767
Standard Deviation	21	1450

Table 1 Statistics of the news dataset used to train the FX model. The quantities are computed over the 7 413 news articles scraped from the web.

5 Method

We create a deep learning model for predicting the *closing price* `Close` of the Forex pair EUR/USD one hour in the future. As features we use past *technical indicators* from the Forex market, as well as a *mood series* - essentially a sequence of sentiment scores from recent news articles. We will refer to our model as FinBERT-LSIMF, making reference to the fact that it accounts for the sentiment of (long-text) article bodies additionally to the short-text titles. See Figure 9 for a graphical overview of what FinBERT-LSIMF wishes to accomplish.

First, we summarize our approach for extracting sentiment scores from individual articles–both from the titles and bodies. Next, we describe the process of combining these into a mood series, aimed to describe the general expectations of the market participants over time. Lastly, we combine the technical indicators and mood series to construct a dataset - with labels - and display how a predictive Forex model can be built, trained, and evaluated.

5.1 Sentiment analysis for financial article bodies

We first create a deep learning model for extracting sentiment scores from long-sequence financial texts. The model inputs a (possibly long) sequence of text, and outputs a triplet

$$s_{\text{body}} = (P(\text{negative}|\text{body}),\ P(\text{neutral}|\text{body}),\ P(\text{positive}|\text{body})) \in [0,1]^3 \tag{4}$$

containing the conditional probabilities that the input text is of either negative, neutral, or positive sentiment.

Architecture. The model consists of three components. First, the text is input to a Longformer layer, creating an embeddings from the text. The embeddings are subsequently fed into a feedforward layer with 768 neurons and ReLU activations. Lastly, the result is fed through a classification head. The classification head uses softmax activations to transform the output into a conditional probability distribution over the three possible sentiment classes.

To enable faster training on limitied resoures we choose a significantly smaller Longformer model than what is proposed by Beltagy, Peters, and Cohan 2020. We configure the Longformer to have two hidden layers (instead of 12) and two 512 token attention windows. This reduces the size of the model to about 60 million parameters, about a third of the size proposed by the authors.

Training. The model is trained on GPU's using Google Colaboratory. We use the categorical cross entropy loss function and the Adam optimizer, and employ early stopping. Furthermore, we use a learning rate scheduler reducing the learning rate by a factor of 5 if validation loss has not improved every 3 epochs.

Due to limited compute resources we *do not* fine-tune the Longformer layer, and subsequently only train the classification head. This means we end up with about 600,000 trainable parameters.

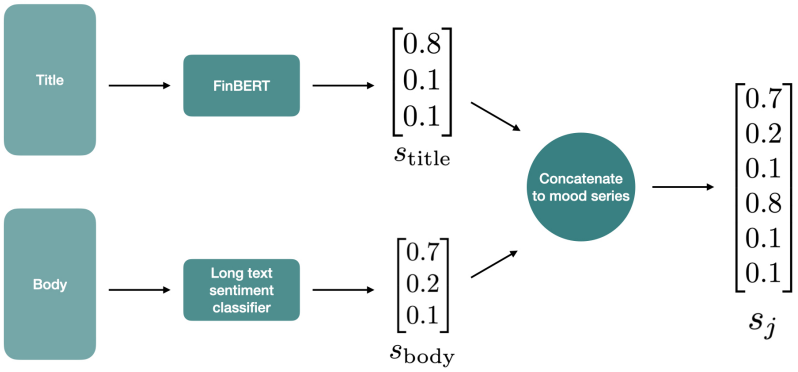

Figure 10 An illustration of how the sentiment values for both title and body are stacked to one vector.

5.2 Sentiment analysis for article titles

Fortunately, for shorter sequences of text we can use FinBERT through the Huggingface inference API in a Python script. Just as for the article bodies, FinBERT outputs for each title a triplet

$$s_{\text{title}} = (P(\text{negative}|\text{title}),\ P(\text{neutral}|\text{title}),\ P(\text{positive}|\text{title})) \in [0,1]^3. \tag{5}$$

5.3 Constructing the mood series

We can now create the mood series: it is a sequence of *hourly* snapshots of the current sentiment of the market. The sentiment is extracted from published news articles. Its purpose is to (hopefully) be an adequate proxy for the real overall expectation of the actors on the EUR/USD market. We begin by stacking the sentiment values from the title and body of each article, creating one vector of sentiment scores $s_j \in [0,1]^6$ for each article. See Figure 10 for an illustrative overview. We then use these to create the mood series.

Formally, the mood series is a sequence $\{S_t\}_{t=1}^N$ of tuples $S_t \in \mathbb{R}_+^6$ of moods, where the first three dimensions are from the title sentiment and the latter three are from the body, as described in sections 5.2 and 5.1, respectively. As the market data is of hourly granularity, the mood series indexation t represents hours as well.

We wish to emphasize that we make a distinction between *sentiment* of individual articles, and the *mood* at different timesteps. The relation between the two quantities is as follows. Ideally, the mood

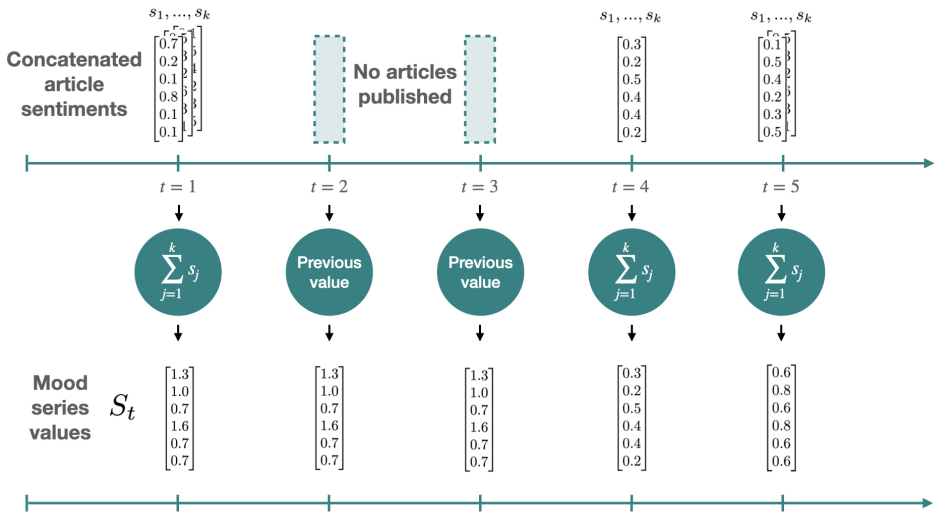

Figure 11 Mood series construction from sentiment scores of individual articles published over time.

S_t would be the aggregated sentiment score for all articles published during the hour up until time t. However, sparsity of the published articles prohibits this if we want to avoid heavy imputation work for the hours without published news. Instead we resort to a backfilling strategy. If the number of published articles during an hour is $k \geq 1$, we define the mood as the sum of the sentiment scores obtained from these k articles. Formally, $S_t = \sum_{j=1}^{k} s_j$ where $s_j = (s_{\text{title}}, s_{\text{body}})$ is the concatenated sentiment score of article j. If no articles are published that hour, we set $S_t = S_{t-1}$, assuming an unchanged mood. In summary, the mood S_t is the sum of sentiments of articles published during the hour before t. If no articles were published, it is the same as the previous hour. See Figure 11 for a graphical depiction of this construction.

5.4 Constructing the features and labels

We start by presenting an intuitive picture of the features we wish to create. These features are subsequently to be fed into our predictive Forex model. The features for our Forex model are, as previously stated, based on both *technical indicators* $\{I_t\}_{t=1}^{N}$ from the market, and the *mood series* $\{S_t\}_{t=1}^{N}$, at different points in time t. The dimensions of each I_t and S_t are $\tau = 14$ and $\sigma = 6$, respectively, as described in sections 4.2, 5.3. Specifically, for each t, we want to use this information from the previous $T \geq 0$ time steps, i.e. from time $t - T$ to t. These will constitute our features. Moreover recall that we wish to predict the `Close` value one hour in the future, i.e. at time step $t + 1$. This is the label we want to create.

We use $T = 7$ in accordance with Anbaee Farimani et al. 2022. Intuitively this can be thought of

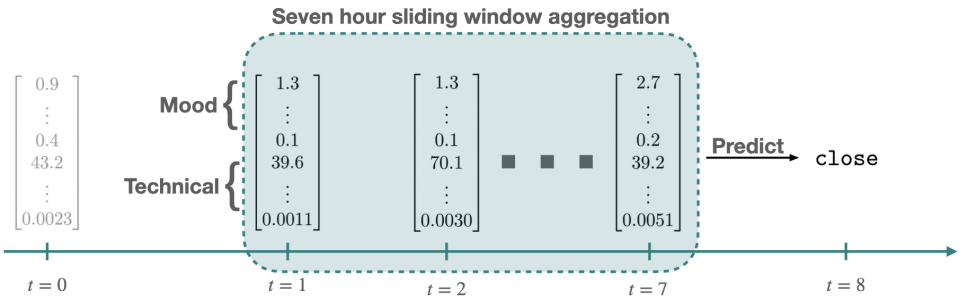

Figure 12 An illustration of how data from the previous $T = 7$ hours are used as input to predict the closing price next hour.

using a 7 hour sliding window from which we use market data and sentiment information to predict the closing price the coming hour. See Figure 12 for a visual representation of the construction of features and labels.

Formally, the construction of the features x_t and labels y_t is as follows. First we normalize the technical indicators $\{I_t\}_t$ to have zero mean and unit variance. Then we concatenate at every time step the normalized technical indicators and mood series into a single vector (I_t, S_t). This gives us in total a $\tau + \sigma = 20$ dimensional numerical representation of the market state at each time step t. A sample x_t is then constructed as the stacking of T such vectors (I_t, S_t), from times $t - T$ to t. We refer to this as a *T-hour sliding window aggregation*. This means a sample x_t is a single $T \times (\tau + \sigma) = 7 \times 20$ matrix. The corresponding label y_t is the closing price at time $t + 1$ i.e. the entry of T_{t+1} corresponding to the closing price. Moreover we can stack again the samples x_t into a third order tensor $X = (x_1, x_2, ..., x_N)^\intercal \in \mathbb{R}^{N \times T \times (\tau+\sigma)}$, and organize the labels into a single vector $y = (y_1, y_2, ..., y_N) \in \mathbb{R}^N$. The tensors X and y are subsequently the objects to be input to the model in the training and evaluation procedure.

As is customary, we partition X and y into three datasets: a training set, a validation set, and a test set. The test set is the last 20% samples of the dataset, and the validation set is 20% of the original data. This is exactly the same partitioning strategy as is used by Anbaee Farimani et al. 2022, meaning this will allow for direct comparison between the performance of our model and their's. The test set covers the time period between October 2020 and May 2021.

5.5 Building and training the predictive Forex model

The goal of our Forex model is to predict y_t using x_t. We do this by creating a parametric model f_θ, and finding a set of (hopefully) optimal parameters $\hat{\theta}$ by minimizing a loss $L(\theta) = \sum_{t \leq N} \ell(y_t, f_\theta(x_t))$ over our entire training dataset.

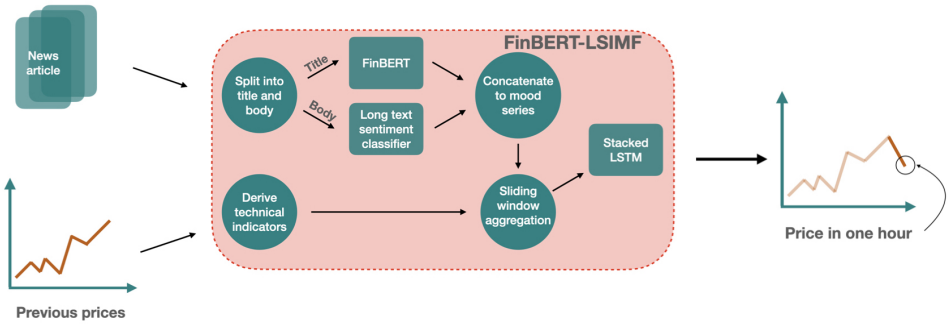

Figure 13 Architecture of the full FinBERT-LSIMF model.

Architecture. We use a deep learning model for f_θ. Our choice of architecture consists of three components; here we describe it in a functional manner. First we feed the input x_t into K-stacked LSTM's with possibly different hidden layer sizes. We consider the output of K'th layer of the last forward pass through the LSTM as the output from this first component. Second, we apply dropout. Lastly, we have a feedforward layer (with one output neuron), outputing a single number. This number is our prediction $\hat{y}_t$. We consider as hyperparameters the number of stacked LSTM layers K, the hidden layer sizes, and the dropout rate. These are tuned to find an optimal model.

We implement the training pipeline and model in TensorFlow. Following Anbaee Farimani et al. 2022 we use as loss function ℓ the Mean Absolute Percentage Error (MAPE) and optimize it using Adam. We employ a learning rate scheduling strategy which reduces the learning rate by a factor of 2 if the validation loss does not improve within 10 epochs, and use early stopping to stop the training if the validation loss is not improving within 80 epochs. Finally, the model's performance is evaluated on the held-out test set.

To summarize, Figure 13 shows a graphical overview of the full pipeline required for processing financial articles and historical market data and predicting the closing price the coming hour. Additionally, we will train an identical model on data not containing article body sentiments (i.e. FinBERT-SIMF), for better comparison with FinBERT-LSIMF. This will allow for a evaluation of the additional information we can extract from the article bodies - information that may not be present in the titles alone.

6 Results

First we present the performance of the sentiment analysis model train for handling long-sequence text. Following up, we show the performance of the Forex model when predicting closing prices

	Cross-entropy Loss	Accuracy
Training set	0.72	0.69
Validation set	0.67	0.72
Test set	0.67	0.72
Dummy classifier	-	0.61

Table 2 Performance metrics for the Longformer-based text classifer on the FinancialPhraseBank corpus. For reference, the performance of a dummy classifier (acting as a baseline) is included.

one hour in the future. Lastly we sanity-check the model by conducting a qualitative error study where we manually inspect the model output for a set of article inputs.

6.1 Performance of the long-text sentiment model

Table 2 shows the performance of the best performing long-text sentiment classifier on the validation set and test set, respectively. We see that in slightly less than three cases out of four, our model is able to correctly identify the sentiment of never-before seen financial text. A second observation is that the accuracy on the test set is greater than that of a potential dummy-model always predicting *neutral* (which would yield 60% accuracy). This means the model is better than (judicious) guessing of sentiment type by about 18%. Lastly, the performance metrics on the training set compared to the validation or test are quite similar. This means we are *not* experiencing heavy overfitting. Rather it might be the opposite: this could be an hint of that the model could potentially be improved for instance by further training.

6.2 Performance of the predictive Forex model

The best FinBERT-LSIMF model we find uses the following hyperparameters: we use $K = 2$ hidden LSTM layers, both containing 128 neurons, and a dropout of 0.5; a learning rate of 1e-3, weight decay of 1e-6, batch size of 32, and train it for 150 epochs. We use the same hyperparameters for building our FinBERT-SIMF model which is trained by ourselves.

Table 3 shows the performance of the best performing Forex models we could find by tuning the hyperparameters. The table contains both the performance of a model which additionally uses article body sentiments, namely FinBERT-LSIMF, and one that only uses article title sentiments, namely FinBERT-SIMF . Additionally, the table contains the performance of two models trained by Anbaee Farimani et al. 2022: the original FinBERT-SIMF model and also the model FinBERT-IMF

	Validation Loss	Test Loss
FinBERT-LSIMF	0.102	0.161
FinBERT-SIMF *(our version)*	0.124	0.399
FinBERT-SIMF (Anbaee Farimani et al. 2022)	0.121	0.291
FinBERT-IMF (Anbaee Farimani et al. 2022)	3.623	6.695

Table 3 Comparison of Mean Absolute Percentage Error loss between different models on the validation and test sets. FinBERT-LSIMF uses sentiment from article titles and bodies; FinBERT-SIMF uses sentiment only from article titles. Both the score for FinBERT-SIMF trained by us, and the score obtained by Anbaee Farimani et al. 2022, is included. Additionally we include the performance of a FinBERT-IMF model leveraging only technical indicators.

[*sic*] which only accounts for technical market features and no sentiment information. Note that the name FinBERT-IMF is a misnomer as it is does not leverage FinBERT.

The validation loss is quite similar for the three models leveraging both sentiment and technical indicators. However, the test loss for FinBERT-LSIMF is smaller than for the other two models not using article body sentiments. Particularly, FinBERT-LSIMF has 44% lower loss than FinBERT-SIMF (trained by Anbaee Farimani et al. 2022) which does not leverage article body moods. This indicates that there is significant information in the article bodies that are not present in the titles alone. This indicates that our model is able to capture some correlation between the sentiment of the article bodies and the closing price of the currency pair EUR/USD in the near future. The fact that the test loss is lower means that the model generalizes better to data not seen during training, and has a smaller error when predicting the future closing price.

Moreover, we note that the loss is significantly lower for the models leveraging sentiments; FinBERT-IMF incurs the greatest loss. Hence, there seem to be a significant improvement to be found by incorporating article sentiments in the model. Lastly, we note that the FinBERT-SIMF model trained by us has larger error than the one trained by Anbaee Farimani et al. 2022. This is likely be because we are not using identical hyperparameters.

6.3 Qualitative error study

Figure 14 displays the predicted and real closing prices of the EUR/USD pair on the test set. In general, the predicted values seem to align quite well with the real values. This is at least not in disagreement with the observed mean average percentage error of 0.16% obtained by the model. The model seems to follow the general trend in the data, but in some time intervals the model seems to be quite off (for instance around December 2020). Additionally, for instance in early November

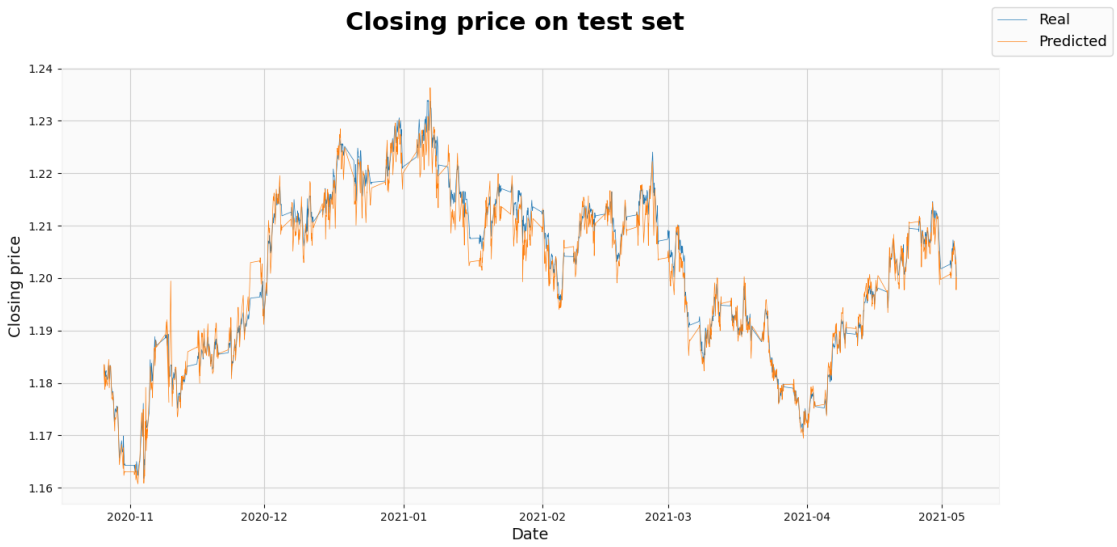

Figure 14 Real and predicted closing prices on the test set. The test set covers the period from around mid-October 2020 to early May 2021.

2020, certain spikes in the data are amplified in the prediction versus the real value.

Next, we visualize the degree to which the short-text and long-text sentiment classifiers align on the test set. Figure 15 displays the inferred mood series on a subset of articles from the test set. The example mood series displayed covers a time span of 75 hours starting from October 25th 2020. By inspection, it seems the sentiments inferred from titles and bodies agree quite well on this particular subset. Generally the sentiment peaks are in the same direction for both types of sentiments, at least for the positive and negative mood series. We however note a few anomalies in the following. First, the negative sentiments seems to be stronger in the titles than in the bodies. Second, on this specific subset of data, positive sentiments are generally inferred as higher when looking at article bodies compared to the titles.

Lastly, we sanity check some of the prominent peaks in the mood series in Figure 15 against articles used as input for the inference. Figure 16 displays a zoom-in on three prominent peaks; 16(a) and 16(b) displays peaks in negative sentiment, and 16(c) in positive sentiment. In both negative sentiment examples, the article titles sentiment indeed seem to be inferred reasonably. For the positive example in 16(c), the connection between the article title and positive sentiment is not as clear.

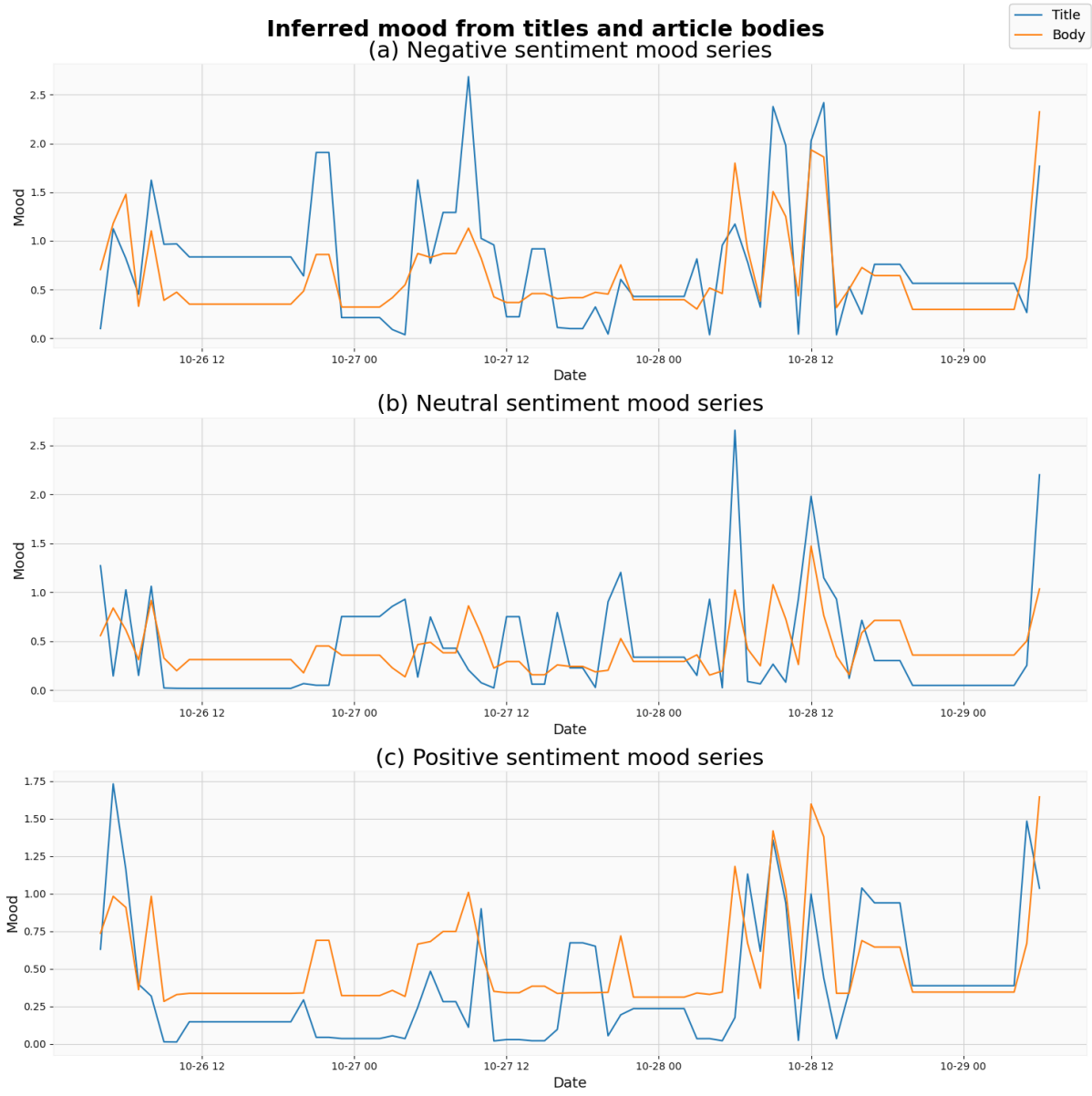

Figure 15 Comparison between inferred mood from article titles and bodies. The comparison covers only a subset of the test set: a 75 hour period starting from October 25th 2020.

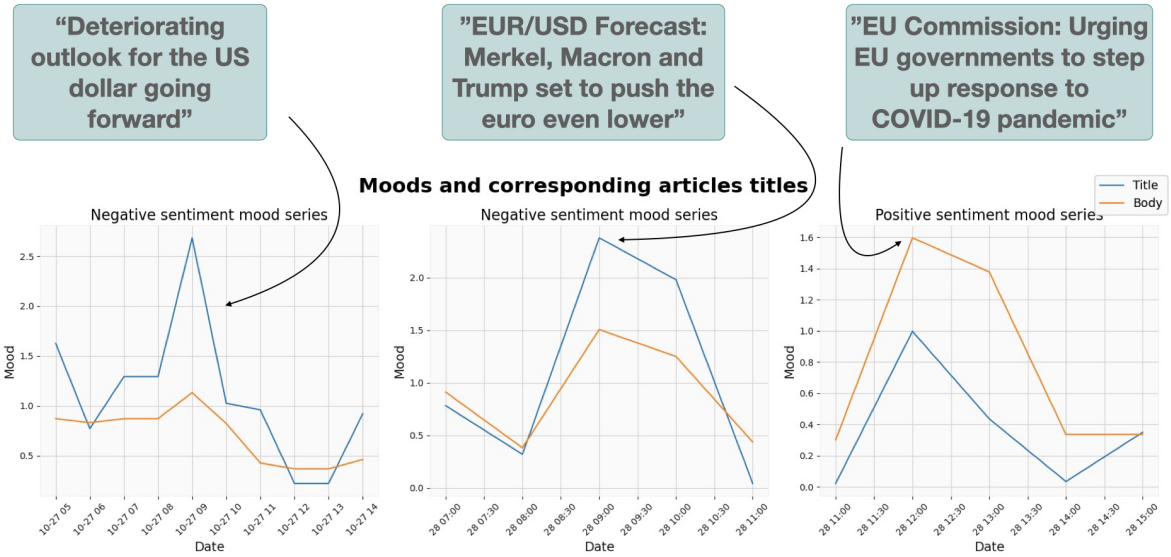

Figure 16 Examples of peaks in constructed mood series with accompanied article titles from that hour. The examples are fetched from the test dataset. Subfigures (a) and (b) display peaks in the negative mood series; (c) displays a peak in the positive mood series.

7 Discussion

What we aim to discuss is both whether we can draw a conclusion regarding the validity of the Efficient Market Hypothesis, and the validity of the experiment in its entirety. In section 7.1 we discuss the model performance and particularly the significance of the results obtained. Section 7.2 covers the issue of using sentiment analysis on articles related to currency pairs. In section 7.3 we discuss potential general issues with the future usage of FinBERT-LSIMF, particularly from a performance, explainability, and fairness perspective. Section 7.4 discusses the validity of the Efficient Market Hypothesis in the light of our experiment. Lastly, section 7.5 provides potential next steps to solve the issues addressed in the upcoming sections.

7.1 Model performance

First, we discuss the significance of the quantitative results from our experiment. Then, we discuss the findings from the qualitative error study.

7.1.1 Significance of quantitative result

According to the performance on the held-out test set, our FinBERT-LSIMF outperforms the models where article body has been left out, trained by both us and Anbaee Farimani et al. 2022. Additionally, it outperforms the model not using sentiments at all. This could indicate that there is additional information to be found in financial articles for predicting the future closing price on the Forex market. However, it is not entirely clear whether we can draw any conclusions from this improvement due to two main reasons:

1. the model has only shown to perform better on a single test set;
2. it is not certain that the higher performance can be attributed to the incorporation of article bodies.

We discuss these two points in the following.

First, the test data is quite limited. This has two main implications: (1) an improvement on one test set does not mean the model is better in every context and situation, and (2) the performance estimates might be untrustworthy. In contrast to the validation loss which is computed using cross validation (i.e. computed as the average performance on multiple validation sets), the test loss is computed only once on a single test set. This induces some variance in the test loss estimate. Since we have not quantified this variance, it is unclear whether the improvement could have been obtained simply by pure chance.

Second, it is not entirely clear to what extent the improvement of FinBERT-LSIMF versus FinBERT-SIMF is caused by the augmentation of the model with article body sentiments. As our long-text sentiment classifier only beats the performance of the dummy classifier by about 18%, the improvement is not enormous. We have not tested what effect a marginal improvement of the underlying long-text sentiment classifiers has on the final predictive Forex model. That would provide a better indication of the extent to which the article bodies contribute to the final predictive Forex model's performance.

7.1.2 Error study and sentiment sanity check

Firstly, it is reassuring that the plotted predicted closing prices overall seem to align well with the real values–as indicated by the test MAPE loss of 0.16%. However, it is difficult to draw additional

conclusions about the model validity using this information, than those already obtained from the quantitative results.

Second, when inspecting the subset of the test set mood series, there is mostly agreement between the sentiment inferred from article titles and bodies. However, the inferred negative sentiment in the titles is generally greater than that from the bodies. At least this holds on this specific subset of the test set. A possible explanation is that the titles are more alarmistic than the bodies. A news article title is often meant to catch the readers attention, resulting in a more hyperbolic language. If that would hold true, adding sentiments from article bodies might have a regularizing effect which reduces the polarity of the over-all sentiment classification.

Lastly, the inspection of random examples of articles and inferred mood seem to show that they are in general quite reasonable. However, the example of an article where the inferred positive mood value is large (in Figure 16(c)) is not as clearly related to the title contents. This points to the importance of carefully interpreting the mood series: although the sentiment of an individual article is not strongly positive, the overall (aggregated) mood might still be highly positive. This is due to the fact that the mood is the accumulated sentiment from articles published within that hour. Hence, the mood is generally stronger for all sentiment categories when a greater number of articles are published. If we consider the mood values at this point in time (12:00, October 28) in Figure 16 we see that the negative and neutral values are in fact higher in absolute terms compared to the positive mood. This was intentionally incorporated into the model as to make it account for the intensity of the news flux in its predictions; an intense news flux could be an important signal to capture in the model.

To summarize, the qualitative error study shows that the model is generally behaving as expected but it would have to be conducted on a larger scale to be directly useful for drawing conclusions.

7.2 Positive according to whom?

One aspect that is not incorporated in our mood series model is that the sentiment of a text has to be in favour of someone. This is strongly tied to the fact that we are dealing with currency *pairs*, and not single assets. When it comes to the sentiment of a stock or index there is a quite clear interpretation: positive probably means that the overall expectation is rising. If the sentiment is positive regarding Nasdaq stock exchange, the stock prices traded will probably rise. However, when it comes to a foreign exchange rate, an appreciation of one currency in the pair is equivalent to a depreciation of the other. Despite this, it is regarded as positive. For instance, as seen in the qualitative error study, the sentence "Deteriorating outlook for the US dollar going forward" is

indeed negative-sounding from the USD perspective. However, from the EUR point of view this would instead be deemed positive (in the EUR/USD currency pair context). This is one clear flaw of blindly applying financial sentiment analysis in our context.

7.3 Potential issues with the usage of FinBERT-LSIMF

Although our model seems to perform well now, it is not an assurance of its performance in the future. For instance, as the model has only been trained and evaluated on historical data, it would not necessarily perform well on real up-to-date data. To enable this, the model would have to be trained on labeled up-to-date data. When that has been done, a re-evaluation of the model would have to be conducted to see the severity of the potential drift of the underlying data distribution. Additionally, recall that the regressor-part of the model has only been trained on data for the currency pair EUR/USD. This means we have no understanding of the model's ability to generalize well on other currency pairs; this would likely also require new training data and a re-evaluation of the model.

There is also the question of explainability of the model. Currently, it is difficult to decipher what specific parts of the input that lead to the model making a specific output. This is especially true as the full FinBERT-LSIMF in itself consists of three models–two sentiment classifiers and an LSTM regressor–each of which is difficult to interpret due to their complexities. For instance, what specific parts of the article bodies that the long-text sentiment classifier most heavily bases its sentiment score on is not evident. The same holds for the two other models as well. The implication of this is that the model might have limited use in fields where model interpretability and explainability is crucial, unless additional appropriate tools are used.

Lastly is the question of bias and fairness. In our experiment setting, we have not taken into account the risk of the sentiment data sources being biased, includes controversial topics, or lead to the model making discriminatory decisions. Fixing this would partly require a thorough investigation in the relationship between the inputs and the labels of the datasets used, and partly require taking appropriate actions to counteract this potential unfairness. A proposal for an intervention to reduce model bias could be to exchange all instances of words that could induce unfairness with generalized dummy-tokens. For instance, all words related to gender (e.g. "man" or "woman") could be swapped for a general token `<gender_token>`, as to intentionally make the model not make a distinction between these words. A similar approach could be used for other types of words that could risk inducing discriminatory behaviour in the model. For this specific task it would likely be a good idea to make sure that the model is not unintentionally biased towards certain geographical areas and currencies.

7.4 Validity of the Efficient Market Hypothesis

In order to discuss the validity of the Efficient Market Hypothesis in the light of our experiment, we need to contextualise our model and its predictions. One way of using our findings is to assess the predictive power of the model. We know that it can predict the future Forex rate quite well, but we need know what success looks like and what level of performance is sufficient for "beating the market". Another way of discussing the EMH is by looking at the marginal improvement of the model when incorporating more information. We discuss these two aspects in the following.

The simplest form would be to show that one can make risk adjusted returns from the Forex market using our model. Since our predictor incorporates historical prices and publicly available information it tests the EMH in its semi-strong form. If such a model were to beat the market, one could argue that this specific market under the period is not efficient in the semi-strong form. This would be further strengthened by the fact that when including more information, i.e. the article bodies, the performance is further enhanced. However, testing this is beyond the scope of this paper.

A less direct way of discussing the EMH in the light of our experiment is by comparing our two predictors and FinBERT-IMF from Anbaee Farimani et al. 2022. The performance of these, found in Table 3, show that performance increases with the amount of information utilised. Since FinBERT-IMF uses only on previous prices, it accounts for all information that would be available in the weak-form setting of the EMH. FinBERT-IMF is however outperformed on the test set by FinBERT-LSIMF: a model that is essentially the same, but with the news sentiment as additional input. There are thus implications that moving from a weak form to a semi-strong form information setting enhances predictions. If the semi-strong EMH form holds, both predictions should be similarly inaccurate, but this is not the case. However, this conclusion requires us to be certain that this performance leap is entirely assigned to the addition of news sentiment in the model. As outline in section 7.1, there are too many uncertainties in the results to make this claim.

7.5 Next steps

We would here like to state a few ideas for future experiments and enhancements of this model. We begin by giving examples of how to more confidently accredit the performance improvement to the addition of article bodies in the model. We then describe a way of testing the models ability to generate risk adjusted returns. As a final idea, we describe how to make sense of the need for intentional bias for the model to work properly.

To address the issue of the uncertainty of the improvement of our the predictive Forex model when

incorporating article bodies, we propose the following two interventions. First, it would be beneficial to collect more test data. This would both give a better idea of the model's performance when used in more contexts, and also give a more low-variance performance metric. Additionally it would give a more elaborate view of the model's ability to generalize well on new examples previously not seen. However, due to the difficulty in obtaining clean data, this was not done in this experiment. Second, the long-text sentiment classifier should be improved. This would give a better understanding of the extent to which the performance improvement can be attributed to the addition of article bodies in the input data. One could then inspect the marginal improvement obtained from this in the final predictive Forex model. If the Forex model strictly improves when the sentiment classifier becomes more accurate, it would indicate that the improvement is indeed coming from the incorporation of article bodies.

To improve the long-text sentiment classifier, we propose the following solutions: (1) collect more labeled sentiment data, and (2) continue to train the long-text sentiment classifier. In this case, it would likely be beneficial to focus on collecting long text sequences (i.e. longer than the average length of 1767 words as in our training set), as this is ultimately what we want the model to be good at analysing. Additionally, collecting a more diverse dataset with respect to labels would likely be useful. Recall that currently about 60% of the texts are labeled neutral. Providing the model with more examples of high-polarity texts (i.e. with non-neutral sentiment) would likely enable it to better capture the sentiment signals in the dataset with financial news. Moreover, the performance could likely also be improved simply by using more computational resources, as it would allow for fine-tuning the Longformer layer of the long-text sentiment classifier.

To test the model's ability to beat the market, as was discussed in section 7.4, one could construct a trading strategy based on the model predictions and run a simulation on historical data. If this is capable of systematically making positive returns it would indicate the all of the information from the news articles are not yet incorporated in the Forex rates.

Lastly, we propose a solution for the issue described in section 7.2 about sentiment and its usefulness in understanding the expectations on the currency market. Incorporating in the training data an intentional bias for a certain currency in the currency pair in question is likely crucial for improving the model further. This would amount to collecting a new dataset and label it manually according to some specification stating a preference for one of the currencies. Then, a new sentiment classifier could be trained on this data as to incorporate this intentional currency bias into the full FinBERT-LSIMF model.

8 Conclusion

In this work, we build and evaluate a model used for predicting future prices of the EUR/USD currency pair using historical prices and mood extracted from financial articles. The mood of the market is extracted from articles using two transformer-based sentiment classifiers. We leverage both the existing FinBERT classifier, and train a new Longformer-based sentiment model (fine-tuned on financial articles) used for processing longer sequences of texts. The final predictive Forex model is a stacked LSTM model which processes the mood and technical indicators over time to predict the closing price in one hour. Our model finds a 44% improvement in mean absolute percentage error compared to a model not leveraging sentiment from article bodies. Additionally, our model is much more accurate than a model relying only on historical prices. This indicates that article bodies contain information that enhances the accuracy when predicting future Forex prices. However, there are several uncertainties in the results and the way in which one quantifies model "goodness". Hence, we cannot prove whether the Forex market for EUR/USD between October 2020 and May 2021 was an efficient market in the semi-strong form. We propose next steps for improving the certainty of the results, for instance by testing the model in a wider array of settings and environments.

Bibliography

Adam, Klaus and Stefan Nagel (2022). *Expectations Data in Asset Pricing*. Tech. rep. National Bureau of Economic Research.

Agarap, Abien Fred (2018). *Deep Learning using Rectified Linear Units (ReLU)*. DOI: 10.48550/ARXIV.1803.08375. URL: https://arxiv.org/abs/1803.08375.

Aggarwal, Charu C.) (2022). *Machine Learning for Text*. 2nd ed. 2022. Cham : Springer International Publishing : URL: http://ludwig.lub.lu.se/login?url=https://doi.org/10.1007/978-3-030-96623-2.

Alamatian, Zohreh, Majid Vafaei Jahan, and Amin Milani Fard (2021). "Using Market Indicators to Eliminate Local Trends for Financial Time Series Cross-Correlation Analysis". In: *Proceedings of the Canadian Conference on Artificial Intelligence*. https://caiac.pubpub.org/pub/jtymmz06.

Anbaee Farimani, Saeede et al. (2022). "Investigating the informativeness of technical indicators and news sentiment in financial market price prediction". In: *Knowledge-Based Systems* 247, p. 108742. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2022.108742. URL: https://www.sciencedirect.com/science/article/pii/S095070512200346X.

Antweiler, Werner and Murray Z. Frank (2004). "Is All That Talk Just Noise? The Information Content of Internet Stock Message Boards". In: *The Journal of Finance* 59.3, pp. 1259–1294. DOI: https://doi.org/10.1111/j.1540-6261.2004.00662.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1540-6261.2004.00662.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1540-6261.2004.00662.x.

Araci, Dogu (2019). "FinBERT: Financial Sentiment Analysis with Pre-trained Language Models". In: *CoRR* abs/1908.10063. arXiv: 1908.10063. URL: http://arxiv.org/abs/1908.10063.

Beltagy, Iz, Matthew E. Peters, and Arman Cohan (2020). "Longformer: The Long-Document Transformer". In: *CoRR* abs/2004.05150. arXiv: 2004.05150. URL: https://arxiv.org/abs/2004.05150.

Bollen, Johan, Huina Mao, and Xiaojun Zeng (2011). "Twitter mood predicts the stock market". In: *Journal of Computational Science* 2.1, pp. 1–8. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2010.12.007. URL: https://www.sciencedirect.com/science/article/pii/S187775031100007X.

Chowdhary, K.R. (2020). *Fundamentals of Artificial Intelligence*. Springer India. ISBN: 9788132239727. URL: https://books.google.se/books?id=8SfbDwAAQBAJ.

Daudert, Tobias (2021). "Exploiting textual and relationship information for fine-grained financial sentiment analysis". In: *Knowledge-Based Systems* 230, p. 107389. ISSN: 0950-7051. DOI: https://doi.org/10.1016/j.knosys.2021.107389. URL: https://www.sciencedirect.com/science/article/pii/S0950705121006511.

Davies, Peter Lloyd and Michael Canes (1978). "Stock Prices and the Publication of Second-Hand Information". In: *The Journal of Business* 51.1, pp. 43–56. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2352617 (visited on 01/18/2023).

Deboeck, Guido J. (1994). *Trading on the Edge: Neural, Genetic, and Fuzzy Systems for Chaotic Financial Markets*. Wiley.

Devlin, Jacob et al. (June 2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423.

Fama, Eugene F. (1970). "Efficient Capital Markets: A Review of Theory and Empirical Work". In: *The Journal of Finance* 25.2, pp. 383–417. ISSN: 00221082, 15406261. URL: http://www.jstor.org/stable/2325486 (visited on 01/18/2023).

— (1965). "The Behavior of Stock-Market Prices". In: *The Journal of Business* 38.1, pp. 34–105. ISSN: 00219398, 15375374. URL: http://www.jstor.org/stable/2350752 (visited on 01/18/2023).

fdeloche (2017). *Recurrent neural network unfold.svg*. Accessed: 20 Jan 2023. URL: https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg.

Ferguson, Nicky J. et al. (2015). "Media Content and Stock Returns: The Predictive Power of Press". In: *Multinational Finance Journal* 19.1, pp. 1–31. URL: https://EconPapers.repec.org/RePEc:mfj:journl:v:19:y:2015:i:1:p:1-31.

Gohel, Prashant, Priyanka Singh, and Manoranjan Mohanty (2021). "Explainable AI: current status and future directions". In: *CoRR* abs/2107.07045. arXiv: 2107.07045. URL: https://arxiv.org/abs/2107.07045.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

Kalyanathaya, Krishna Prakash, D Akila, and P Rajesh (2019). "Advances in natural language processing–a survey of current research trends, development tools and industry applications". In: *International Journal of Recent Technology and Engineering* 7.5C, pp. 199–202.

Kim, Chiheon et al. (2021). "Automated Learning Rate Scheduler for Large-batch Training". In: *CoRR* abs/2107.05855. arXiv: 2107.05855. URL: https://arxiv.org/abs/2107.05855.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. URL: http://arxiv.org/abs/1412.6980.

Kirkpatrick II II, Charles D. and Julie R Dahlquist (2010). *Technical Analysis: The Complete Resource for Financial Market Technicians*. FT Press.

Malo, Pekka et al. (Apr. 2014). "Good Debt or Bad Debt: Detecting Semantic Orientations in Economic Texts". In: *Journal of the American Society for Information Science and Technology*. DOI: 10.1002/asi.23062.

MartinThoma (2014). *Perceptron-unit.svg*. Accessed: 20 Jan 2023. URL: https://commons.wikimedia.org/wiki/File:Perceptron-unit.svg.

Mikolov, Tomas et al. (2013). *Efficient Estimation of Word Representations in Vector Space*. DOI: 10.48550/ARXIV.1301.3781. URL: https://arxiv.org/abs/1301.3781.

Nisar, Tahir M. and Man Yeung (2018). "Twitter as a tool for forecasting stock market movements: A short-window event study". In: *The Journal of Finance and Data Science* 4.2, pp. 101–119. ISSN: 2405-9188. DOI: https://doi.org/10.1016/j.jfds.2017.11.002. URL: https://www.sciencedirect.com/science/article/pii/S2405918817300247.

O'Hare, Neil et al. (2009). "Topic-Dependent Sentiment Analysis of Financial Blogs". In: *Proceedings of the 1st International CIKM Workshop on Topic-Sentiment Analysis for Mass Opinion*. TSA '09. Hong Kong, China: Association for Computing Machinery, 9–16. ISBN: 9781605588056. DOI: 10.1145/1651461.1651464. URL: https://doi.org/10.1145/1651461.1651464.

Otter, Daniel W., Julian R. Medina, and Jugal K. Kalita (2018). "A Survey of the Usages of Deep Learning in Natural Language Processing". In: *CoRR* abs/1807.10854. arXiv: 1807.10854. URL: http://arxiv.org/abs/1807.10854.

Patel, Pareshkumar J. et al. (2014). "Factors affecting Currency Exchange Rate, Economical Formulas and Prediction Models". In.

Rogers, Anna, Olga Kovaleva, and Anna Rumshisky (2020). "A Primer in BERTology: What We Know About How BERT Works". In: *Transactions of the Association for Computational Linguistics* 8, pp. 842–866. DOI: 10.1162/tacl_a_00349. URL: https://aclanthology.org/2020.tacl-1.54.

Seifollahi, Saeed and Mehdi Shajari (Feb. 2019). "Word sense disambiguation application in sentiment analysis of news headlines: an applied approach to FOREX market prediction". In: *Journal of Intelligent Information Systems* 52. DOI: 10.1007/s10844-018-0504-9.

Sherstinsky, Alex (2020). "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404, p. 132306. ISSN: 0167-2789. DOI: https://doi.org/10.1016/j.physd.2019.132306. URL: https://www.sciencedirect.com/science/article/pii/S0167278919305974.

Sidehabi, Sitti Wetenriajeng, Indrabayu, and Sofyan Tandungan (2016). "Statistical and Machine Learning approach in forex prediction based on empirical data". In: *2016 International Conference on Computational Intelligence and Cybernetics*, pp. 63–68. DOI: 10.1109/CyberneticsCom.2016.7892568.

Sra, Suvrit, Sebastian Nowozin, and Stephen J. Wright (2011). *Optimization for Machine Learning*. The MIT Press. ISBN: 026201646X.

Srivastava, Nitish et al. (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1, pp. 1929–1958.

Stetco, Adrian et al. (2020). *Neural Network.png*. Accessed: 20 Jan 2023. URL: https://commons.wikimedia.org/wiki/File:Neural_network.png.

Tetlock, Paul C. (2007). "Giving Content to Investor Sentiment: The Role of Media in the Stock Market". In: *Journal of Finance* 62.3, pp. 1139–1168. URL: https://EconPapers.repec.org/RePEc:bla:jfinan:v:62:y:2007:i:3:p:1139-1168.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

Wolf, Thomas et al. (Oct. 2020). "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6.

Zaccone, Giancarlo, Md. Rezaul Karim, and Ahmed Menshawy (2017). *Deep Learning with TensorFlow*. Packt Publishing. ISBN: 9781786469786.

Zhang, Zhilu and Mert Sabuncu (2018). "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2018/file/f2925f97bc13ad2852a7a551802feea0-Paper.pdf.