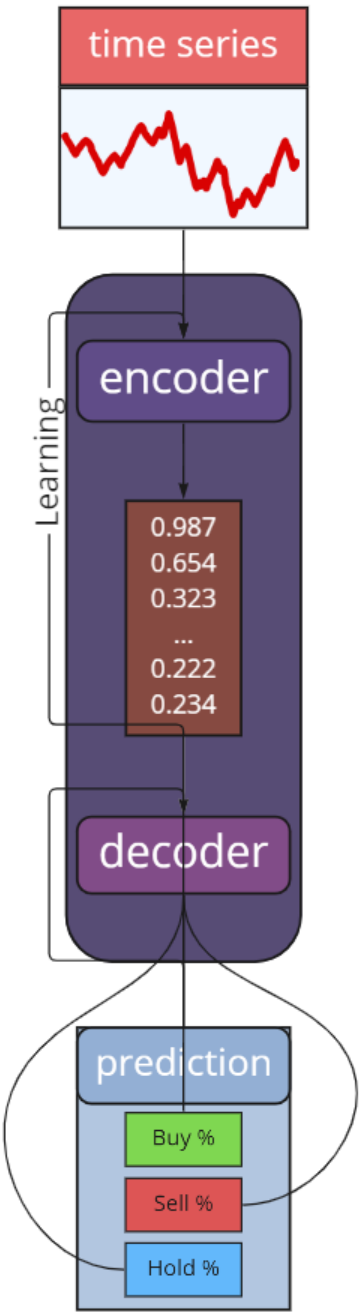
A Novel Approach to Financial Forecasting

Predicting Bitcoin Prices By Applying Long-Short Term Memory Networks and Supervised Contrastive Learning To Technical Indicators

Jiaqi Chen, Thomas Hickey, Henk van der Meijden
Utrecht University



Utrecht University

Abstract

We attempted to predict the movements of the price of Bitcoin using machine learning methods. We did not attempt to make a numerical prediction of the price but rather attempted to predict whether the price will go up, down, hold steady or otherwise have too much uncertainty regarding its movement. For this we encoded labels with associated classes *buy*, *sell* or *hold* respectively. This was done through application of machine-driven technical analysis, particularly using a long-short term memory neural network to recognize (repeating) trends over time. As a way to combat the information loss in reducing a large latent space to three classes we applied contrastive learning, which splits up the classifying network into an encoder and decoder, with separate loss functions. The encoder uses a contrastive loss to cluster similar classes, and the decoder gives probabilities based on the clustering. Our results show that further investigation is necessary to determine the effective application of these methods for the price prediction of Bitcoin.

Introduction

The report starts by considering the design decisions we made from a more global level, many of which were already in place early on, before the first line of code was written. It is followed by a more in-depth look at the specific implementation in our code of these design decisions. Lastly, the discussion section where we consider the multiple improvements to our model we made learning on the go as well as possible (future) improvements that were not implemented.

Why Bitcoin?

Our choice to approach Bitcoin as opposed to any other (more traditional) currency or stock is not necessarily on account of it (or most other cryptocurrencies) being fundamentally different from the point of price prediction. The motivation is that it is a relatively new and novel currency, which still has over 10 years of price history. It is not yet as consolidated as more traditional assets, which is one of the reasons why the market is so volatile. We expect this volatility to increase with decreased market capitalization and vice versa, thus conceding the possibility of it becoming more consolidated than existing traditional currencies.

A difference caused by the current (comparatively) lower consolidation of Bitcoin is that if a model has profitable performance then it would be able to leverage the larger movements inherent in a less consolidated currency for more profit. We believe that the purely speculative cryptocurrency market makes technical indicators more reliable than they would be for assets with intrinsic value (as fundamental analysis would be another axis along which price predictions can take place). In addition, Bitcoin is decentralized and as such there is no single authority (in theory) that is able to alter these prices (in unpredictable ways). Such that, theoretically, the price should be more predictable. Obviously, in practice, a lower market capitalization (~€390 million at time of writing) and volume (~€25 million over the past 24 hours at time of writing) allows for the ability of a single (wealthy) entity to greatly alter its price trajectory. (Although they would have a hard time actually accumulating the currency, due to its distribution and lack of centralized supply). This decentralization also means that there is no single authority there to prevent market manipulation.

With Bitcoin being (on its inception) one of a kind also makes it an interesting object of study in its own right. This was also the case for us and this being the case has also helped us at the start, when orienting ourselves we had access to multiple papers on exactly the topic of Bitcoin (or other cryptocurrency) price prediction[1]. Bitcoin is the obvious pick among these as it is the oldest and thus has the most amount of available data.

Why Artificial Neural Networks?

There are many methods of classifying timeseries. Our considerations for applying artificial neural networks (ANNs), are for its ability to generalize given enough training examples, despite being more difficult to train, compared to methods like decision trees. ANNs are one

[1] Table 1 from (Sebastião & Godinho, 2021) provides an extensive overview of past studies on the application of machine learning methods for cryptocurrency price prediction.

of many supervised learning algorithms, which are a good fit if you have some way of rendering the desired outcome. This is certainly the case with price prediction as our data can already return what we want it to predict simply by looking ahead. An ANN is also able to handle input across multiple dimensions and does so in a non-deterministic way. This is also helpful as we have no reason to suspect that this problem can be solved in a linear or deterministic manner. We can expect such a relatively simple function to answer such a big question to already have been found if that were the case. This complexity ties into the necessity of non-deterministic outputs as we expect there to be times where different data is contradicting each other, where some part(s) might also be expected to weigh more heavily than others, thus resulting in an output that is non-deterministic.

Why Long-Short Term Memory Networks?

We believe a Long-short Term Memory (LSTM) network to be an excellent approach that resolves many of the theoretical problems one runs into when trying to do financial forecasting. The initial motivation is that we all already have some practical and theoretical knowledge of artificial neural networks, ANNs, make up a larger class of networks that LSTMs are a part of. First and foremost is the necessity that all financial forecasting relies on knowledge of the past (and recognizing trends therein). Basic ANNs, categorically, are unable to do this as they only consider single inputs in isolation. As a response to this recurrent neural networks (RNNs) were developed. RNNs solve this by making the output of a node function as an input on the next layer, but RNNs have their own problems with sufficiently large sequences or deep networks; in these cases information gets lost. This happens because learning information gets passed through the system through multiplication and with a long enough sequence of multiplications any value above 1 will approach infinity (and lose informative value) and any value below 1 will approach 0 (and lose informative value).
The LSTM networks were developed as a response to this shortcoming and others from RNNs, this is why we choose it. LSTM network work by taking sequences of inputs and learning from them, being able to remember salient information from the sequence and forget irrelevant information from the sequence all the while taking into account relevant information of the present.
Our choice of LSTM networks has not been made following the consideration of every possibility. It might very well be that a different approach could prove more effective[2]. It is a fact that the artificial intelligence field, especially in recent times, develops at rapid speeds such that it is an impossibility to be (or stay) fully up to date. This is reflected in our teaching and subsequently in our own knowledge, e.g. we were not taught anything regarding transformers and as such had no (existing) theoretical grounding if we were to attempt to implement a transformer for this project. Thus it might be the case that there are

[2] We ourselves had already discussed the idea to implement a transformer for our predictions. Transformers are chronologically the next step to LSTMs, analogous to the step from LSTMs to RNNs. The big benefit for transformers is the ability to take in a whole sequence at once. Currently the LSTM reads the input data, row by row. Using transformers cuts training time and improves performance over longer term patterns. We might have done so in addition to our LSTM network time permitted. Transformers are behind the highly impressive GPT-3 language model and are able to account for many of the problems one runs into that LSTM networks also attempt to resolve.

shortcomings within LSTM networks (that we are not aware of) and these could even be shortcomings that have been resolved by an existing algorithm (that we are not aware of).

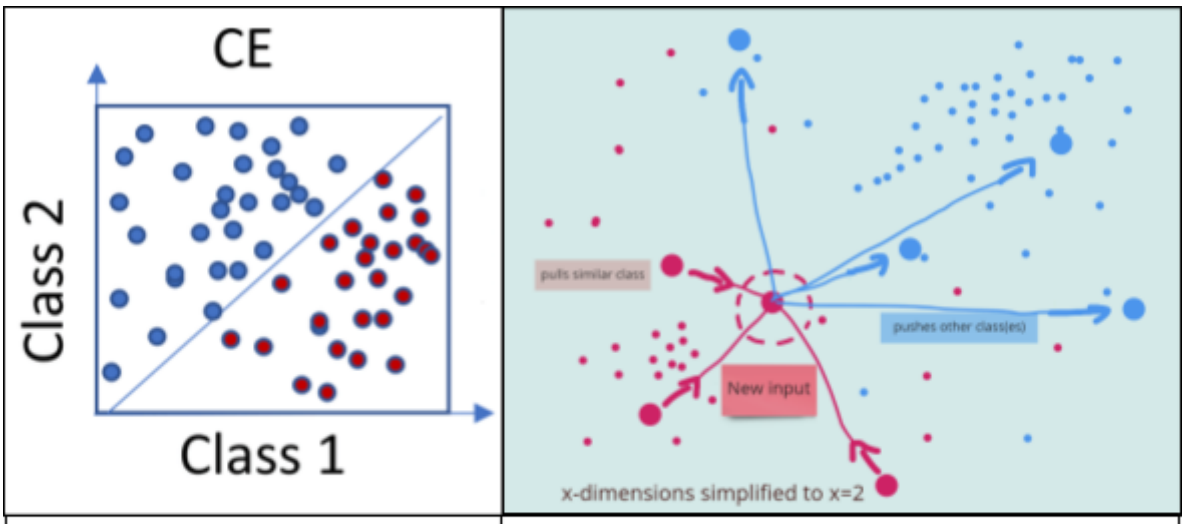
Loss functions and Supervised Contrastive Learning

During the training of a neural network, a loss function needs to be chosen. Loss functions generate feedback for the model during training. The feedback that the ANN receives is nothing more than a number based on which it can alter its weights. At the output nodes the total loss is calculated and this loss is propagated backwards over all nodes in the hidden layers recursively until it reaches the input layer, giving instructions for how to change the weights between those nodes along the way (where a larger loss is a larger change). This process is called backpropagation. Loss functions evaluate how well a model is performing to (mathematically) guide optimization. For classification tasks in machine learning 'cross entropy' and (Mean Squared Error) MSE are common choices for a loss function[3] - which calculate the loss by measuring the distance between data points and a decision boundary which separates two or more classes.

Supervised contrastive learning is a relatively novel method, which uses the distance between training examples, rather than distance from the decision boundary. This is unlike with cross entropy and MSE where the loss aims to separate classes on either side of a decision boundary. With contrastive learning, points of the same class are pulled towards each other, while repelling points of the opposite class. For a simplified example below, imagine two classes expressed in two dimensions. If the neural network receives a new input with a label, it will give an output in two dimensions (x, y). Based on the location of this new point, contrastive loss will use pairs of datapoints, and try to adjust the domain of its output classes, such that the classes overlap as little as possible, while being in proximity to its own class. There are cases where contrastive learning has improved the performance of a model, but this increase in performance appears not to be generalizable to all scenarios[4] - which was in line with our observations. Sometimes we saw an improvement in the clustering of points, but it was not always consistent.

[3] Table 1 (p. 189) of (Wang et al., 2022) notes cross-entropy as a loss function belonging to classification problems.

[4] (Khosla et al., n.d.) p.1 discusses the increase in performance from the application of supervised contrastive learning but does not extend this increase in performance to (certain) larger datasets.



Simplified 2D visualization of cross entropy loss, where the decision boundary is used to calculate the loss.	Simplified 2D visualization of supervised contrastive loss, where a new example causes points of a similar class to be pulled, and others to be pushed away.

[5]

Code Overview

This code overview consists of multiple parts which roughly and quasi-chronologically describes the workings of the code. In addition, it describes parts of the code that are not necessary for its working but that were necessary or helpful in creating a better working model. This is, chiefly, the hyperparameter optimization.

Handling Data & Feature Engineering

What Are We Working With?

The format of data we are working with, is called a time series, which is similar to a table. The table must have rows which denote (often) equal intervals of time, noting certain properties at that time in the columns. This format of data is useful in many domains, such as healthcare data, speech recognition, data about outer space or in the oceans. Time series are everywhere, as long as data is recorded over a period of time. Many aspects of our model can be generalized to other domains.

The finest resolution of financial trading data is called ‘tick’ data. These are time series, per tick, which can be a very small unit of time, or even per trade. We used 1-minute data, as it is more readily available, and makes training more practical. High resolution data, like 1-minute data, is high in noise and is very long. If we take 120 minute intervals, we essentially have the same chart, but over periods of two hours. The noise of lower timeframe movements is removed. We choose to take 1-minute data, and reconstruct any timeframe we see fit.

[5] Image source (left) (Hassanin et al., 2020)

The first step to learning from data is to turn raw data *(.csv)* into machine-readable data to learn from that can later be fed into our LSTM network as input. Not only do we perform this conversion, we also engineered several features that we hoped would 'cut through the noise'. To this end our code has several different methods to deal both with converting and transforming data to be used in our LSTM network.

Initt: Dataframes & Timeframes

The *Initt* method takes two arguments, *minutes* and *path* (to a folder containing exchange data), and converts this file *(.csv)* into a dataframe which takes into account the relevant timeframe in minutes. A dataframe is a way of storing arrays with labeled rows and columns (e.g. price, volume) which allows for many types of transformations, e.g. adding new columns/features, which are themselves a function of existing data. The data we load in has only a couple of columns, which is not informative enough for us to expect to get an accurate model. Thus, this initial dataframe from the *Initt* method forms the basis of the eventual dataframe which is expanded by future feature engineering feats.
The *minutes* argument declares the amount of minutes between each successive datapoint in our dataframe. This gives us the ability to train a model that is able to predict trends over the timescale of days, hours or minutes, etc..
The data obtained from exchanges is generally noted in OHLCV format. This stands for 'Open, High, Low, Close, Volume', and the values are given per column. These values are used to construct candlestick charts, and are used to build other metrics in technical analysis (TA). 'High' and 'Low', will be the maximum and minimum, while 'Open' and 'Close', will be the starting and ending price for that bar. Volume is simply the amount of goods exchanged within the time period, measured by either currency (in this case USD).

Get_dataset: Feature Engineering

The get_dataset method takes three arguments, *df, windowsize* and *get_features*. *Df* is simply the dataframe from the previous method. The *windowsize* argument denotes the length of a single input example, which is the amount of bars/time periods in the context of financial charts. The *get_features* argument is set to *True* by default, when set to *False* the method simply returns the inputted dataframe sliced up into sequences of length *windowsize*, *as is*, not adding features to it.

This method is itself a wrapper for doing most of the preprocessing in one call. The main processing steps are: Given the OHLCV dataframe,

- Get y: the labels for supervised learning with the method get_label
- Add features with separate methods
- Normalize features, with different methods per feature.
    - with min-max of price, in the window (OHLC, EMA)
    - with min-max of own column, of whole price history (Volume)
    - with min-max of own column, in the window (RSI, MFI)
    - between 0-100 (oscillators)
- Slice dataframe into training examples of windowsize
- Return X and y, where

- X.shape = (nr_samples, windowsize, features)
- Y.shape = (nr_samples, nr_classes)

Exponential Moving Average (EMA)

One of the features is the EMA. A 'Moving Average' looks at the history of the closing price for a given length, and calculates the average price in that period. The Exponential moving average weighs the most recent price history more. Moving averages are one of the most widely used indicators[6] - as they can show outliers from the average price. Another use case is comparing different lengths of moving averages, such that we can see when shorter impulses in movement cross the longer term price movement, and indicate a trend in the market. The EMA can be calculated taking more previous data into account and multiple EMAs can be used in parallel, e.g. to compare the average of the past day to that of the past hour. The equation for the EMA is:

$$EMA_{current} = \frac{close_{current} - close_{last}}{length + EMA_{past}}$$

Here the EMA for the current time step gets calculated through use of the close price of the current price step in combination with the past EMA. It is worth noting that EMAs have a certain length (which one is free to decide) and that an EMA needs to be initialized by calculating the average from a certain point, i.e. adding close prices over a certain length and dividing by that length. We have a lot of different length EMAs[7], where the length of each step is dependent on the amount of minutes of the dataframe.

Volume-Weighted Average Price (VWAP)

Another one of the features is the VWAP, it is similar to an EMA but in addition to taking into account the price it also takes into account the volume at which something was traded. The intuition is that we do not wish to give equal weight to e.g. an extremely high price set only by a single trade versus an extremely high price that is backed by thousands of trades. The equation for the VWAP is:

$$VWAP = \frac{high + low + close}{3} \cdot \frac{volume}{cumulative \quad volume}$$

The volume is the total amount of money that was traded (so a lot of small trades can have the same impact on the volume as one big trade), the cumulative volume is volume based on adding the total volumes of the past. We have multiple features for how far back the VWAP goes[8].

Relative Strength Index (RSI)

The Relative Strength Index is a momentum oscillator, measuring the speed and size of price fluctuations. The RSI is calculated by first calculating an upward (U) or downward (D) change, if the Close is higher than the previous close we do $U = current - previous$ and

[6] (Hatchett et al., 2010): *" The most popular method of forecasting basis is historical moving averages"* (p. 18). The EMA is a subtype of historical moving averages.
[7] To be precise, the lengths are: 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610 and 987.
[8] How far back for all of our VWAP features are: 4 hours, 12 hours, 1 day, 3 days, 1 week.

otherwise $D = previous - current$. If the current and previous close are the same then both U and D are 0 (and this would mean that there is no up or downtrend). Then to get the relative strength we do: $RS = \frac{SMMA(U,n)}{SMMA(D,n)}$. Where SMMA (Smoothed Modified Moving Average) takes these values and turns them into moving averages[9]. To then get the RSI we take this value and turn it into a percentage (resulting in: $RSI = 100 - \frac{100}{1+RS}$).

Money Flow Index (MFI)

Money flow index is an oscillator which uses price and volume data to try and find overbought or oversold signals in our dataset. Essentially meaning that it is trying to find points at which prices are higher or lower than they 'should be' such that we can expect that a price change is imminent. Unlike conventional oscillators such as the RSI mentioned before, the MFI incorporates not only the normal price index but also the volume data. An MFI above 80 is considered overbought and an MFI below 20 is considered oversold, although sometimes people also use 90 and 10 for this to decrease risk. The MFI is calculated by taking the typical price of the last 14 time periods (where $Typical\ Price = \frac{High + Low + Close}{3}$). Then for each period, mark whether the typical price was higher or lower than the prior period. This will show if the current money flow is either positive or negative. Then we calculate the money flow by multiplying the typical price by the volume for that period, where positive numbers indicate an uptrend and negative numbers a downtrend. Then to calculate the money flow ratio we add all the positive numbers together and divide this by the negative numbers (all still within these 14 periods). Finally, the money flow index is simply converting the ratio into a percentage. Thus: $Money\ Flow\ Index = \frac{100}{1 + Money\ Flow\ Ratio}$ where $Money\ Flow\ Ratio = \frac{14\ Period\ Positive\ Money\ Flow}{14\ Period\ Negative\ Money\ Flow}$.

Stochastic RSI (StochRSI) & other oscillators

Both the Stochastic RSI, and the Ultimate Oscillator, are oscillators that range between 0 and 100. We can achieve this with the following function over another indicator, for instance RSI. The stochastic RSI can be obtained by substituting X with the RSI values of the same time series in the following formula.

$$Stoch(x) = \frac{x - min(x)}{max(x) - min(x)}$$

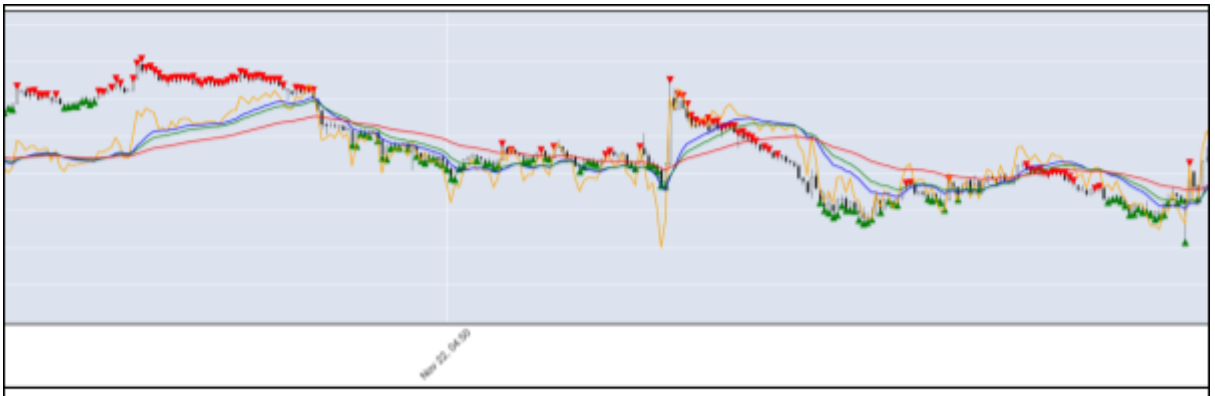
This way, the values will reflect the original indicator, but always be ranging between the same max and min. Often thresholds are added to indicate 'oversold' and 'overbought', and can demonstrate waves of momentum for different periods of time. The oscillating format is ideal for machine learning, as the normalized values will follow the same distribution as

[9] How to calculate the SMMA is not covered but it bears much resemblance to the EMA.

before normalization. Different lengths can also encapsulate the momentum of multiple timeframes.

Get_label: what to predict?

Supervised learning is generally more effective than unsupervised learning. Unsupervised learning means that patterns are learned without having the labels be given, thus a category can be learned without learning from examples with given categories. An unsupervised neural network learns to associate solely based on similarity. With supervised learning you need labels but for unsupervised learning the neural network has to figure out correlations within the data and across multiple examples. Unsupervised learning can perform very well in certain situations, but with supervised learning we can direct the weights and biases towards the desired conclusion, regardless of whether the problem is learnable[10] or not. However, the big challenge with supervised learning is the requirement of labeled data, which is having an answer key to each input example, which the AI will try to learn to predict. Data can originate from a 'true' function, for example samples generated by some polynomial have that underlying polynomial as their 'true' function. Then ideally, the AI should predict the polynomial, given the input examples without the true function. Since the market has no 'true function' we look at the future for each point in time.
The get_label method takes a dataframe, which must have at least the OHLCV columns, and for each bar, it will look into the future to determine if the price is higher or lower than the current price. With the price from multiple lookahead lengths, the score of a single bar is made, so that an average of long and short term is taken. There are two methods implemented to translate this score to a classification for *buy, sell* or *hold*. We can use a threshold (old method), where after normalization, we can assign thresholds (EG: 0.25 and 0.75) for classifying *buy* and *sell*, with the remaining being *hold*. This threshold can be moved easily to balance the classes. The second and newer method takes moving averages over the score, and uses that to determine noisy moves, with respect to the larger trend. By adjusting the EMA lengths, we can balance classes, and adjust how sensitive the signals are.



Buys and sells are only allowed if the signal (yellow) is under and above the shortest signal (blue)
EMA lengths can be applied to make it the larger trend, and balance label frequency

[10] An example of a learnable problem is 'given a picture of a cat or dog, decide what is in the image." An example of an unlearnable problem is: 'Given data of previous coinflips, decide what the outcome of the next coinflip will be'. N.B. The importance here lies in predictability, some deterministic processes are not viable candidates (e.g. deterministic functions in chaos theory) while some stochastic processes might be viable candidates (e.g. Gaussian processes (which follow a normal distribution)).

LSTM Network Training and Testing

The LSTM network takes the data from the previous section as input and uses it to try to find patterns that are able to predict our desired output. This happens over several methods which converge in a wrapper method and are called in succession, where the output of the previous method is necessary for the method that follows it.

Desired Output

We want our model to be able to predict some useful information for us but it is not entirely clear what this information should be. The first intuition is perhaps for it to predict the future closing price. This is a prediction that a model will almost always get wrong as there are very many possibilities, this is not a problem in itself, but the bigger problem is that it is not a directly useful prediction. It is not directly useful since we might reasonably want to conduct trades based on our model's prediction and this would already require us to calculate the difference between the current price and the predicted future price to see if it would be a good prediction. Instead we want our model to directly tell us what to do at the current time step, do we *buy* more Bitcoin, do we *sell* (part of) our Bitcoin or do we *hold* onto our Bitcoin. If our model has a solid estimate on whether the price will go down or go up it should predict either *buy* or *sell* and when it is more unsure of what will happen next (or sure that it will not go up or down) it will *hold*.
Our desired output is calculated based on taking any timestep and calculating what the best move would have been at that moment (knowing what we know now). This is done through the method *get_label*

Network Architecture

The architecture of our network is made up of three distinct parts, the encoder, the projection head and the classifier. When training the neural network, the user can choose different losses such as supervised contrastive learning, or categorical cross entropy. We would need more trials of experimenting, to determine what loss function works best for the classifier.

Encoder (Bidirectional LSTM network)

The encoder encodes the input through a bidirectional LSTM network. This encoding is the foundation of the network's architecture as it is here that the data, sliced according to a certain (temporal) window size, gets its features encoded as a mathematical representation (which is dependent on the temporal sequence of the input data).
In a standard LSTM/RNN, each node's prediction is based on the input data, and the previous predictions of the previous bars in the time sequence, the data gets fed one way. The bidirectional LSTM, will see dependencies that go both ways in time, not only forward. Because LSTM networks can only view one input at a time, recognising patterns both forwards and backwards is important to get the most out of the data. The motivation for a bidirectional LSTM network is that it can glean valuable insights from getting the data

forward and backward in time such that it is able to tune its weights in such a manner that it also increases its accuracy when it only gets data given forward in time[11].
The encoder's setup, the amount of layers and the parameters for these layers, gets initialized with the method *create_encoder*.
In the method make_LSTM, the encoder will require a fully connected network to connect to the outputs of the LSTM layers. This is because our labels do not match the layer size of the LSTM layers. We use the fully connected network to feedforward and converge the layersize to an output of 3. This way, there is no mismatch between label and layer size, this is necessary for backpropagation, and thus for learning to take place.

Projection Head

When training the encoder which is made up of LSTMs, to check with the labels, we need the output layer to be of size 3, such that the network can output probabilities for the prediction of the labels: *buy, hold and sell*. However, we wish to have a high-dimensional representation of the data to retain as much information as possible. The projection head allows us to connect the encoder to the labels. The LSTM network has an internal representation of previous inputs in the sequence, its memory, and all this information cannot be translated into the prediction classes which we want. This is a result of the structure of the LSTM network, its representation is unfit to calculate a loss over. The projection head fixes this as it adds a densely connected (vanilla) ANN layer; the LSTM network converges, i.e. gets projected, to this layer to calculate the loss. This layer forms the output layer.
This is done with the method *add_projection_head*, whose chief input parameter is an encoder such that its output becomes a new encoder with a projection head. This new encoder is then trained and tested so that we have a frozen model (without projection head) to train the classifier.

Classifier

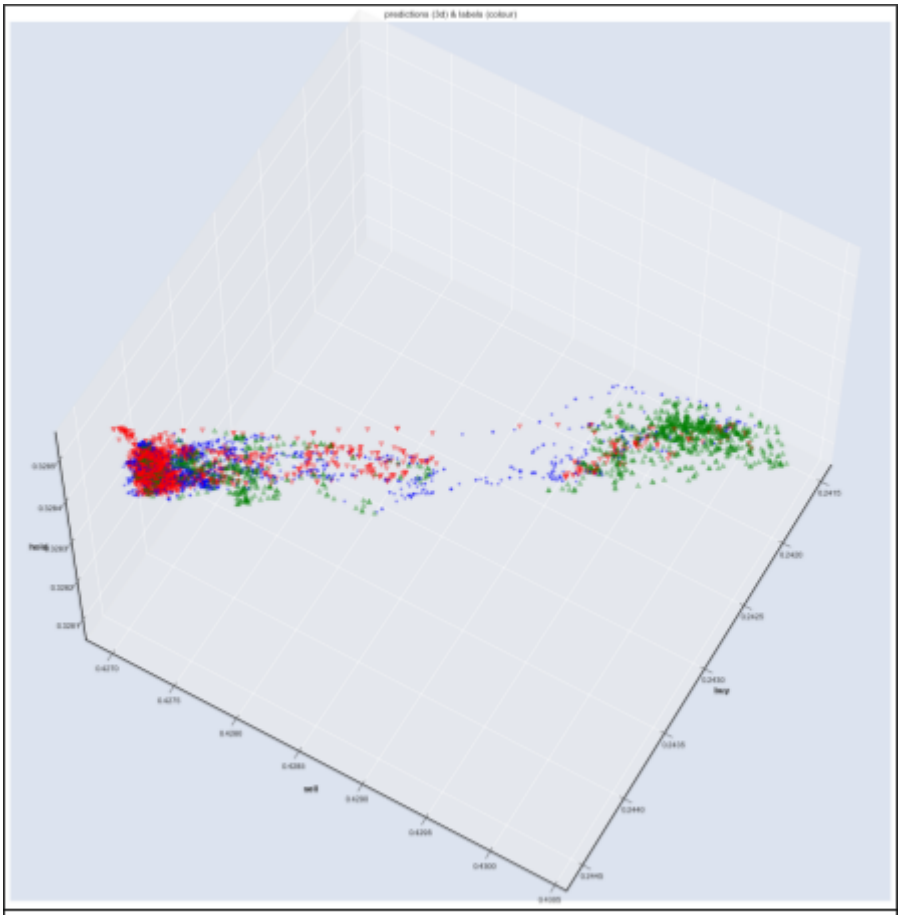
The classifier is used to assign class labels to the data inputs, for example in image recognition this would be classification in the sense of male/female or dog/cat. In our code the classification will be into 3 categories; *buy, sell* and *hold*.
Our classifier uses the previously trained encoder with head and a variable named *trainable*. If *trainable* is set to *False*, then the classifier freezes the previously trained layers so it can use the pretrained encoder and only train the newly added layers. This means that those previous layers' weights are left unchanged while the newly added layers' weights keep changing during training. The classification class adds a fully connected layer on top of the encoder, and another layer that matches the output size we want.

Supervised Contrastive Loss[12]

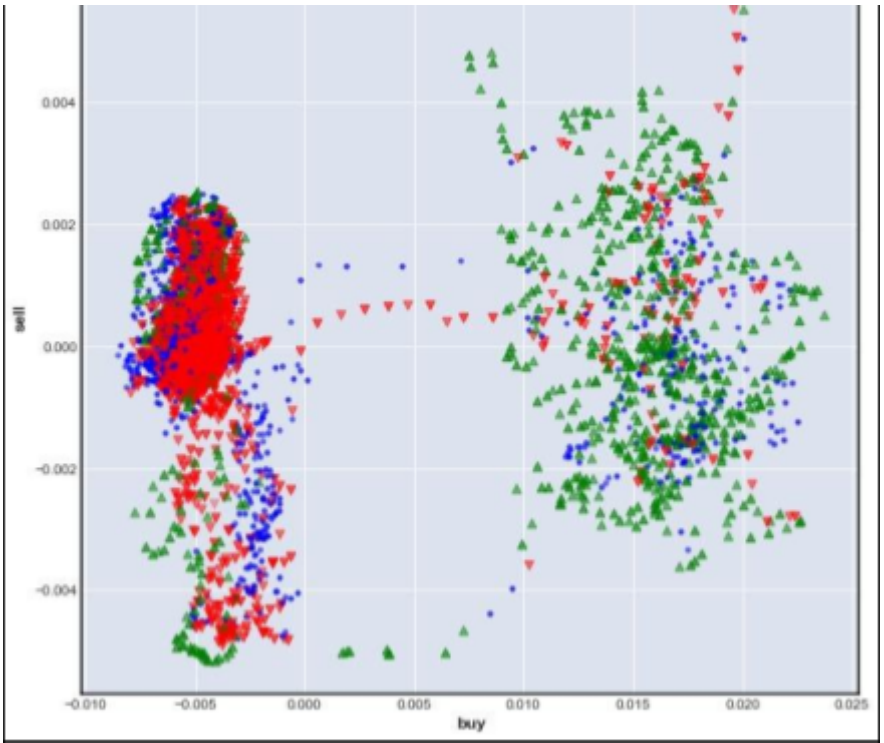
Supervised contrastive loss is a loss function which was created as an alternative to cross entropy that is argued to better leverage label information. It does this by calculating the

[11] (Althelaya et al., 2018)
[12] Our implementation of supervised contrastive loss is an adaption of the example on the Keras repository (*Supervised-Contrastive-Learning.py at Master · Keras-Team/keras-Io*, n.d.) (whose application was for image recognition), theoretical grounding for supervised contrastive learning (and thus loss) is from (Khosla et al., n.d.).

distance from itself to an example of the same class and then contrasts this with the distance to examples of another class, meaning that the loss is low if datapoints belonging to each of the separate classes (*buy, hold, sell*) are encoded to separate multidimensional representations. The distance calculation is done by taking the cosine distance of the vectors and using this as the predictions' probabilities you would see in a more typical categorization method. The resulting idea is that we can take these distances and normalize them between 1 and 0 to then apply cross entropy loss. If done correctly this will create clusters as seen on the image below.



This is another example of successful clustering, in 3d. The axes are the probabilities for *buy*, *sell* and *hold*.



The image is a PCA plot of the model output (contrastive learning encoder + cross entropy decoder, i.e. the classifier), with dimensionality reduced from 3 to 2 (the dimension for *hold* is gone).

Cross Entropy Loss

Cross entropy loss or log loss measures the performance of a model based on how far it drifts away from the actual labels, so in our case loss increases as the distance from the cluster increases. In our code this gets used in the classifier where it receives the earlier created normalized distances with which it then creates the loss values. For this in our code we use the already existing categorical cross entropy loss function from keras (we use categorical cross entropy since we do not deal with just positive or negative values but with the probabilistic predictions for classes *buy, hold, sell*).

Hyperparameter Optimization

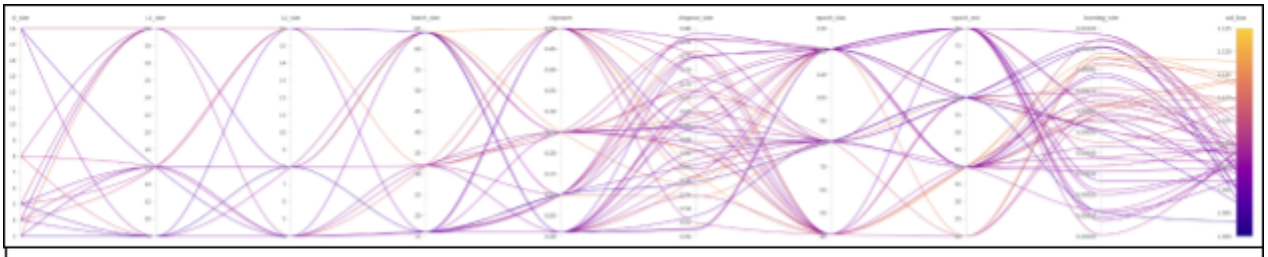
In the training phase of a model it undergoes automated optimization, generally, in terms of maximizing the accuracy or minimizing the loss. What is already set before the training starts are all the hyperparameters. Whereas training changes the weights the hyperparameters are all out of reach for changing through the training. Among these are the amount of layers, the amount of nodes in a layer, the learning rate or even the loss function. This shows the extent to which hyperparameters themselves can exert a great influence on the accuracy and loss of the model. As such it is important to find a way to optimize these hyperparameters to some extent as well and not have their specifications be the result of puny human minds (puny with regards to their ability to manually find the optimal hyperparameters). Even when you are utilizing techniques to prune unpromising hyperparameter values it is still a computationally intensive process that requires a lot of (automated) trial and error; the upside of this is that once the hyperparameter optimization is complete there should be little to no reason to run this optimization ever again. That is, if nothing else changes that could affect what/how the model optimizes. This very fact causes hyperparameter optimization to commonly only occur near the end of the project, as a bit of an afterthought[13]. With this also comes with the fact that it might not happen at all or be lacking/incomplete (see subsection: *Window Size Optimization & Timeframe Optimization* from the *Discussion* section).

Our hyperparameter optimization happens in the *main1.py* file and occurs only if *sweep* is set to *True*. When it is, new models are initialized with differing hyperparameters, either over a range (e.g. learning rate) or a set of values (e.g. batch size) or a category (e.g. supervised contrastive learning [ON/OFF])[14]. In order to find the optimal combination of hyperparameters it tries to minimize the validation loss across all runs. In order to save on computing costs there is an early stopping mechanism such that highly unpromising runs (in comparison to the previous runs) are cut off early; if the loss of the testing (validation) set does not decrease in a given amount of epochs, then the run will stop.

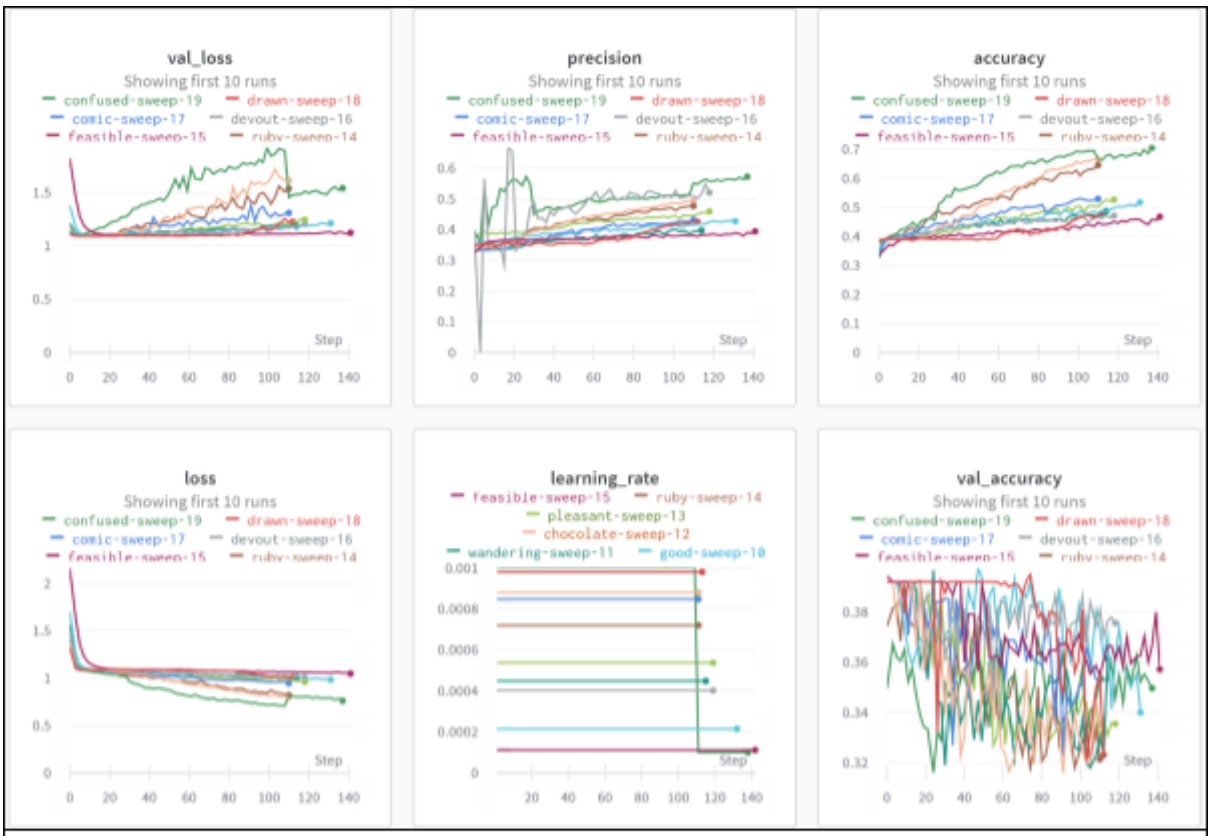
---

[13] Since hyperparameter optimization has the ability to increase performance significantly (but also has the ability to not make much of a difference) an interesting middle-ground can be found with algorithms that are (with less required computation) able to determine whether or not hyperparameter optimization will greatly improve performance. See: (Tran et al., 2020).

[14] The full list of these hyperparameters are: batch_size (batch size) {16, 32, 64}, learning_rate (learning rate) [1e-6, 1e-3], dropout_rate (dropout rate) [0.6, 0.9], epoch_enc (nr. of epochs for the encoder) {43, 67}, epoch_class (nr. of epochs for the classifier) {43, 61}, L1_size (nr. of nodes in the first (LSTM) layer) {8, 16, 32} , L2_size (nr. of nodes in the second (LSTM) layer) {4, 8, 16}, D_size {3, 4, 5, 8, 16}, SupCon {True, False} and clipnorm {0.01, 0.1, 0.25, 0.5}.



A figure displaying multiple runs with accompanying parameters and their corresponding loss.



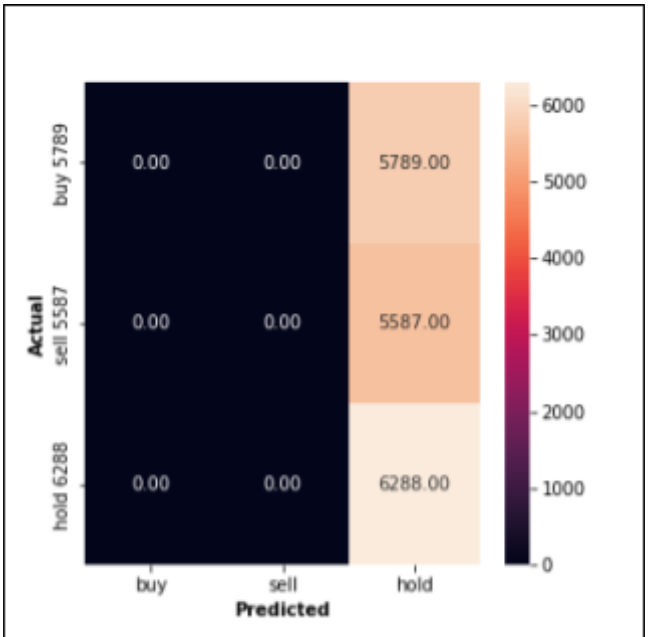
This image shows one of the trials we have done to find optimal parameters. You can see 10 runs on display, which is a subset of the whole trial. During a trial each run will get different parameters (such as learning rate). We can see that generally, accuracy and loss increase and decrease as they should. However, validation loss rises after falling for a bit, and validation accuracy fails to improve. This means that our model is not good enough in generalizing, and overfits to the given data.

Conclusion

We conclude not just with the results from our specific undertaking but about the prospect of predicting the price of Bitcoin more generally and on a broader scale the prospect of financial forecasting (aided by machine learning techniques).

Results

Our results show a promising accuracy (>40% on certain runs) yet with less promising results under the hood[15]. Many of the runs have an associated confusion matrix similar to the one below. Such an outcome is generally ascribed to the imbalanced classification with class imbalance problem; it is when classifications are skewed in such a way that it affects the model's predictions to be skewed as well[16]. In the confusion matrix below on the left we can see that there is indeed a higher likelihood of *hold*, overall, as there are simply more instances of it ($hold : 6288 > buy : 5789 > sell : 5587$). We can also conclude that this was picked up through learning (to quite a dramatic extent) and was what caused the prediction to converge to always be *hold*. Whilst failing to learn to predict reliably there is a tiny success in the fact that it did at least pick *hold* over *buy* or *sell*.



A confusion matrix predicting hold 100% of the time.

The confusion matrix shows the actual values for the associated labels on the vertical axes and the predicted labels on the horizontal axes (with values being displayed in the matrix on the table).

This case (and its frequency across multiple runs) provides reason to assume that it is a case of class imbalance, something for which there is not a cut and dry fix but something for

[15] N.B. The accuracy is calculated by dividing the amount of correctly predicted labels by the total amount of predictions.

[16] For an overview of the class imbalance classification problem and potential ways to combat this, see: (Ali et al., n.d.).

which there are many different approaches with proven efficacy[17]. Our attempts at this have been able to bring about a more varied prediction across the multiple classes. These have not been consistently more accurate than a single class would have been. The greater the class imbalance, the harder it was for our model to learn. The likelihood of a certain class will be out of proportion, causing a negative feedback loop to predict the same class.
What the root(s) of this problem are is still an open question. There is a significant chance that hyperparameters were (partly) responsible for a lack of 'expressive power' of our model (see: *Possible Improvements* in the *Discussion* section). What is hard to deny is that *something* is being learned by the encoder, merely by looking at the clusters forming we can deduce that there is probably some sensible representation of our data in the latent space, but it fails to accord well with the desired labels. A different response might be to say that it does learn these clusters in accordance with the labels but that the classifier is unable to extract this information from these clusters. In both of these cases it could mean that an LSTM network or supervised contrastive loss (or both) are wrongheaded approaches or it could be that these methods can still be hugely improved with mere hyperparameter tuning. A good indication that problems might lie somewhere with the encoder would be by implementing feature selection and seeing which features contribute to which degree to the model's performance (see: *Possible Improvements* in the *Discussion* section)[18]. The very last option is that this is an undertaking that is doomed to fail. In some sense that is true (see subsection below), but in many ways when we decide on an undertaking we can simultaneously decide when that undertaking has succeeded. An existing paper published on the subject of Bitcoin price prediction notes their 52% accuracy in predicting whether the price will go up or will go down[19]. This can set a precedent where our undertaking is not a fundamental failure as long as our predictions for *buy, sell, hold* are able to surpass 33⅓% accuracy (without exploiting a classification imbalance to achieve this). There is little (but not no) reason to assume that this is an unattainable goal.

Possibility of Bitcoin Forecasting

Whether or not (or to which extent) we were successful at predicting the price of Bitcoin is a different question from the question to which extent we were unable to do this. Technical analysis for financial forecasting has seen varying success as has the application for neural

[17] For the claim that there is no 'cut and dry' method. (Abd Elrahman & Abraham, 2013): *"The researchers for solving the imbalance problem have proposed various approaches. However, there is no general approach proper for all imbalance data sets and there is no unification framework"* (p. 339).
[18] It is worth noting that such an implementation could possibly point to a necessity to revise the features themselves, i.e. all/some features are shown to be (relatively) uninformative. That this will be the case for all features seems like an unlikely possibility but it would still not detract greatly from the undertaking as it would lead us to still be utilizing technical analysis but simply with different technical indicators. The current set of indicators has resulted primarily from an approach of *'seeing what sticks'*, this would merely be a continuation of that process. Likewise for the case where we need to add more and/or revise existing technical indicators.
A problem would arise if it were to be shown that technical indicators *overall* do not contribute much to performance compared to the simple *open, high, low, close* and *volume* features. This would mean a failure of the technical analysis as a whole, which would be unlikely given its success when utilized by human agents (the first part of footnote 14 elucidates this point).
[19] (McNally et al., 2018) *"The LSTM achieves the highest classification accuracy of 52%"* (p. 339).

networks in tandem with technical indicators[20]. Even so, this could still be chalked up to undiscovered technical indicators with high predictive powers or similar undiscovered machine learning techniques with high predictive powers (or an alternative is a lack of computing power with existing machine learning techniques). But there are other reasons to assume that a high accuracy is highly unlikely with the application of technical analysis and neural networks in tandem.

The seminal paper describing the workings of Bitcoin[21] was published in 2008 around the time of the failure of banking institutions in the United States, which caused a global economic collapse. This failure had been predicted by some and many more (primarily due to the power of hindsight) would say that the writing was on the wall, but what did not and could not reliably predict such a crash was technical analysis. Bitcoin itself was set up to be free from these centralized institutions as a currency with full transparency. Yet, of course, this transparency does not necessarily mean that its valuation is transparent to thorough technical analysis. This transparency, insofar as it cannot be meddled with by centralized institutions, adds more hope toward successful application of technical analysis. On the other hand, whatever centralized measures exist precisely to withhold someone from meddling is what *takes away* hope toward successful application of technical analysis. As mentioned in the *Introduction* section, a higher error rate might not matter much if it is able to better exploit the large market movements that this, less consolidated and thus, more volatile currency brings. As such, a decrease in predictive power with increased volatility could still be a more successful model in the end when evaluated in terms of the returns it is able to realize.

A striking example of this lack of centralized authority making technical analysis harder came from an adjacent cryptocurrency called Dogecoin. Someone had a tremendously successful prediction model for Dogecoin. Their prediction model was not based on technical analysis but on sentimental analysis, and it was the sentiment that they themselves expressed that was the predictor; they are being sued for running a pyramid scheme[22]. This is not to say that adding sentimental analysis to the technical analysis would be a solution (though it has the potential to increase accuracy significantly) but rather that there are always more things that factor into something's value than one is able to factor in. We can already name the large effects on Bitcoin's price that come from its ability to 'compete' with other cryptocurrencies, legislation aiming to centralize cryptocurrency (e.g. through mandatory identity verification on exchange markets), environmental concerns

---

[20] On the claim that technical analysis has varying success: In the abstract of (Park & Irwin, n.d.) *"Among a total of 92 modern studies [of technical trading strategies], 58 studies found positive results regarding technical trading strategies, while 24 studies obtained negative results."*
On the claim that the application of technical analysis to neural networks has varying success: (Sezer et al., 2017) states *"The results indicate that by choosing the most appropriate technical indicators, the neural network model can achieve comparable results against the Buy and Hold strategy in most of the cases"* (p. 1) which is on neutral ground and (Thawornwong et al., 2003) notes that *"the proportion of correct predictions and the portability of stock trading guided by these neural networks are higher than those guided by their benchmarks"* (p. 313), likewise (Sang & Di Pierro, 2019) notes that *"We show that our strategy, based on a combination of neural network prediction, and traditional technical analysis, performs better than the latter alone"* (p. 1). Not as convincing with N=3 as (I could find) no literature review which handles artificial neural networks specifically whose features are technical indicators exclusively.

[21] (Nakamoto, 2008)

[22] (Stempel, 2022)

regarding crypto's high power usage and much more. Technical analysis seems to be able to capture some of these outside forces acting upon a valuation and predict accordingly but it cannot integrate enough factors to do so reliably. How was the 2008 housing bubble to be distinguished from those saying that the internet was just a fad (Dot-com bubble notwithstanding). Attempts to integrate more and more of these sources into one's prediction, e.g. sentimental analysis, will still have an easily intelligible *ceteris paribus* case with a wholly different outcome.

This does not need to mean that such an approach is doomed to fail, it could just mean that the bar is simply set too high. The main gripe is that the predictions need to be reliable, but human predictions in these cases can (and generally *is*) also be unreliable. Humans that make trades and that are very often wrong but are still right when it counts can still turn a profit. As such this might be what we measure success by for our model, profitability[23]. Even a model that makes less reliable predictions can be more profitable than a human as they (very) rarely take breaks and are okay getting paid a poverty wage equivalent in electricity.

[23] An alternative measure for success based on accuracy was mentioned in the *Results* section.

Discussion

This section covers multiple (possible) improvements to our model and some reflection on the process of our project.

Implemented Improvements

Along the way, as we were both programming and checking the performance of our models, we ran into ways to improve our model either by altering an existing implementation in the code or adding an altogether new implementation. The ideas often come through observing bottlenecks in our performance and finding ways to fix it or otherwise just trying something new or coming across a paper with tips on what might bring about a better performing model. This subsection highlights some of these implementations.

Bidirectional LSTM

When we started out we used a regular LSTM which only gets its input sequences chronologically. Later on we switched to a bidirectional LSTM which considers its input sequences moving forward in time and backward in time. The motivation for implementing this was a published paper that saw improved performance for the bidirectional LSTM when compared to the LSTM[24]. After implementing it ourselves we observed an improvement on performance. The network was able to converge to clusters in less training epochs than with the unidirectional LSTM. It did however take longer to train.

From Softmax to Sigmoid

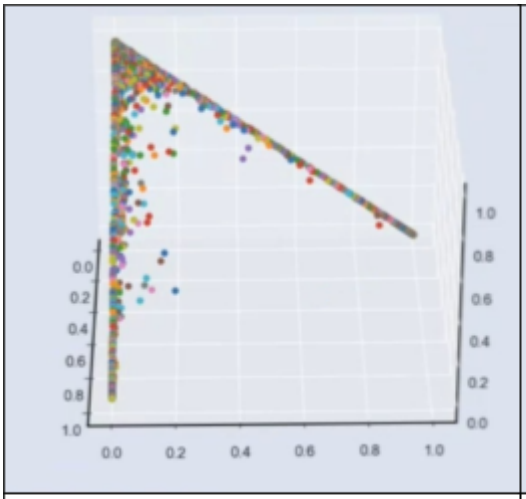
Each layer in a neural network must have an activation function, to map the inputs to a fixed domain. This dataprocessing step can help reduce over/underfitting, and can help with exploding or vanishing gradients[25]. There are multiple activation functions that can be considered.

An earlier version of our model outputted (the different labels) with a softmax activation function. It was the relatively poor performance in combination with the three-dimensional plots that gave us insight into how we might further improve our model. Looking at the plot there was a strange and undesirable behavior that our outputs seemed to quite rigidly keep outputting along two-dimensional axes in this three-dimensional plane. The problem we might figure this to be is that softmax requires the sum of the output to be 1 (i.e. $P(buy) + P(sell) + P(hold) = 1.0$). The sigmoid activation function did not have this problem, and assigns a probability between 0 and 1, for each class separately. Upon changing to this as the output activation function we saw some improved performance and clearer clusters forming along the different axes on the three-dimensional plot. Though in

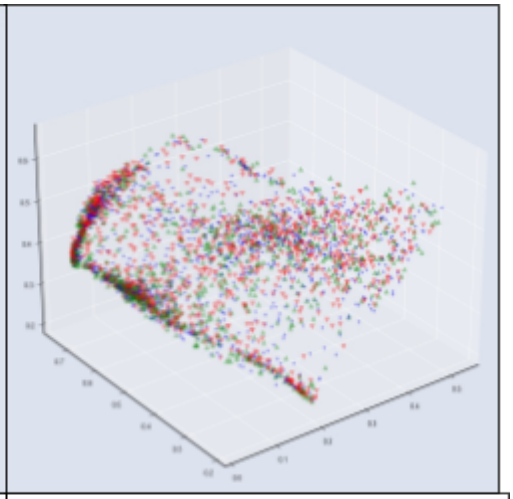
[24] (Althelaya et al., 2018)

[25] (Li et al., 2019) finds reduced overfitting with their uniquely created activation function. (Dubowski, 2020) compares many types of activation functions in sparse neural networks and finds corresponding over and underfitting for specific activation functions. (Mercioni & Holban, 2020) notes the advantages and disadvantages of several different activation functions including their (in)ability to solve the exploding and vanishing gradient problem.

retrospect, dimensions did not collapse only due to our activation function, but also due to the model architecture and hyperparameters. A possible answer to why many sets of hyperparameters resulted in scatterplots which resemble lines/planes is that the dimensions collapsed because the network was not stable enough. Stable meaning, the balance between learning rate and other hyperparameters, resulting in optimal learning. When the parameters are not in the 'sweet spot' the resulting mathematical representation will make less sense as the network has focused on the wrong details. Tweaking the activation function helped that problem. Now we are usingNow we are experimenting with combinations of softmax, sigmoid, and relu.



Plot of a model with unoptimized hyperparameters, resulting in collapsing of dimensions.

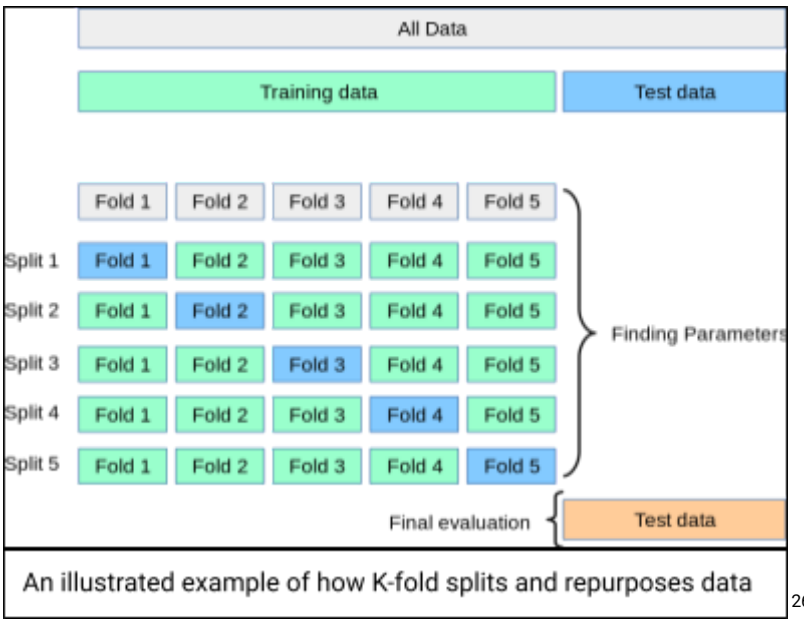


Among other tweaks, replacing softmax functions with sigmoid, helped us create more expressive plots where more complex dependencies can be seen.

K-fold Cross Validation

Closing in on the end of the project, we still managed to implement k-fold cross validation, where our dataset gets partitioned into k parts, and we train on all the parts except one, which will be used as testing data. A new model will be trained and tested *k* times. By shifting the part that is used for testing across all k runs we can use our whole dataset for testing rather than only evaluate on a small subset of the data.

Ideally we would want to see similar performance across all models, but this was not the case. Probably this was due to the differences in the training data, as one partition will contain more uptrend or downtrend data than the other, thus creating imbalanced labels. We could see much better clustering when using the last partition as a testing set. This could have to do with the chronological order of the data. We will have to experiment more to make any conclusive statements.



An illustrated example of how K-fold splits and repurposes data [26]

Possible Improvements

Due to time constraints (and perhaps resource constraints) we were able to think of more possible implementations that could improve the performance of our model than we were able to realize. This section discusses some of those promising leads.

Window Size Optimization & Timeframe Optimization

Our current code already has quite a robust hyperparameter optimization in place. But it does not have anything in place to optimize the window size. How to find a robust optimization technique for the window size is not straightforward as there does not exist a case where everything else remains the same for the inputs and the desired outputs; when the window size is altered the entire dataset shifts such that a better accuracy or lower loss does not in and of itself mean a *better* model.
With timeframes the exact same problem occurs such that whatever differing results the model gives they are not truly commensurable. Nevertheless, it is also immediately obvious that neither window size nor timeframe are arbitrary properties and that *there are* better and worse choices, just without a clear idea of how to measure this beyond the level of intuition.

Handling Multiple Timeframes

From the start it was clear that a model with a more holistic overview of the market would be better able to give predictions. This was already the guiding principle to our feature engineering and our network architecture. An LSTM is able to have much more salient information at its disposal than a vanilla RNN and an RNN than a vanilla ANN. What an LSTM is lacking is multiple levels of overview. The most intuitively simple implementation of this we came up with would be a network whose nodes themselves are made up of LSTM networks with different timeframes that all converge with a single output. Simple on the level

[26] Image source: (Pedregosa et al., n.d.) p. 3.

of intuition does not mean simple on the level of implementation. Thus this is not quite as straightforward as this description might make it out to be. These LSTM networks must (most likely) have different input channels yet must converge, meaningfully, into a single output layer. Hopefully this network would learn with which kind of inputs to give more sway to long-term or short-term considerations, not stepping into the pitfall of suggesting trades that would be good advice for the next minute, if only it was not followed by a successive minute.
The next large problem one faces when trying to implement this was already touched upon in the previous subsubsection and it is the question of how to decide which timeframes to have, you *do not* wish to have redundant computation (by picking highly similar timeframes) and you *do not* want to have your well-considered selections rest upon randomly picking relatively far apart timeframes. It could be that randomly instantiating several of these networks with multiple LSTM networks and applying a genetic algorithm for the selection among these would prove fruitful, reservations on their incommensurability notwithstanding.

Higher Upper-Bound on Hyperparameter Optimization

Whilst hyperparameter optimization could be expanded by adding new hyperparameters to optimize for, as shown above, there is still another possibility for improvement. This improvement would come from increasing the upper bound on certain hyperparameters, most notably the amount of layers and the amounts of nodes in a layer[27]. Whilst the amount of parameters might already look staggering, in the tens and hundred of thousands, it is nothing compared to the billions in state-of-the-art machine learning models. Bigger is not necessarily better, but there is little reason to assume that we have already reached a saturation point with our model. For our purposes it might mean much more, highly needed, expressive power to capture nuances in the data.
Obviously there were practical considerations that caused us not to increase the upper bound as it is a computationally intensive (and thus cumbersome and costly) process to train huge models. It is still a worthwhile addition (especially) if the alternative is to abandon this route altogether.

Feature Selection

Implementing some type of feature selection algorithm would give us insight into the individual contribution of our features to the model's performance which could lead to us removing some features as they might not add to performance or could even detract from it, not *'cutting through the noise'* (as we would like from our features) but *'adding to the noise'*. Such a feature selection algorithm would also make it (more) feasible to implement an obscene amount of technical indicators (with computing power to match) and removing those relatively uninformative features.
A possible approach to this is through implementation of a random forest-based feature selection. This creates (random) trees based subsets of our features and checks how much

[27] We do not expect to see similar possible gains in increasing the amount of epochs as the model seems to consistently converge. Yet this might no longer happen *when* the amount of layers and nodes per layer have been increased such that a higher upper bound for the amount of epochs would also need to be implemented.

they aid in the predictive capabilities, averaging over all of these means we can get a good idea of the effectiveness of these features across many different contexts. This already has a shortcoming which is that similar features will be judged similarly, e.g. our different length EMAs.
With a proper feature selection module, we can experiment with much more values in an automated manner. We will be able to optimize the hyperparameters of the features, besides only those of the neural network.

Ensemble Learning

Improvements in performance could be achieved by implementing some type of ensemble learning. *"Learning algorithms that output only a single hypothesis [buy, sell, hold] suffer from three problems that can be partly overcome by ensemble methods: the statistical problem, the computational problem, and representation problem"*[28].
This statistical problem arises in those cases where there might be little difference between two classes but the model still has to make the call between one of the two. The computational problem arises when the model is stuck in a local minimum that does not approach an optimal solution. The representation problem arises when the model's output is unable to represent the underlying 'true function' we presume there to be.
Slight differences in training data, or with hyperparameters, can change the mathematical representation of our data. We ideally hope for all our representations to look similar, as that would indicate us approximating some 'true' representation. However, due to the high-variance and unpredictability of the problem, representations are very sensitive to small changes. This can be compared to overfitting a model, and not being able to generalize well enough.
The way ensemble learning is able to partly overcome these problems is by first taking a multitude of models instead of one (naturally, this would require *a lot* of extra computation to implement). To all these models a meta-algorithm is applied to pick the best prediction. This way the different representations can all be incorporated to construct a big picture. There are two main ways in which the best prediction is picked. The simplest would be a majority vote (or hard vote), e.g. if we have three different models and the predictions for *buy* or *hold* are two to one we output *buy* since it has more votes than *hold*[29]. A more nuanced approach would be to take the average of all model's results to guide one's output, this method is called soft voting.
Three popular methods of ensemble learning that will be discussed briefly are bagging, boosting and stacking[30]. Bagging requires the least amount of changes to be made to our codebase to be implemented, but it might also increase performance less. Bagging is done

---

[28] (Dietterich, n.d.) p. 3
[29] The amount of models can be arbitrarily large but we can expect there to be a natural point of saturation, namely when the models which have high independent performance have been exhausted and a newly added model would have a significantly lower performance than the ones added thus far.
N.B. such a majority vote system would also require a tie-breaking system to function. Putting a higher level of trust in the independently higher performing models for tie-breaking purposes is the usual approach. (Even requiring an ordinal ranking of all models for tie-breaking edge cases.)
[30] For an introduction to the ensemble methods explaining the workings of (types of) bagging and boosting see: (Bühlmann, 2012).
For an introduction to and applications of stacking see: (Wolpert, 1992).

by taking subsets of data and training models on those subsets of data[31]. Over these different models we apply the soft or hard vote to return our predictions.
Boosting would be the easiest to implement after bagging. With boosting we train a model and then we train a new model which focuses on the subset of the data that the previous model made incorrect predictions for. We are able to repeat this, either 'boosting' across multiple initial models or 'boosting' a model multiple times. This method is more prone to overfitting than either bagging or stacking because it specifically hones in on a subset of data (instead of being randomly selected for bagging). Again soft or hard voting is applied over these different models to return our predictions.
Stacking requires a significant amount of work to be implemented as it trains all new 'types' of models. Our current LSTM and contrastive loss based model would just be one among many wholly different types of models, e.g. among implementations of support vector machines or transformers. After having trained models (on the entire training set) for all of these types we can again apply soft or hard voting to return our predictions. Stacking would most likely give the best performance out of these implementations (if all models have adequate performance independently) but would be extremely time intensive to implement[32]. The only part we would (directly) be able to reuse for each model is the feature engineering and label creation.
Ensemble learning has much to offer for exactly the types of problems we have run into with our project; first and foremost to counter the imbalanced data classification problem we have. Bagging and boosting would be the most likely candidates for their (comparative) ease of implementation.

[31] Cf. k-fold cross-validation. A method that similarly splits the data into subsets which is implemented in our code. The main difference being that k-fold cross-validation is for validation enhancing purposes whereas bagging is for performance enhancing purposes.
[32] (Wolpert, 1992) states: "[T]he conclusion is that for almost any real-world generalization problem one should use some version of stacked generalization to minimize the generalization error rate" p. 241

Bibliography

Abd Elrahman, S. M., & Abraham, A. (2013). A review of class imbalance problem. *Journal of Network and Innovative Computing*, *1*(2013), 332–340.

Ali, Shamsuddin, & Ralescu. (n.d.). Classification with class imbalance problem. *Int. J. Advance Soft Compu*. https://www.researchgate.net/profile/Aida-Ali-4/publication/288228469_Classification_with_class_imbalance_problem_A_review/links/57b556d008ae19a365faff16/Classification-with-class-imbalance-problem-A-review.pdf

Althelaya, K. A., El-Alfy, E.-S. M., & Mohammed, S. (2018). Evaluation of bidirectional LSTM for short-and long-term stock market prediction. In *2018 9th International Conference on Information and Communication Systems (ICICS)*. https://doi.org/10.1109/iacs.2018.8355458

Bühlmann, P. (2012). Bagging, Boosting and Ensemble Methods. In J. E. Gentle, W. K. Härdle, & Y. Mori (Eds.), *Handbook of Computational Statistics: Concepts and Methods* (pp. 985–1022). Springer Berlin Heidelberg.

Dietterich. (n.d.). Ensemble learning. *The Handbook of Brain Theory and*. http://courses.cs.washington.edu/courses/cse446/12wi/tgd-ensembles.pdf

Dubowski, A. (2020). Activation function impact on Sparse Neural Networks. In *arXiv [cs.NE]*. arXiv. http://arxiv.org/abs/2010.05943

Hassanin, M., Moustafa, N., & Tahtali, M. (2020). A Deep Marginal-Contrastive Defense against Adversarial Attacks on 1D Models. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1499–1505.

Hatchett, R. B., Brorsen, B. W., & Anderson, K. B. (2010). Optimal Length of Moving Average to Forecast Futures Basis. *Journal of Agricultural and Resource Economics*, *35*(1), 18–33.

Khosla, Teterwak, & Wang. (n.d.). Supervised contrastive learning. *Advances in Engineering*

Education. https://proceedings.neurips.cc/paper/2020/hash/d89a66c7c80a29b1bdbab0f2a1a94af8-Abstract.html

Li, H., Li, J., Guan, X., Liang, B., Lai, Y., & Luo, X. (2019). Research on Overfitting of Deep Learning. *2019 15th International Conference on Computational Intelligence and Security (CIS)*, 78–81.

McNally, S., Roche, J., & Caton, S. (2018). Predicting the Price of Bitcoin Using Machine Learning. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. https://doi.org/10.1109/pdp2018.2018.00060

Mercioni, M. A., & Holban, S. (2020). The Most Used Activation Functions: Classic Versus Current. *2020 International Conference on Development and Application Systems (DAS)*, 141–145.

Nakamoto. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*. https://www.debr.io/article/21260.pdf

Park, C.-H., & Irwin, S. H. (n.d.). The Profitability of Technical Analysis: A Review. In *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.603481

Pedregosa, Varoquaux, & Gramfort. (n.d.). Scikit-learn: Machine learning in Python: User Guide: 3.1. Cross-validation: evaluating estimator performance. *Learning Research and Practice*. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?ref=https://githubhelp.com

Sang, C., & Di Pierro, M. (2019). Improving trading technical analysis with TensorFlow Long Short-Term Memory (LSTM) Neural Network. *The Journal of Finance and Data Science*, *5*(1), 1–11.

Sebastião, H., & Godinho, P. (2021). Forecasting and trading cryptocurrencies with machine learning under changing market conditions. *Financial Innovation*, *7*(1), 3.

Sezer, O. B., Ozbayoglu, A. M., & Dogdu, E. (2017). An Artificial Neural Network-based Stock Trading System Using Technical Analysis and Big Data Framework. *Proceedings of the SouthEast Conference*, 223–226.

Stempel, J. (2022, June 17). Elon Musk sued for $258 billion over alleged Dogecoin pyramid scheme. *Reuters*. https://www.reuters.com/legal/transactional/elon-musk-sued-258-billion-over-alleged-dogecoin-pyramid-scheme-2022-06-16/

supervised-contrastive-learning.py at master · keras-team/keras-io. (n.d.). Github. Retrieved June 24, 2022, from https://github.com/keras-team/keras-io

Thawornwong, S., Enke, D., & Dagli, C. (2003). Neural Networks as a Decision Maker for Stock Trading: A Technical Analysis Approach. *International Journal of Smart Engineering System Design*, *5*(4), 313–325.

Tran, N., Schneider, J.-G., Weber, I., & Qin, A. K. (2020). Hyper-parameter optimization in classification: To-do or not-to-do. *Pattern Recognition*, *103*(107245), 107245.

Wang, Q., Ma, Y., Zhao, K., & Tian, Y. (2022). A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science*, *9*(2), 187–212.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks: The Official Journal of the International Neural Network Society*, *5*(2), 241–259.