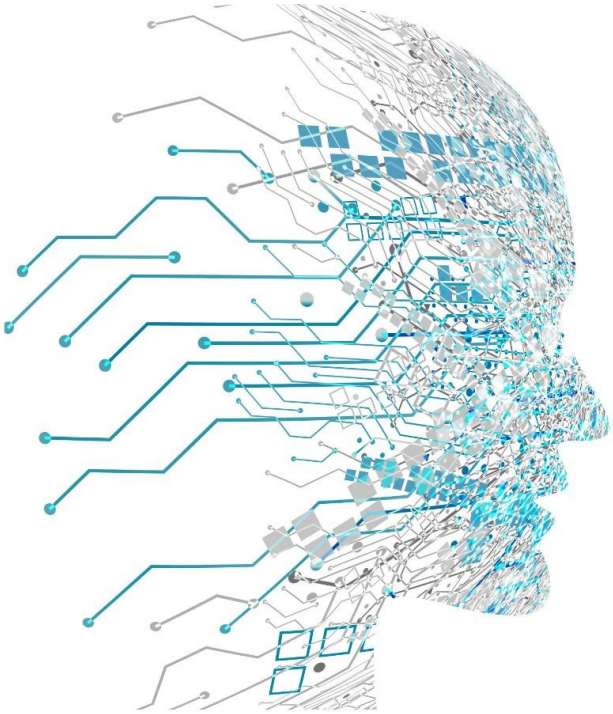
Stockpulse

Deep Learning on Emotional Data for Daily Long / Short Decisions



Research Report 01/2021

Introduction

In the past few years, machine learning has become a popular application in various fields. One of the core concepts used are neural networks. A neural network is a computational system that loosely models the human brain in order to solve classification problems. Chaining multiple neural nets is called "deep learning", with "deep" referring to the potentially large number of layers in these kind of models. Our aim of using deep learning is to recognize hidden patterns which potentially influence the oscillation of equity prices.

Data

In this report, we use emotional data from our Buzz/Sentiment dataset and combine them with prices from our daily quotes dataset. The analyzed period starts in August 2012 and spans until August 2020. This period is divided into three parts for training, validation, and testing. Data before 2018 is used to train the model. Learning progress is validated with data from January 2018 to March 2020. Data starting from April 2020 is used for testing, once the training of the model has concluded.

Input Data

Sentiment and buzz values are used as input data (in the following, we will also refer to input data as features). We compute the relative change of close prices in times t and t_{-1} as our target label. The price change from t_{-1} is also used as a feature. For example, sentiment and buzz values of 2020-07-02 are used together with the price change from 2020-07-01 to 2020-07-02 to predict the price change of 2020-07-03. In addition, we optionally include momentum of close prices as an additional feature while training the model. The calculation works the same as the for the relative price change, but instead of t_{-1} a different reference is used.

Output Data

The output data from the daily model is a set of signals, which contains the dates, titles and positions. The signal dates exclude weekends and regional holidays, and positions can be buy, sell, or flat.

Deep Learning

In this section, we briefly describe our approach of finding a well-working deep neural network and the setup of the training and validation process. Neural networks have many parameters which influence their learning capacity and predictive power. It is important to make "good" choices for these parameters in order to find models which produce well-working signals. Input data can also be structured in different ways, adding another layer of complexity to the optimization problem at hand.

As the number of possible configurations for neural networks is very large, a suitable method is required to find "good" configurations in an acceptable amount of time. While our data center is equipped with nVidia GPUs to maximize training speed, it is still necessary to reduce the number of configurations significantly to bring down the necessary amount of time.

Genetic Algorithm

To do that, we use a heuristic approach called genetic algorithm. A genetic algorithm is inspired by Charles Darwin's theory of evolution. It follows the principle of natural selection where the fittest individuals are selected to produce offspring for the next generation. The following items are four essential processes in a genetic algorithm:

- Selection: a process of finding fittest individuals in a population. The purpose is to keep genes from these individuals through inheritance to the next generation.
- Fitness: a rating of how individuals perform. In our experiment, the fitness score is determined by financial key statistics like Sharpe ratio, volatility, or return on investment.
- Reproduction: a process of producing offspring. Pairs of individuals (parents), chosen during the selection step, are creating offspring from their genes. In our experiment, optimized parameters are the genes.
- Mutation: a process of mutating genes during reproduction to avoid falling into local optima. In our experiment, the possibility of mutation is 20%.

Implementation

We apply the genetic algorithm to optimize the neural net's structure. We implement the entire procedure in Python with machine learning library Tensorflow. The genetic algorithm runs through 10 generations with a population of 50 networks per generation. The initial population of neural networks is created with randomly chosen layouts, which can also have different kind of input parameters, e.g. the length of input history can vary. The following parameters are tuned by the genetic algorithm:

- Size of history fed into the model
- Type of neural network layer (Dense, SimpleRNN, LSTM)
- Number of neurons per layer
- Number of layers
- Activation of layer (linear, elu, relu, tanh, sigmoid)

Network Layout

In general, a neural network consists of nodes (neurons) and edges (connections between nodes), so it can be seen as a graph. There are many different configurations of neural networks. We build neural networks via Tensorflow's

Keras module, which constructs the networks by defining layers. Each layer passes its output into the next layer, thus defining the graph. For more information about Tensorflow and Keras refer to https://www.tensorflow.org.

Recurrent Neural Networks

Different types of layers are available in Keras, for instance the Recurrent Neural Network layer (SimpleRNN). Recurrent neural networks (RNNs) are able to learn and represent temporal sequences, thus this type of layer is particularly fit for solving time-related problems, such as forecasting weather or periodic demands. In addition to SimpleRNN, we also make use of Long short-term memory (LSTM)[1] layers, which is another architecture based on RNN.

Conceptually, we have defined four stages in our neural network: recurrent stage, squeeze stage, dense stage and prediction stage. These stages may have individual layer types, number of layers, numbers of neurons, as well as activation and regularization functions. The following table is an example of the model summary output from Tensorflow (Keras):

Layer (type)	Output Shape	Param #
recurrent_stage (LSTM)	(None, 1, 24)	2688
recurrent_stage_1 (SimpleRNN)	(None, 1, 128)	19584
recurrent_stage_3 (SimpleRNN)	(None, 8)	1096
dense_stage (Dense)	(None, 8)	72
prediction_stage (Dense)	(None, 1)	9

Total params: 23,449
Trainable params: 23,449
Non-trainable params: 0

Training

Each neural network is trained using the Nadam optimizer with mean squared error as the loss function. We apply dropout and regularization at various stages in the network to avoid overfitting. We train until loss is stable for a certain number of epochs. Once training is finished, the performance is evaluated by producing signals for the validation data and feed them into our proprietary backtest platform.

Model outputs are floating point values, predicting the relative price change for a given feature vector. Thus, we need to derive the actual signal label (buy, sell, flat) from these continuous values. To do this, we apply threshold based classification, so that for values smaller than the threshold, signal is sell, and for values larger than the threshold, signal is buy. Optionally, we define a quantile around the threshold. Values falling inside the quantile will be labeled as flat. We optimize the threshold through sensitivity analysis.

1 http://www.bioinf.jku.at/publications/older/2604.pdf

Validation

During sensitivity analysis of the threshold, each set of signals is run through the Stockpulse backtester. The backtester performs a complete trading simulation, considering transaction cost, dividends, trading hours, etc. The result is a comprehensive set of financial performance keys, such as cummulative return, return p.a., volatility, maximum drawdown, Sharpe ratio, Sortino ratio, and more. In addition to these financial metrics we also calculate the hit ratio for each signal set. We keep the threshold with the best result during the sensitivity analysis.

When evaluating model performance, we look at different performance criteria which, when all fulfilled together, will label a model as a "good" model. Models not fulfilling the criteria are considered "bad". The criteria are as follows:

- Sharp ratio equal or larger than 1
- Hit ratio of buy signals equal or larger than 0.51
- hit ratio of sell signals equal or larger than 0.51
- Overall hit ratio larger than 0.53
- Each year's return equal or larger than 5%

Inside the good and bad model classes, models are ranked by Sortino ratio. In the genetic algorithm, we make use of the model classes during selection. This is done by keeping the best 40 percent of the population in the selection step, with good models being considered before bad models.

Testing

Only when a model has found to be in the class of good models, by backtesting on the validation dataset, we run an additional backtest, but this time on the test dataset. In case the signals produced for the test dataset fulfill the criteria for good models as well, the model is stored and marked as being in the class "good". That means, that models have to fulfill all criteria with signals from both datasets, validation and test, in order to be considered as "good".

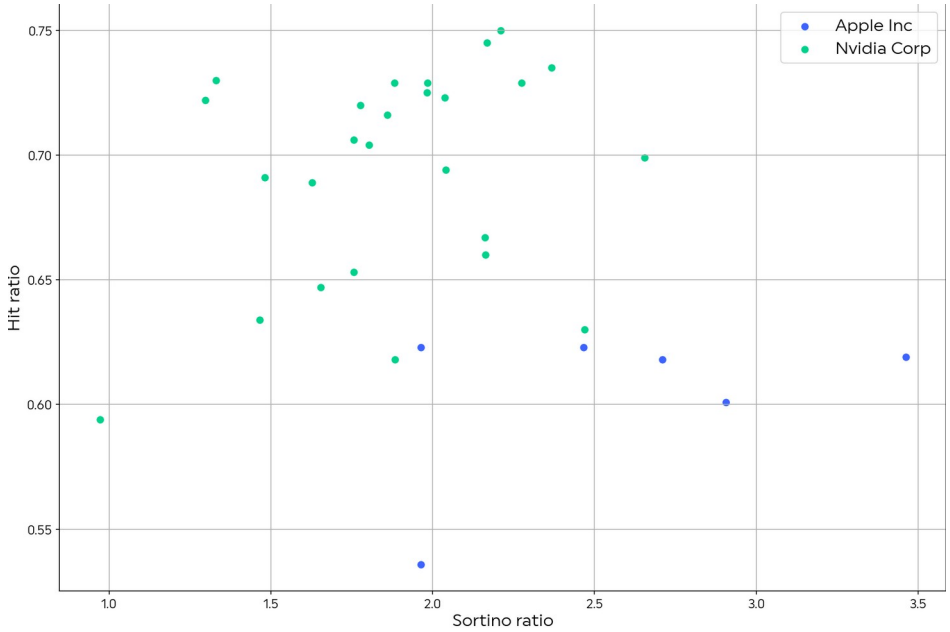
Results

We consider the common stocks of Apple Inc. (ticker: APPL) and NVIDIA Corp. (ticker: NVDA) as examples for how the genetic algorithm trains a population of neural networks. In general, we observe that the average performance of the population is improved from generation to generation. Eventually, the performance converges to a maximum.

The next table shows the top ten models by Sortino ratio, found by evolution of 10 generations with population 20 for NVIDIA Corp. and Apple Inc. Comparing the structure of those models, the average number of layers is four. Besides the number of layers, the type and activation of layers also influence its performance.

Stock	Hit Ratio	Sortino Ratio	Vola	No. Recurrent	Has Squeeze	Has Dense	Activation	Depth
AAPL	0.619	3.462	0.331	1	1	1	tanh	3
AAPL	0.601	2.906	0.3	3	0	0	tanh	3
AAPL	0.618	2.71	0.345	1	1	1	tanh	3
NVDA	0.699	2.655	0.507	2	0	1	elu	3
NVDA	0.63	2.469	0.497	4	0	1	tanh	5
AAPL	0.623	2.466	0.289	4	0	0	linear	4
NVDA	0.735	2.367	0.514	3	0	1	elu	4
NVDA	0.729	2.275	0.502	4	0	0	elu	4
NVDA	0.75	2.21	0.508	4	0	1	elu	5
NVDA	0.745	2.168	0.513	3	0	1	elu	4

The figure below plots the relation between hit ratio and Sortino ratio. Each point represents a model. There is no obvious correlation, however, some models for AAPL achieve higher Sortino ratios and beat NVDA for maximum Sortino ratio as well. NVDA in return sees a larger number of good rated models in total, with higher values for hit ratio.



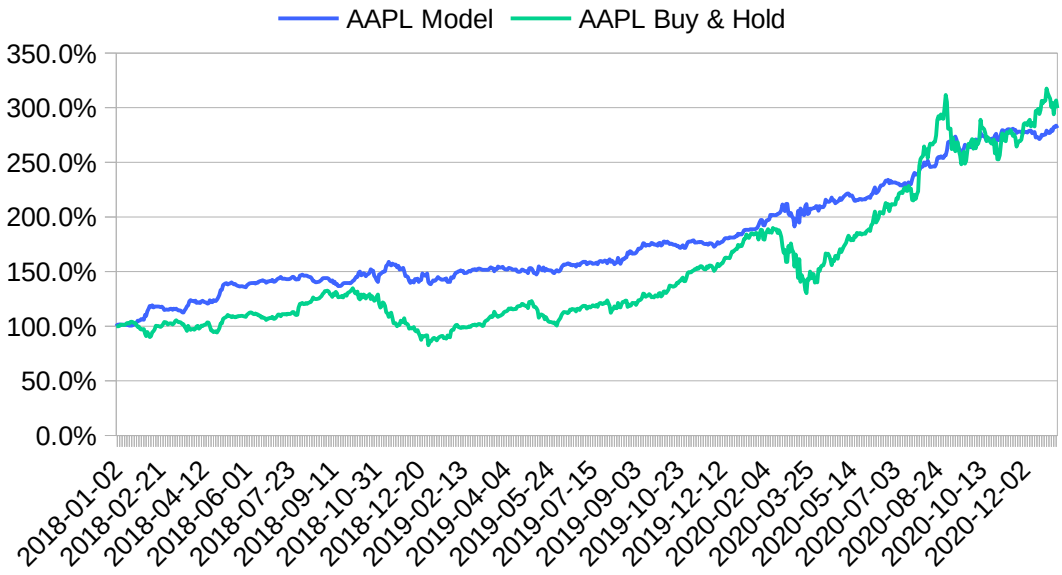
Performance Evaluation

In this section we compare the performance of our models with the market. From all found models, we choose the model which has the highest performance inside the good model class for this purpose.

AAPL – Apple Inc.

We start by considering the result for a model trained for making predictions for AAPL. For backtesting, the trading time is set to run between 08:00 and 17:00, timezone America/New York. The first figure displays historic performance for our model and the market performance, proxied by a simple “buy and hold”

strategy for AAPL. We display the period for the validation and test datasets, spanning from 2018-01-02 to 2021-01-12.



Financial key performance statistics are given in the next table. We can observe that, although cummulative returns are in the same level, volatility is less than half of the simple buy and hold strategy. The model seems able to make long and short decisions in a meaningful way. This is confirmed by close to zero values in beta and correlation, which indicate that there is no strong relation between the market and the model.

	Cum. Ret	Ret p.a.	Ret/D	Vola p.a.	Down Vola p.a.	Max DD	Sharpe	Sortino	Beta	Correl	R^2
AAPL Model	181.24%	34.14%	0.137%	17%	11.42%	12.89%	1.99	2.97	-	-	-
AAPL Buy & Hold	199.54%	40.6%	0.162%	34.34%	24.59%	38.73%	1.18	1.64	-0.02	-0.03	0

The statistics figure displays information about the model's hit ratios, overall and split into buy and sell. The model achieves a higher hit ratio for buy signals, which can potentially be explained with the general uptrend for AAPL during the analyzed period.

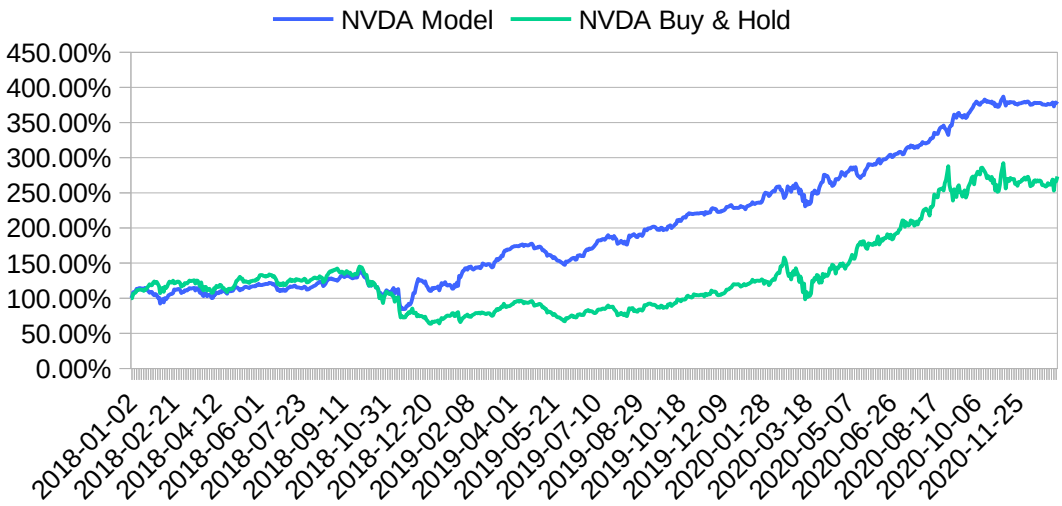
Total Trades	193	**Average Holding Time**	6740min
Profit Trades (% of total)	115 (59.59%)	**Loss Trades (% of total)**	77 (39.9%)
Long Trades (won %)	100 (67%)	**Short Trades (won %)**	93 (51.61%)
Largest Profit Trade ($)	15.08% (150752.03)	**Largest Loss Trade ($)**	-10.56% (-105591.34)
Avg. Profit Trade (%)	29910.48 (2.99%)	**Avg. Loss Trade (%)**	-21133.3 (-2.11%)
Avg. Max Virt. Profit (Profit Trades)	+3.74%	**Avg. Max Virt. Profit (Loss Trades)**	+0.63%
Avg. Max Virt. Loss (Profit Trades)	-0.47%	**Avg. Max Virt. Loss (Loss Trades)**	-2.44%
Avg. Diff to Max Profit (Profit Trades)	0.75%	**Avg. Diff to Max Loss (Loss Trades)**	-0.32%

NVDA – NVIDIA Corp.

Next, we compare our nvidia daily L/S model in the same way, shown in the next three figures. The model is using a threshold quantile and a history size of 100

days, two stacked recurrent layers followed by a dense layer with 1024 neurons and the final prediction stage with activation elu. We compare the model performance with the alternative simple investment approach of buy and hold.

The



following table shows the overall key performance statistics. Metrics are calculated for the combined periods of validation and test: 2018-01-01 to 2021-01-12. We can see that our model beats the simple alternative investment strategy for NVDA. However, a beta value of .34 and a correlation value of .54 mean that the model will resemble market phases, to a degree. This is confirmed visually looking at the chart of the historic performance.

	Cum. Ret	Ret p.a.	Ret/D	Vola p.a.	Down Vola p.a.	Max DD	Sharpe	Sortino	Beta	Correl	R^2
NVDA Model	280.18%	46.97%	0.188%	30.59%	22.11%	38.63%	1.53	2.11	-	-	-
NVDA Buy & Hold	173.32%	43.42%	0.174%	48.44%	35.8%	56.08%	0.89	1.21	0.34	0.54	0.29

The next table gives the trade statistics for the NVDA model, as calculated by our backtester. Looking at the average holding time we can observe, that although the model is trained to give daily predictions, signal direction stays stable for approximately one week (~12k minutes). We also observe that return for loss trades is less than the return for profit trades.

Total Trades	133	**Average Holding Time**	11967min
Profit Trades (% of total)	93 (69.92%)	**Loss Trades (% of total)**	40 (30.08%)
Long Trades (won %)	67 (73.13%)	**Short Trades (won %)**	66 (66.67%)
Largest Profit Trade ($)	17.44% (174310.5)	**Largest Loss Trade ($)**	-21.45% (-214529.7)
Avg. Profit Trade (%)	46162.2 (4.62%)	**Avg. Loss Trade (%)**	-37282.98 (-3.73%)
Avg. Max Virt. Profit (Profit Trades)	4.98%	**Avg. Max Virt. Profit (Loss Trades)**	1.44%
Avg. Max Virt. Loss (Profit Trades)	-1.35%	**Avg. Max Virt. Loss (Loss Trades)**	-7.13%
Avg. Diff to Max Profit (Profit Trades)	0.36%	**Avg. Diff to Max Loss (Loss Trades)**	-3.4%

Yearly Model Rebuilding

In general, training with more historical data enables neural networks to learn more and achieve higher predictive values. However, we have observed a

different pattern for some stocks, where the model performs well in the past but less so in recent years. Assuming data has its own half-life, as patterns in communication slowly change over time, we hypothesize more recent data is likely to be more relevant.

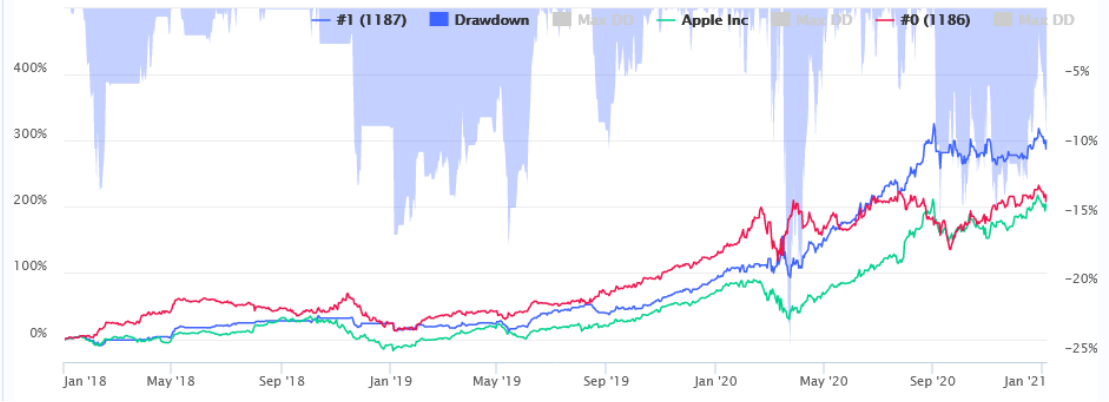
Thus, we implement another model building style, which trains with data comprising four years, validates with the following year and tests with the year following the validation year. Then, the start year is moved one year to the future and the process is repeated, until the test year reaches the current year. We call this adjusted model building style "yearly model rebuilding".

The next table briefly compares performances of our models and the market, proxied through the simple buy and hold strategy for AAPL. Our model beats the market performance at every period in this case.

	Trained Year	**2012 - 2015**	**2013 - 2016**	**2014 - 2017**	**2015 - 2018**
	Test Year	**2016 - 2017**	**2017 - 2018**	**2018 - 2019**	**2019 - 2020**
Cum. Return	Model	171.61%	101.06%	81.28%	374.18%
	Apple Inc.	60.63%	34.5%	70.48%	252.06%
Return p.a.	Model	40.93%	28.95%	26.21%	67.34%
	Apple Inc.	15.7%	10.76%	18.65%	43.07%
Vola p.a.	Model	13.23%	12.82%	15.76%	25.56%
	Apple Inc.	16.15%	18.59%	21.52%	29.12%
Max DD	Model	7.94%	12.95%	17.71%	22.07%
	Apple Inc.	19.38%	36.73%	38.73%	31.43%
Sharpe ratio	Model	3.07	2.24	1.65	2.62
	Apple Inc.	0.96	0.57	0.86	1.47
Sortino ratio	Model	4.94	3.68	2.38	3.81
	Apple Inc.	1.38	0.8	1.19	2.13

Besides the market performance, we also compare the performance of models trained with entire history and with the nearest 4 years. Signals of yearly rebuilding models are assembled signals from the model which has the best Sortino ratio in the year.

Test	Cum. Return	Return p.a.	Return/D	Vola p.a.	Downside Vola p.a.	Max DD	Sharpe	Sortino	Alpha	Beta	Correl	R^2
#0 (1186)	218.52%	35.72%	0.143%	30.11%	20.98%	33.4%	1.18	1.69	-	-	-	-
#1 (1187)	300.73%	41.21%	0.165%	27.04%	18.79%	24.27%	1.51	2.18	-	-	-	-
Apple Inc	204.04%	26.37%	0.105%	27.52%	19.37%	38.73%	0.95	1.35	51.45%	0	0	0
#0 (1186)	218.52%	35.72%	0.143%	30.11%	20.98%	33.4%	1.18	1.69	48.79%	0.05	0.06	0



In the above figure, the group 1186 represents the model trained with entire history and the group 1187 represents the yearly rebuilding model. Both models outperform the market performance (the green line). Further comparing these two models, we can see that the yearly rebuilding model performs better from May 2020. It is believed that the yearly rebuilding model is more flexible to recent situation.

Discussion

The optimization strategy presented in this report is a rather simplistic one, which does not examine more complex network structures. This is due to the sequential nature of the built networks, which do not allow parallel branches of data streams, which restricts the possible layouts of the networks. However, more sophisticated layouts may yield better predictive power.

We also have not considered other allocation cycles, like weekly or quarterly signal generation. A combination of a short-term micro-layer, which is built from daily models, with a longer-term macro-layer, which could be built from weekly models, could potentially improve pattern recognition of the neural networks.

It is clear that the two models presented in this study should not be run isolated but rather they should be used in a portfolio approach. As hit ratios are above .5 for the two analyzed instruments, we would consider building market neutral portfolio from a bucket of similar daily L/S models a viable approach.

We will look into these and other potential improvements of the deep learning process in another research report.