CS230

Attentive Neural Models for Algorithmic Trading

Will Geoghegan
will.geoghegan@stanford.edu

Abstract

Financial markets are inherently unpredictable. There has nevertheless been significant research into modeling these systems with deep neural networks. Attention mechanisms offer a way to improve these networks by conditioning outputs on the information most relevant to a specific input. We use the Simple Neural Attentive Meta-Learner model to predict stock returns from both technical and fundamental data on 7000 publicly traded US stocks over a 10 year window. We show that the stock market forecasting problem can be modeled as a meta-learning problem across different assets, and achieve significant above-market returns on an 8-month held-out testing window using a model that incorporates this causal attention mechanism.

1 Introduction

Countless statistical models have been developed for predicting and trading in financial markets. Many use purely technical indicators based on pricing, trade volume, and volatility information. Many are based on companies' underlying financial strength in the form of quarterly income statements. We combine both of these approaches, and use state-of-the-art attention mechanisms embedded in a deep neural network architecture to learn functions that prioritize the most relevant information in each prediction. By representing our problem as meta-learning where each assets represents a distinct but related learning problem, we are able to generalize very well to unseen data.

We take as input 20 days of price and broader market information as well as 8 quarters of income statements for a given asset, and output a recommendation to enter either a long or short position on that asset. We are able to evaluate the strength of our model by backtesting it on real sequences of stock prices and comparing the model's return to that of the entire market over the same time period. This metric of "beating the market" signifies whether a model is viable for real-world use.

2 Related Work

Learning-based financial modeling. There have been many learning-based approaches to financial modeling and algorithmic trading. Deng et. al., (2017) use a deep RL model for real-time trading. [3] They show that it is possible to generate a tradable signal from a learned function on financial data. We choose not to use a reinforcement learning approach because, while we implement a simple trading algorithm for backtesting purposes, the input to our model at time t does not depend on our trading decision at time $t-1$.

The majority of approaches in this domain focus on technical indicators and sentiment information, as in Vargas et. al., (2017), and on short-term time horizons – either daily or intraday. [11] By leveraging quarterly financial reporting and looking at longer time intervals, our model attempts to learn signals created by companies' underlying strengths and weaknesses rather than purely technical ones.

CS230: Deep Learning, Spring 2019, Stanford University, CA. (LaTeX template borrowed from NIPS 2017.)

Attention and meta-learning. The Simple Neural Attentive Meta-Learner, introduced by Mishra et. al. in 2017, combines temporal convolutions with causal attention to generalize learning across related tasks. [8][12] This is of interest because our problem can be thought of as learning a slightly different function on each asset in our dataset; given identical market conditions and previous price movement, different assets will perform differently based on a multitude of factors such as the sector to which the assets belong and the size of the companies. SNAIL has been successfully applied to both reinforcement and supervised learning; here we focus on the supervised version. Attentive Neural Processes (Kim et. al. (2019)) are a form of neural process (Garnelo et. al. (2018)) that incorporate attention mechanisms. We experimented with ANPs in our model but achieved better results with SNAIL. [6][4]

3 Dataset

Source. We source our data from Quandl's Sharadar Core US Equities and Fund Price datasets. These provide daily closing price, price/earnings, price/book, and price/sales ratios as well as trade volume and quarterly financials for roughly 7000 publicly traded US stocks over the past 10 years (since August 2009).

Structure. Each example $\mathbf{x}^{(i)}$ consists of a pair of inputs $(\mathbf{x}_t^{(i)}, \mathbf{x}_f^{(i)})$. $\mathbf{x}_t^{(i)} \in \mathbb{R}^{20\times 23}$ represents the price and trade volume of a specific stock as well as those of certain market indices over the last 20 trading days. $\mathbf{x}_f^{(i)} \in \mathbb{R}^{8\times 210}$ represents 210 features of the company's reported financials (e.g., cash flow, debt, assets, tax liabilities) over the last 8 quarters.

Augmentation and Preprocessing. We only have one data point per roughly 90 days per stock for all of the fundamental features, so we augment our data by propagating these features forward until the next quarter (i.e., $\mathbf{x}^{(i)}$ will share quarterly features with $\mathbf{x}^{(i+j)}$ provided that $j < k$, where k is the number of trading days in a quarter). After removing non-numerical features and removing examples for stocks with fewer than 20 data points, we end up with 995287 total examples.

Training/validation/test split. Our training set consists of 957527 examples corresponding to dates between 1 August 2009 and 1 July 2018. The validation and test sets consists of 37760 examples corresponding to dates between 1 July 2018 and 11 March 2019. We randomly divide these between validation and test sets.

Labeling. We experimented with labels $\mathbf{y}_d$ where $y_d^{(i)}$ is equal to the future return of the ith stock-date pair at a time d trading days in the future. Choosing d involves a tradeoff between indivudal asset signal being lost to short-term noise versus long-term market trends: From our experiments, $d = 32$ resulted in the best returns for the baseline model. This corresponds to roughly 6 calendar weeks.

4 Methods

4.1 Weighted Cross Entropy Loss

As discussed above, our labels $\mathbf{y}$ represent the return of each asset in $\mathbf{X}$ 32 trading days in the future. Rather than modeling our problem as a regression, however, we want to learn a classifier indicating whether we should adopt a long, short, or neutral position. [1] In order to speed up learning, we ignore the neutral option in training, which reduces our problem to a binary classification.

Since we want to preserve the magnitude of each asset's movement as well the direction, we use for our loss function a weighted version of cross entropy:

$$\mathcal{L}_{weighted}(\mathbf{y}, \mathbf{b}, \hat{\mathbf{y}}) = -\frac{1}{m}\sum_{i=1}^{m} |y_i| b_i \log \hat{y}_i \tag{1}$$

where $y_i \in \mathbb{R}$ is the true return label vector, $b_i \in \{0, 1\} = y_i > 0$, and $\hat{y}_i \in [0, 1]$ is the output of the model for example i. The result of this is a binary classification corresponding to upward or downward price movement where each example is weighted by the magnitude of the movement.

4.2 SNAIL

In the supervised application of the SNAIL model, the label y_t is predicted from inputs x_t and the sequence $(x_1, y_1), \ldots, (x_{t-1}, y_{t-1})$ for time steps $1, \ldots, t-1$ by passing the inputs through several causally masked temporal convolution blocks and attention layers. We employ the original SNAIL paper's architecture of two temporal convolution blocks followed by a causal attention layer, with this pattern repeated twice.

A temporal convolution block consists of $\lceil \log_2 t \rceil$ 1D convolutional layers, where t is length of the input sequence, and the ith layer within each block uses kernel size $k = 2$ and dilation rate $d = 2^i$. [8] We include batch normalization and dropout layers before and after each temporal convolution layer respectively. [10] Additionally, the input to each convolutional layer is concatenated to the output, forming skip connections.

A causal attention layer performs key-value lookups based on self-attention. [8][12] For an input I, key and value sizes k, v, affine transformation f, and causally masked softmax operation s:

$$K = f(I, k) \tag{2}$$
$$Q = f(I, k) \tag{3}$$
$$L = QK^T \tag{4}$$
$$P = s(\frac{L}{\sqrt{K}}) \tag{5}$$
$$V = f(I, v) \tag{6}$$
$$R = PV \tag{7}$$

Attention layers return R concatenated to I as in the convolutional layers. The major difference in our implementation of SNAIL from Mishra et. al. (2017) is that we replace the final output $\hat{y}$ with a length 256 encoding vector v_f in the fundamental data module and v_s in the technical data module which are then concatenated and passed to the fully connected block.

4.3 Model

We will discuss results for both a baseline deep feedforward model and an experimental model incorporating parallel SNAIL modules on the fundamental and technical data inputs.

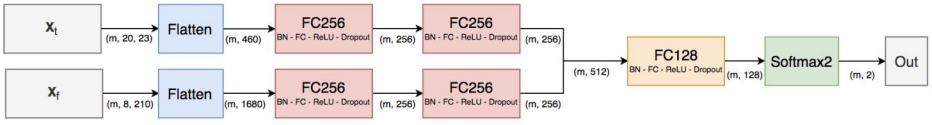


Figure 1: The baseline model.

Both models separately encode the technical data $\mathbf{x}_t^{(i)}$ and the fundamental data $\mathbf{x}_f^{(i)}$ of each input i as length 256 vectors. The baseline model does this by passing each through two fully-connected blocks with 256 hidden units. Each block consists of a batch normalization layer, a fully-connected layer, a ReLU activation layer, and a dropout layer. [5][9]

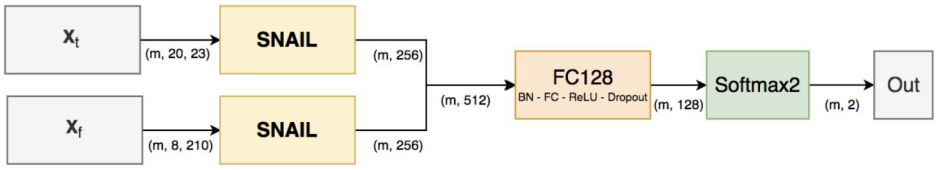


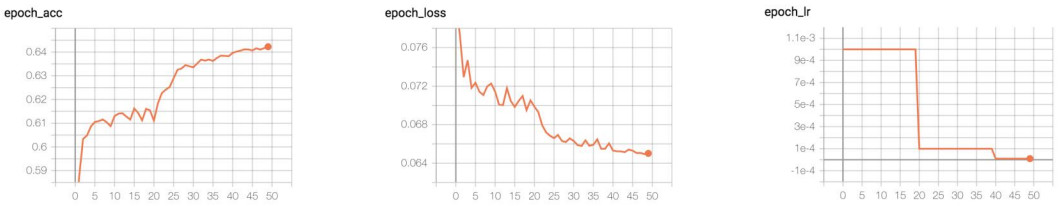
Figure 2: The experimental model, with SNAILs as defined above.

The experimental model passes $\mathbf{x}_t^{(i)}$ through SNAIL module S_t and $\mathbf{x}_f^{(i)}$ SNAIL module S_f, with each outputting a length 256 vector rather then a $\hat{y}$ prediction, retaining end-to-end differentiability. In both models, the output vectors are then concatenated and passed through another fully-connected block with the same structure and 128 hidden units. Finally, a 2-unit softmax layer outputs the class probabilities.

5 Experiments

5.1 Hyperparameter Selection and Tuning

Optimization. Our model is written in Python using the Tensorflow library. [2] This gives us a premade Adam optimizer that can be used out of the box. [7] We use the default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$, and use learning rate decay to tune the learning rate α, which we decay on a schedule from 10^{-3} to 10^{-5}. We use a batch size of 128, as larger batch sizes resulted in rapidly diminishing returns in training speed on our AWS instance.



(a) Training accuracy (b) Training loss (c) Learning rate

Figure 3: Training accuracy, training loss, and learning rate over 50 epochs on the baseline model. The effect of decaying the learning rate after epoch 20 can be seen clearly in both the accuracy and loss.

Regularization. Additionally, we use L_2 regularization and dropout throught our architectures to minimize overfitting. We sampled L_2 coefficients in the range $[10^{-6}, 10^{-3}]$ and found that 10^{-5} offered the best balance between low bias and low variance. Similarly, we tested drop rates in $[0.1, 0.5]$ and settled on a drop probability of 0.4.

5.2 Evaluation Metrics

Because we model our problem as a binary classification problem, accuracy is a passable metric. Accuracy, however, doesn't capture the magnitude of movements in price. The metric most indicative of our model's performance is the return it would actually generate when trading on our dataset. We calculate the return R on both the training and validation sets after each epoch,

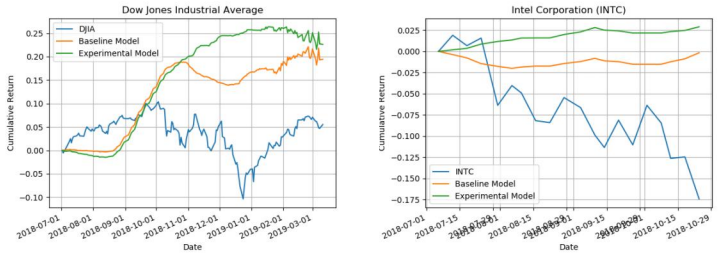
$$R(x, y, f) = \frac{1}{S} \sum_{s \in S} \prod_{t \in T} 1 + f(x_{st}, y_{st}), \tag{8}$$

where S is the set of stocks being evaluated and T is the set of trading days being evaluated for each stock in S. $f(x_{st}, y_{st}) = y_{st}$ if the model predicts the direction of x_{st} correctly, and $-y_{st}$ otherwise. R thus represents the mean return the model would achieve by predicting each stock in the dataset for each trading day and evenly diving some amount of capital among those positions.

5.3 Results

Model	Training Set		Test Set (all)		Test Set (S&P500 only)	
	Accuracy	Return	Accuracy	Return	Accuracy	Return
Market	N/A	183.68%	N/A	5.53%	N/A	3.49%
Baseline	**64.22%**	3717.28%	60.44%	19.52%	**55.10%**	**5.78%**
SNAIL × SNAIL	62.50%	**5652.70%**	**62.77%**	**22.67%**	51.75%	5.55%

Table 1: Attentive model compared to baseline model and the market.



(a) Models vs. DJIA (b) Models vs. INTC

Figure 4: Backtested test set performance on all stocks vs. the Dow Jones Industrial average, and backtested performance on Intel Corporation test data vs. INTC's real performance

5.4 Discussion

As seen in the above table, both neural models achieve large returns on the test set. The test set consists entirely of data corresponding to trading days that take place after the latest trading day in the training set, so there is no look-ahead bias involved.

Qualitatively, it can be seen in the chart the the baseline and experimental models perform quite similarly, except the experimental attentive model gains an edge in in volatile periods. This provides an explanation for why the baseline slightly models outperform on the S&P500 subset: this is a set of large companies that are less volatile than the rest of the stocks in the dataset.

We see some overfitting to the training set despite the regularization used, but not to the extent that the model fails to generalize to the test data. The overfitting we do see is likely due to each training example sharing index price and volatility features. Examples taken from the same day are highly correlated because of broad market forces, and since each one shares values the model can overfit specifically to these universal features.

6 Conclusion

Summary. We have shown that both a basic feedforward MLP and a hybrid attentive DNN can outperform the market in making medium-frequency trades. The attentive model outperforms the market on assets of all capitalization sizes. Modeling asset price prediction as a meta-learning task across assets leads to models which are able to generalize well both temporally and across assets.

Future work. Our return metric R is not weighted by confidence. By allocating capital among stocks based on confidence rather than uniformly, results could be improved without needing to change the model itself. Additionally, this model could be adapted to intraday trading. The attention mechanism seems well suited to picking out transient signals that can inform profitable trades. Finally, our model gave no consideration to trading fees. Since we only have one data point every four days for any given stock, this is not a huge factor, but it is a factor, and any market-ready model needs to be able to account for this.

Code. The project code can be found at `https://github.com/wdg3/attentive-neural-algo`.

Contributions

We would like to thank Jay Whang for his valuable feedback over the course of this project, especially regarding loss functions and project structure.

References

[1] Stock purchases and sales: Long and short. `https://www.investor.gov/introduction-investing/basics/how-market-works/stock-purchases-sales-long-short`.

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

[3] Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28:653–664, 2017.

[4] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes. *CoRR*, abs/1807.01622, 2018.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[6] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, S. M. Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *CoRR*, abs/1901.05761, 2019.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[9] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[10] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[11] Manuel Vargas, Beatriz Lima, and Alexandre Evsukoff. Deep learning for stock market prediction from financial news articles. pages 60–65, 06 2017.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.