Ranking financial assets using neural networks

Anton Ponomarev
aponom22@stanford.edu

ABSTRACT

In this paper, we explore the application of neural networks to the problem of selecting the best performing financial asset from a list with a pre-defined investment horizon, while using only historical prices. We investigate a number of network architectures and come to a conclusion that a network with LSTM layers delivers best results. However, while we manage to train the network to achieve a meaningful accuracy, the results fall short of our expectations. We offer a number of explanations for this and propose several potential enhancements.

INTRODUCTION

Technological advances made over the past decade in the way information is acquired and shared have disrupted many facets of people's lives. The area of managing personal finances and investments is an example of that. The advent of mobile devices and services has had a dramatic effect on people's relationship with money. Deep Learning has a tremendous potential to further disrupt the way technology assists people in making sound financial decisions. But so far, in the opinion of the author, most relevant applications of machine learning techniques in finance have mostly been reserved for sophisticated institutional investors and academics.

Upon reviewing the existing literature, one may draw similar conclusions that most attention has been paid to applications dealing with large data sets. In finance, the easiest way to achieve that is by increasing the frequency of market data sampling, for example by looking at tic-by-tic price changes driving the dynamics of the order book [1], or by analysing financial news headlines and other social media sources to predict future stock price movements [2,3,4]. This seems reasonable considering the inherent hunger for data that most deep learning techniques possess. However, this also makes the applications more unattainable to the average investor.

In this project, we aimed to explore potential applications of neural networks to helping answer probably one of the most common questions an average investor would ask: "given a list of financial assets and a desired investment horizon, which asset should one buy to earn the highest return?".

It is worth highlighting that this question differs from the one that a large body of academic work is focused on. Instead of predicting a future price movement of an individual asset, we believe that a more useful insight, at least from the perspective of an average investor, can be extracted by focusing instead on the relative performance of a groups of assets. Our theory is that by not explicitly trying to model the price dynamics of the asset, we may be able to soften the data and compute requirements, thus making this more useful to a wider group of investors.

The rest of the report is divided in three sections: we discuss the data used in this work, followed by a description of the model architectures that we have explored and the experiments that were conducted. We close with the discussion of the results and ideas for future research.

DATA

In order to be consistent with the objectives of this project, as outlined in the Introduction section, we have made certain assumptions about the data. As we did not intend on using any other information apart from the historical prices, we assumed that market is efficient meaning that all available information is already factored into the market price. Therefore, we mainly looked at highly liquid markets, like large-cap equity indices and major currency pairs.

We used Yahoo Finance as a source of the market data for two reasons. Firstly, it is one of the most widely used free financial market data portals and secondly, we wanted to leverage Pandas existing Yahoo API that allowed us to implement the data download directly into the data pipeline[1]. We had to correct for any gaps due to differing trading day schedules by forward filling from the last available data point and removing any days with no prices for the majority of assets. Next, we converted prices to log returns, as is a common practice when dealing with financial time series. We did not explicitly consider transaction costs in the analysis as they would not have an impact on the selection of the architecture.

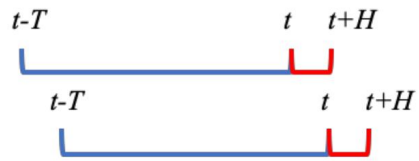
To create a dataset, let us define the following hyper-parameters:

- number of assets, N
- size of the look back window, T
- size of the investment horizon, H

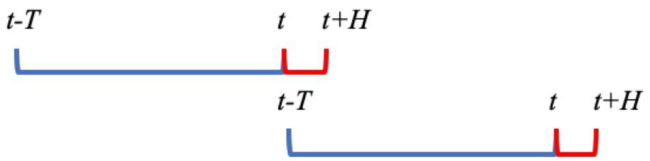
Then at time t, the training example $(X_t^{[train]}, Y_t^{[train]})$ can be represented as following:

$$X_t^{[train]} = \begin{cases} X_t^{(1)}, X_{t-1}^{(1)}, X_{t-2}^{(1)}, \dots, X_{t-T}^{(1)} \\ X_t^{(2)}, X_{t-1}^{(2)}, X_{t-2}^{(2)}, \dots, X_{t-T}^{(2)} \\ \dots \\ X_t^{(N)}, X_{t-1}^{(N)}, X_{t-2}^{(N)}, \dots, X_{t-T}^{(N)} \end{cases} \qquad Y_t^{[train]} = \underset{l:l\in[1,N]}{argmax} \sum_{i=1}^{H} X_{t+i}^{(l)}$$

In order to maximise the amount of training data, we have adopted a sliding window approach, however we have also explored the staggering of examples. While using the sliding window results in high correlation between the examples, using the other approach was reducing the size of the training set too much, especially for larger values of T. In practice, we did not observe a significant improvement in the models as a result of using the staggering method. Instead we shuffled the sliding window examples to avoid the model overfitting the first few batches. The diagrams below explain the differences between the two approaches.



sliding window example



staggered window example

For the purposes of exploring various model architectures, we built a dataset for three major equity indices (FTSE100, S&P500 and Nikkei225) with 28 years of historical data. In total, the dataset contains 22,500 data points. We allocated 95% to the training set and 5% to the test set.

[1] We have drawn inspiration from https://github.com/zpf4934/-stockPrediction and https://github.com/cs230-stanford/cs230-code-examples as examples of data pipeline implementations

We decided to forego the dev set and use part of the test set to assess different model configurations.

As we are concerned with the relative position of the best performing asset, we convert $Y_t^{[train]}$ and $Y_t^{[test]}$ to one-hot vectors as described below.

$$Y_t^{[train]} = \{-0.0109 \quad 0.0087 \quad 0.0235\} \rightarrow \{0 \quad 0 \quad 1\}$$

We would like to note that the choice of seeking the top performing asset was an arbitrary, albeit most desirable, decision. The task can be generalized to search for asset with a particular rank, thus creating a ranking list. We summarise the dimensions of the dataset that is used to demonstrate performance of the models below, together with some statistics.

Hyper-parameters	N=3, H=100, T=1	
Training set	X:(7025,300)	Y:(7025, 3)
Test set	X:(274, 300)	Y:(274, 3)
Class probability split	(1) 28.9% | (2) 32.1% | (3) 39.0%	

MODELS

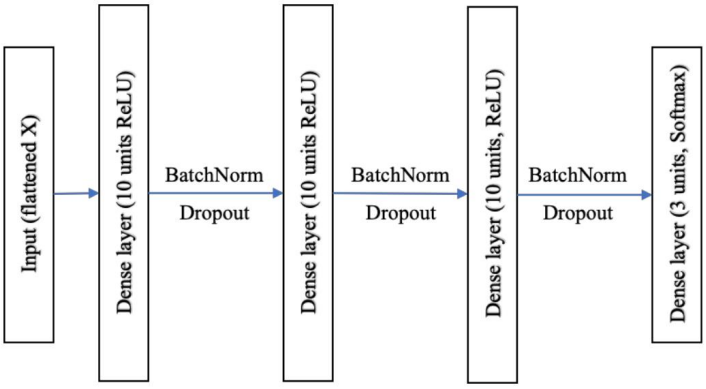
We have explored a number of architectures suitable for this type of a supervised multi-class classification problem. We briefly describe the key ones below before presenting and discussing the results in the next section.

Naïve model

This is a very basic model that is widely used in finance. The predictions for the top performing asset over the investment horizon H are based on the average performance over the look back window T. In effect, the model is attempting to exploit the persistence of trends in the financial markets, i.e. "winners continue to be winners". We have optimized the lookback window over the training set and fixed it over the test set in order to keep the conditions comparable to other configurations.

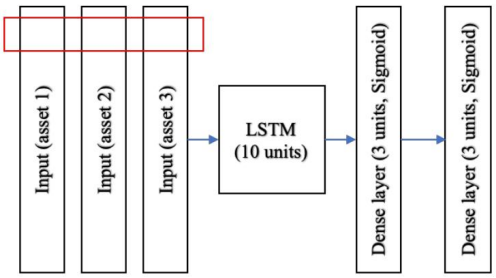
Multi-layer fully-connected network

The first neural network we considered consisted of stacked fully-connected layers. The main challenge with this type of network architecture is that it was very prone to overfitting the training dataset. We had to use a stack of layers with a relatively small number of units with BatchNorm and Dropout in between. An example of such architecture is presented below. Even such a simple construction was overfitting the data after a certain number of epochs. We have experimented with different activation functions, but we found that ReLU was producing the most consistent results.

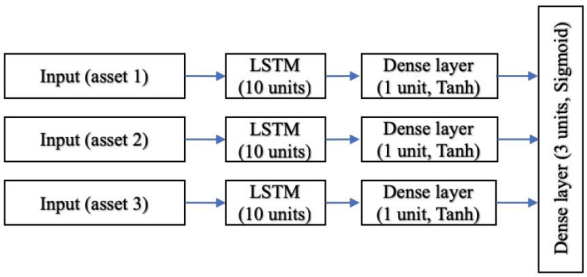


Recurrent network

Recurrent neural networks seem like a natural choice for sequential data and we use LSTM as a prime example. We believe Because it can be used for different types of classification and prediction problems (e.g. many-to-one, many-to-many) we have considered two types of architectures.



Model 1



Model 2

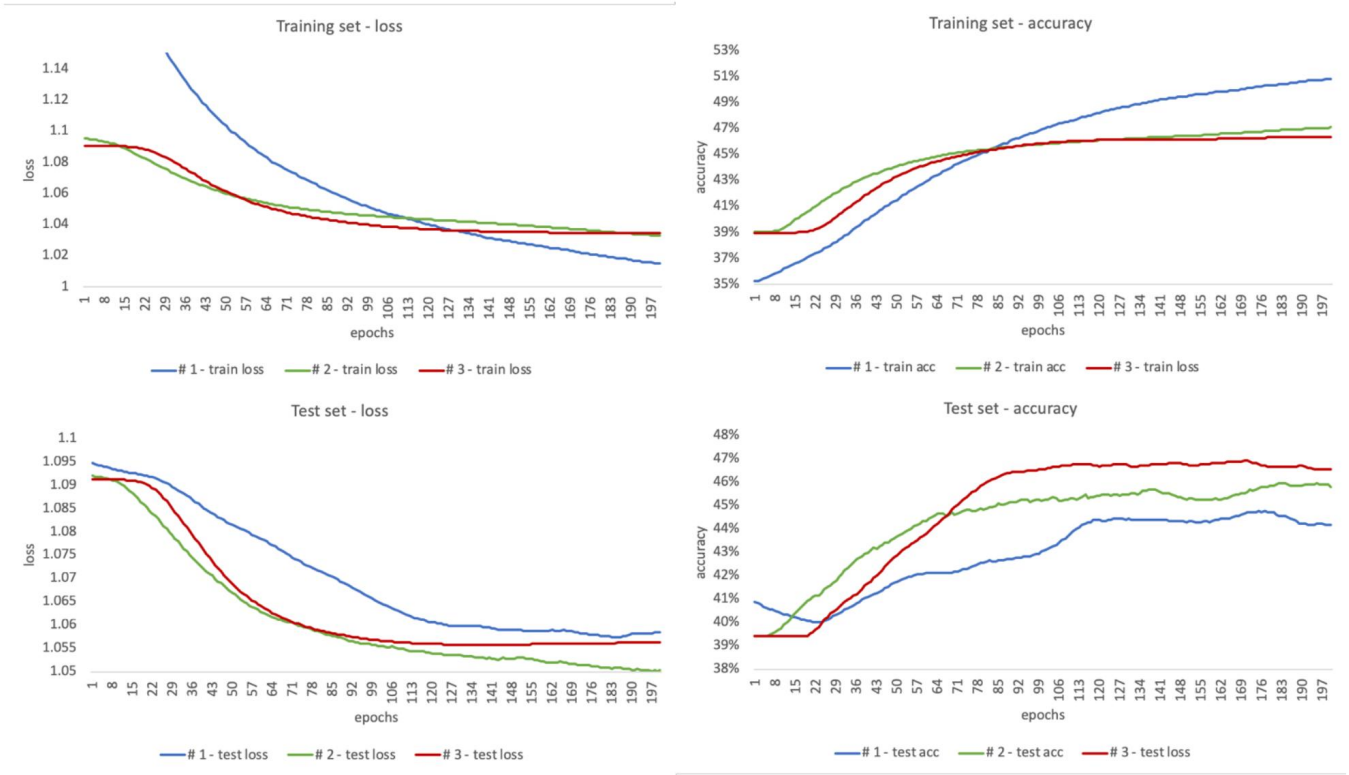
Firstly, we pass the input data as one array, where each asset represents a "feature" dimension, meaning that in our experiment the input to LSTM has a shape (7025, 100, 3). The output of LSTM is further processed via a Sigmoid activation before being turned into a probability distribution with a Softmax layer.

The second approach uses LSTM layer for each asset independently. The idea here, inspired in part by [5], is to almost create a hybrid model that is performing a regression and classification tasks at the same time. The LSTM fits the data to predict the next time step in the sequence, while the fully-connected layers fit the predictions to the labels. We have explored a number of different configurations of this nature, include multi-input multi-output format, but we did not find an optimal parameter set to boost the performance proportionately in line with the increase in complexity. We have also experimented with Convolutional layers, but we chose not to present the findings in this report as the results were underwhelming.

RESULTS

For the majority of the experiments, we have selected a cross-entropy loss function and minimized it using Adam algorithm. We have tested other optimizers, like Adadelta, but we did not find a meaningful impact on the performance. We used categorical accuracy as an appropriate metric for the multi-class classification problem. We summarize the key results after 200 epochs in the table below.

#	Model	Loss		Accuracy	
		Train	Test	Train	Test
0	Naive	N/A	N/A	35.84%	29.37%
1	Fully-connected	1.015	1.059	50.83%	44.15%
2	LSTM 1	1.033	1.050	47.07%	45.82%
3	LSTM 2	1.034	1.056	46.36%	46.61%



- All Neural Network models outperform the naïve model, although overall achieved test set accuracy is arguably underwhelming.
- Each model demonstrates an incremental improvement over the other, however the behaviour clearly varies when trained for the same number of epochs.
- Models #2 and 3 appear to have stopped fitting the training set quite early on, implying that either the learning rate is too low or the network lacks depth to capture the entire complexity of the relationship between returns and multi-class labels
- Model #1 on the other hand is clearly more prone to overfitting, so further training is likely to decrease the test accuracy.
- There is clear evidence that more data would benefit all considered approaches, however it also appears that relying solely on historical data may be putting an implicit cap on the predictive power of these models.

FUTURE RESEARCH

There are several directions of further research that we believe are worth pursuing next. Firstly, there is a lot more that can be done with regards to the tuning of the hyperparameters. In fact, establishing the appropriate lookback window for a particular investment horizon can be a study in its own right. In addition, a thorough analysis of the relationship between the behaviour of the model and the number of assets being considered needs to be carried out.

Another interesting direction would be to apply learning-to-rank algorithms used in the area of information retrieval. One can think of ranking assets as ranking documents in terms of optimal relevance to a particular query. A lot of progress has been made in recent years to improve the performance of these algorithms, specifically for list-wise problems. There have even some attempts at applying this in finance [6]. Moreover, at the time of writing this report, Google has released a dedicated TensorFlow library for ranking algorithms [7], which can facilitate a more efficient training using GPUs.

REFERENCE

Contributors: the author of this report is the sole contributor
GitHub repository: https://github.com/ant-po/CS230

[1] R. Cont, J. Sirignano, 2018. Universal features of price formation in financial markets: perspectives from Deep Learning
[2] Y. Peng, H.Jiang, 2015. Leverage Financial News to Predict Stock Prices Movements Using Word Embeddings and Deep Neural Networks
[3] M. Langkvist, L. Karlsson, A. Loutfi, 2014. A review of unsupervised feature learning and deep learning for time-series modelling
[4] M. Dixon, D. Klabjan, J. Hoon Bang, 2016. Classification-based Financial Markets Prediction using Deep Neural Networks
[5] T.Kimoto et al, 1990. Stock market prediction system with modular neural networks
[6] Q. Song, A. Liu, S. Yang, 2017. Stock portfolio selection using learning-to-rank algorithms with news sentiment
[7] R. K. Pasumarthi et al, 2018. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank