Project Report : CS 7643

Sakae Watanabe
Gatech
swatanabe30@gatech.edu

Ashish Narasimham
Gatech
anarasimham3@gatech.edu

Michael Walker
Gatech
mwalker311@gatech.edu

Abstract

We apply state-of-the-art Deep Learning approaches to the problem space of short-term trading, using daily market open/close/volume data to create time-series batches of sequences within the dataset. We take 3 different approaches: Spatio-Temporal Transformer (STT), Liquid Time-constant Network (LTCN), and Identity Recurrent Neural Network (IRNN); and attempt to extract signal from high-volatility public company stock movements in the healthcare space. Our results indicate that further investigation is required to create nuanced models that balance buying, selling, and holding inferences to achieve lower training loss. However, we have been able to create a profitable trading strategy even with suboptimal trading accuracy.

1. Introduction

1.1. Background

"Everybody needs money, that's why they call it money!" - Mickey Bergman *Heist* 2001

Algorithmic trading can be defined as a method of investment analysis that seeks to use financial data with computer technology to find short-term abnormalities in market prices and exploit them to make a profit [1]. Attempts to use algorithms to monitor and exploit these systems have existed for many years and many attempts have been made to create highly profitable algorithms that can remove the human element from the trading process. There are many large firms that have financial interests in optimizing profit algorithmically, though, they tend not to share their techniques. This results in very few papers published surveying modern techniques with the state of the art being an adversarial model that hit around a 64% accuracy on their chosen stocks [2]. For this paper, we intend to test out a set of modern deep learning methods in an attempt to generate a profin on stock market historical data.

1.2. Rationale

For our approach, we started by choosing to focus on a small collection of 5 stocks from publicly traded healthcare companies. We chose to focus on smaller companies, conjecturing that these are less influenced by algorithmic trading than the larger companies that have more visibility and are traded with higher volumes of shares exchanging hands. Due to the aforementioned reluctance of large trading companies to spill their algorithmic secrets, and the limited number of academic research in this space, we sourced ideas from other industries and reapplied the tools to a new context while considering what might work well for time-series datasets. Modern approaches generally have limitations across several dimensions. 1) They are better at classification than they are at regression. 2) They can predict near-term movements better than longer time horizons. 3) A myriad of data sources is required to accurately model stock movements - everything from political events and weather anomalies to X (formerly Twitter) posts and key individuals' thoughts can provide strong signal for stock movements. 4) They need significant amounts of data to accurately distill the patterns into weights. If successful, beyond the immediate profits generated on the public trading markets, this exploration would inform time-series deep learning approaches more broadly across industries. Real-time human vitals could be analyzed for anomalous adverse health events; automobile telemetry could be analyzed for part failure; and factory workers could be kept safe from industrial machinery malfunctions.

1.3. Data

We gathered data from Yahoo! Finance's daily stock data APIs to obtain historical pricing data for 5 different symbols[3, 4]. A simple ETL pipeline was used to transform the data into usable input features. Four of the stock tickers were selected based on high volatility (more opportunity to outsmart the market if the stock goes further up and down), participation in the Healthcare space (interesting area that could be the next frontier for innovation), and small market cap (hopefully not as many competing algorithmic market participants). The final symbol SPX was se-

	Train	Valid	Test
Start Date	2020-10-18	2023-10-18	2024-04-18
End Date	2023-10-17	2024-04-17	2024-10-18
Time Span	3 years	6 mo	6 mo

Table 1. Data splits and time frames.

Model	Parameter Count
RNN	4,003
Transformer	3,275,971
LTCN	7,107

Table 2. Parameter counts for each of the models used

lected to capture a broader signal to promote learning. Raw and processed data are stored in the GitHub repository.

1.4. Dataset Properties

The ticker data consists of columns date, open, high, low, close, adjusted close, and volume over the range 2020-10-18 to 2024-10-18. This daily ticker data can be used to train medium-term models on the order of days; intraday predictions would require data at 1- or 5-minute intervals.

1.5. Training Splits

The data was partitioned into training, validation, and testing sets as shown in Table 1.

We labeled each row with one of three labels: buy, sell, or hold. The labels were a simple calculation based on the difference with the previous day: buy if it went up, sell if it went down, and hold if the delta is under a threshold.

1.6. Target Examples

Training examples were created by partitioning observations into windows of fixed-length τ, where each window contained features $X_t = x_{t-\tau}, ..., x_t$. The target label for the window was computed based on next day returns.

1.7. Features

A variety of technical indicators are commonly used in the enrichment of financial time-series data[5, 6]. The following features were added to the training data:

- Simple Moving Average (SMA): 10, 20, and 30 day
- Exponential Moving Average (EMA): 10, 20, and 30 day
- Relative Volume: 10, 20, and 30 day
- %B: Computes the relative position of the current price within the 95% confidence interval bands of SMA20
- Moving Average Convergence Divergence (MACD): An oscillation indicator using the 9, 12, and 26 day EMAs
- Relative Strength Index (RSI): An oscillation indicator grounded in the ratio of average gains to losses over a 14 day period. $RSI = 100 - \frac{100}{1+RS}$, where $RS = \frac{AvgGain}{AvgLoss}$
- Momentum / Rate of Change: 10 day momentum. $Moment_t = price_t / price_{t-10} - 1$

Human traders that engage with markets on this timescale often use technical indicators such as these to make buy/sell decisions on their portfolio[6]. Future areas of exploration would include combining these short-term signals with longer-term signals like metrics from the company's balance sheet, sentiment analysis from earnings press releases, and social signals from the broader market participant pool.

2. Approach

We took existing approaches in other industries and reapplied them to this new-to-us domain, taking a no-frills approach to data engineering, model training, and trading simulation in order to gain a deep (no pun intended) understanding and first-hand experience of the challenges in the space. We decided to structure the task as classification of next day movement - up, down or flat - rather than regression. Simply - we do not care by how much a stock will go up, we only care that it goes up so that we can buy it and subsequently sell when we predict it will reverse direction.

We applied the Liquid Time-constant Network (LTCN), Spatio-Temporal Transformer (STT), and Identity RNN (IRNN) to the 5-stock dataset; we go into the model descriptions below.

2.1. Model Details

2.1.1 Identity RNN

The **Identity RNN (IRNN) [7]** is the simplest of our three models, leveraging the basic Recurrent Neural Network architecture. The basic Recurrent Neural Network has three weight matrices, U, W, and V, and two inputs, H and X. Outputs are computed by applying the following equations [8]:

$$a^{(t)} = b + Wh^{(t-1)} + Ux^{(t)} \quad (1)$$

$$h^{(t)} = tanh(a^{(t)}) \quad (2)$$

$$o^{(t)} = c + Vh^{(t)} \quad (3)$$

$$\hat{y}^{(t)} = softmax(o^{(t)}) \quad (4)$$

The name IRNN is such because this variant simplifies the initialization to an identity matrix for the weights:

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \tag{5}$$

It additionally assigns 0's for the biases, b and c in the above equations. These initializations encourage the RNN to behave more like an LSTM. This works because when initializing to the identity matrix, the architecture tends toward an LSTM with the forget gate set to not decay the cell state at all. This is not equivalent to an LSTM but can stand in for one with comparative performance and a lot less parameters, making for faster training cycles and lower inference latency. The identity matrix can be scaled by a constant before assignment to both the input weight matrix, $U = \beta I$, and the hidden weight matrix, $W = \beta I$, to interpolate between no memory and full memory.

2.1.2 Spatiotemporal Transformer

The **Spatiotemporal Transformer (STT)** leverages time2vec temporal encodings, a transformer-encoder stack, an LSTM decoder, and a fully connected classification module. The positional encoding utilized in the original transformer architecture only considers relative position within a context window. Temporal encodings extend upon this concept, allowing encodings to have relative importance across different context windows[9, 10, 2, 11, 12].

STT combines temporal encoding with expanded multivariate time series data within the sequence. Financial time-series data consist of observations $x_t \in \mathbb{R}^d$, where x_t represents the feature vector at time t. Rather than project x_t to an arbitrary dimension, STT concatenates the temporal encoding $t2v \in \mathbb{R}^k$ with x_t^i, the i-th feature at time t to produce intermediate embeddings. The expansion of the features combined with the concatenation of $t2v$ allows for more complex interactions across sequences and variables.

The intermediate embedding $E_t \in \mathbb{R}^{d \times (k+1)}$ is then passed through a dense embedding layer that projects to $\mathbb{R}^{d_{model}}$ that the transformer-encoder accepts.

Pretrained t2v Embedding The t2v module of STT was pre-trained using a simple task of having a date at time t predict the date at $t + 1$ for the date range [1900-01-01, 2100-01-01]. This pre-training step provided additional stability during the primary model training and evaluation process. A more complex pre-training task may provide increased benefit, but was outside the scope of this experiment.

2.1.3 Liquid Time-Constant Network

Liquid Time-Constant Network (LTCN) [13] is a take on the Recurrent Neural Networks that is based on and inspired by biological neurons, specifically C. Elegens. It uses a process to solve a series of Ordinary Differential Equations (ODEs) to solve problems in a way that allows the neural network to act in a spiking fashion, as well as being transparent in how it is performing its actions. LTCNs tend to require fewer resources than their deeper cousins and work well with time series data, though they suffer from some of the same issues other RNNs suffer from, specifically vanishing gradients, especially with longer sequence data. Some solutions to these issues have been made, specifically Closed-form Continuous-time Neural nets [14], which seeks to reduce the number of Differential Equations down to 1 in an effort to better learn spatiotemporal tasks. Additionally, Liquid Structural State-Space Models (Liquid S4) were developed as an extension of the SSM in an effort to build a better performing model with spatiotemporal tasks [15].

The main piece of the LTCN is the use of ODE's, a system of equations, which depending on the implementation, could require an exponential number of discretized steps when simulated [13]. Instead, the paper that describes this model develops an ODE solver called a Fused Solver that fuses the explicit and implicit Euler methods. The Fused solver is defined by the following equation:

$$x(t + \Delta t) = \frac{x(t) + \Delta t f(x(t), \mathbf{I}(t), t, \theta) A}{1 + \Delta t (1/\tau_f(x(t), \mathbf{I}(t), t, \theta)} \tag{6}$$

Where $\mathbf{I}(t)$ is the M-dimensional input, θ is the set of parameters, τ is the time constant (this is "the parameter characteriziong the speed and the coupling sensitivity of an ODE" [13] the inverse of which is the membrane capacitance of our neuron [16]), A is a bias vector, $x(t)$ is our hidden value (which could be cellularized for a more LSTM experience, or have an LSTM tacked on as was present in the official library implementation [17]), Δt is the step size, generally defined as $1/L$ where L = the number of layers or iterations we will run for our ODE solver, and finally $f(.)$ is a non-linear function. In our implementation, which we derived in part from the official library [17] and the following step by step tutorial [16], we added a way to switch up the nonlinearity beyond just a sigmoid or tanh, although most other nonlinearities did not train as well as the sigmoid or tanh.

2.2. Evaluation Metrics

Evaluation of classification performance focuses on True Positive (TP), True Negative (TN), and their false counterparts False Positive (FP) and False Negative (FN). The following classification based metrics were used to evaluate

Accuracy, MCC				
	ATNF	BIVI	CYCC	VTAK
IRNN	0.465, .0405	.4879, .0411	.4718, -.001	.2916, .0333
STT	.3782, .026	.354, -.0061	.4091, -.0067	.2751, -.0816
LTCN	.5197, 0	.5039, 0	.5354, 0	.4961, 0

Table 3. Data splits and time frames.

model performance: Accuracy: The percentage of correctly predicted labels

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

Matthews Correlation Coefficient (MCC): The MCC is commonly used when evaluating trade classifications for financial time-series data [9, 2, 5]. MCC is bound to the range [-1, 1], with +/- 1 indicating positivelynegatively correlated predictions respectively. MCC provides more robustness to class imbalance when compared to ACC.

$$MCC = \frac{TP \times TN - FP \times FN}{TP + TN + FP + FN} \tag{8}$$

2.3. Implementation Details

All models were implemented using the pytorch library in python[18]. All code, data, and figures are available in the GitHub repository at https://github.com/RLStudentsOfDLFall24/i-like-the-stock.

3. Experiments and Results

Table 3 contains the results we were able to achieve with best hyperparameter configuration per model class, with the additional constraint that we are using the same set of hyperparameters for all ticker symbols to promote generalization across market data. This was set up as a 3-class classification task - sell, hold, or buy.

Although we were initially disappointed by the 50% test accuracy across all three models, they generally ended with gains across the 6-month test window when put into simulation. This indicates that although they were noisy in their inferences, the training data enabled them to identify the key inflection points of the chart as they were happening. Some models learned to be very selective when buying and liberal when selling, in effect identifying entry points with very high confidence of a positive movement and then selling after losing some of that confidence. You can see in Table 4 that classes aren't evenly distributed in the test set, requiring the model to accurately identify a smaller number of entry (buy) points in many of these symbols' cases.

3.1. Shared Hyperparameters

We found that each stock will be predicted optimally by a specific set of hyperparameters; it may not be the case

Symbol	Test Sell Pct	Test Hold Pct	Test Buy Pct
ATNF	0.526	0.073	0.401
BIVI	0.504	0.095	0.401
CYCC	0.54	0.102	0.358
VTAK	0.496	0.124	0.379

Table 4. Test Set Class Distribution by Symbol

that a single set of hyperparameters should work well across the board, given the nuances of each symbol's movements. However that single configuration can give a great place to start from when incorporating a new symbol into one's trading portfolio. One other note - the transformer leverages about 1000x the parameter count (see Table 2) as the other two models do. This begs the question, is it worth it? The results paint a mixed picture, which is not a clear signal towards the Transformer-based model. Under the construct of using a single hyperparameter set, we don't have an answer. However future research could investigate individually tuning per symbol, perhaps allowing a best algorithm to emerge there.

The experiments below are ablation studies testing two hypotheses we had; we focused on the $BIVI symbol and stock ticker for these experiments.

3.2. Optimizers and Loss Functions

The first ablation study we were interested in was testing different loss functions due to the distribution of the source data. Since the label distribution is imbalanced (24%, 37%, 35%, and 42% buy labels), class-balanced focal loss could help to appropriately weight the dataset[19]. We contrasted this with vanilla Cross Entropy Loss (CE). Results are presented in Figure 1.

Both the RNN and Transformer perform better with the standard cross-entropy loss, but the LTCN performs best with class-balanced focal loss. For the tested symbol, the Transformer had the best overall strategy test outcome. Both the RNN and the STT (containing an LSTM) do better with CE loss; an explanation for this could be that the models should not be weighting the imbalanced classes higher, as they might learn e.g. a buy probability that is too high for that particular situation. It is interesting that the LTCN has almost the same strategy across the two losses, it is just shifted up a little bit. This indicates that the class balancing that the loss function applies is scaling the strategy's profit in a beneficial way.

3.3. Adding Additional Data with Pretraining

Another idea we validated was that signal from a broader selection of market data will benefit learning for the target dataset and ticker. We tested including other tickers from our dataset in a pre-training phase. We included all tickers

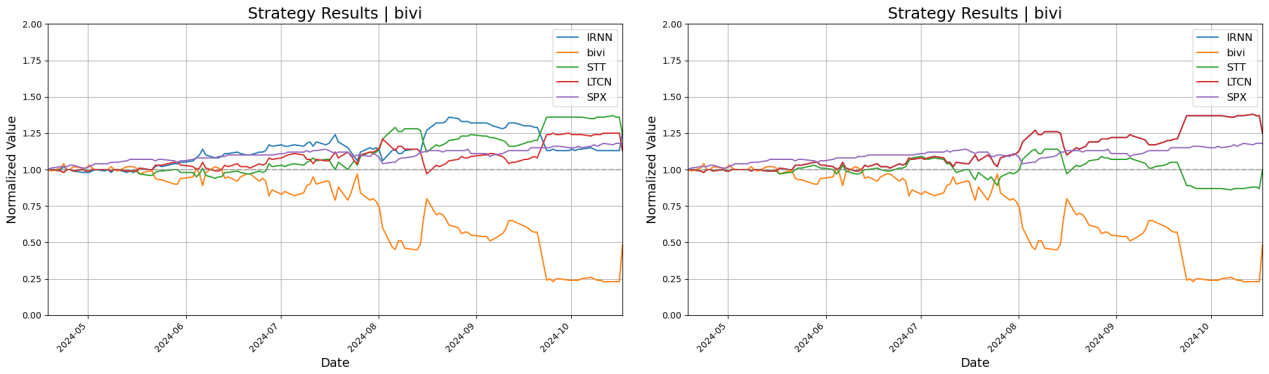

Figure 1. Comparison of results using both CE and CB loss functions for the symbol $BIVI. *Left*: Simulation results comparison for CE. *Right*: Simulation results using class balanced focal loss (CB).

(including the target symbol) in the pre-training phase, running the ablation study by varying the percentage of epochs spent training on the full dataset versus fine-tuning on the target symbol dataset only. For fine-tuning, we introduced another parameter to step down the learning rate and held that constant per model across this study, using the best value found per model. Results are presented in Figure 2.

The RNN did not perform well across both the 75% and 99% splits. Interestingly however and to investigate why this might be, we ran an experiment excluding the target symbol in pretraining data (not shown in visualizations here), and that significantly and positively affected the RNN output; the RNN in these trials moved to the highest-performing spot out of the three models. In those cases, this may be due to the RNN not overfitting the target symbol training data and therefore gaining accuracy on the test set through intelligent generalization. The test chart for the RNN in that not-shown experiment was almost a mirror image of the downward-direction symbol chart, indicating that even a few presentations (epochs) of the fine-tune data is enough to update the (relatively few) parameters and generalize well.

The Transformer interestingly is sensitive to this change in the opposite way; it prefers to see the target symbol in the pretraining phase for best results. This could indicate that it is not generalizing well and is overfitting the target symbol's training data. The LTCN does not seem to be as sensitive to this type of data, proving consistent across the ablation range. These results indicate that future research through gathering a larger pretraining dataset would help to distinguish why these models may be behaving this way - pretraining may have more of an effect on both of these models with a larger dataset. This would be true Transfer Learning.

4. Conclusion

We conjectured that the Spatio-Temporal Transformer with its many parameters would have higher representational power and therefore be able to extract much higher returns from the market for the target symbols; we were proven wrong there. The model that came back with the most positive surprise was also the simplest - the RNN. The RNN, with the fewest number of parameters out of the three, excelled at predicting a subset of the symbols that we tested. It did not perform well on all of them, but given the single hyperparameter constraint this was not too much of a surprise. Out of the box and given a few interesting technical indicators extracted from the world of trading, these models are able to identify strong signal from the data and use that to become profitable, at least for the test sets defined in this set of experiments. The LTCN showed a lot of promise in concept, but the theoretical strength did not translate to profitability results; the LTCN performed slightly better than our baseline of buy-and-hold SPX in most cases but did not achieve outsized returns relative to the other two models.

Looking forward, we'd look at testing these models further to validate their generalizability across industries, instruments, etc. - commodities, futures, and cryptocurrency are all interesting areas with their own opportunities and challenges. In terms of engineering work, additional intelligence can be added both when creating the labels for the dataset and when running the simulation to improve profitability of the strategy. Execution control, e.g. selling when at a 10% loss, could probably also improve profitability but outside of the Deep Learning walls of this experiment.

References

[1] Yongfeng Wang and Guofeng Yan. Survey on the application of deep learning in algorithmic trading. *Data Science in Finance and Economics*, 1(4):345–361, 2021.

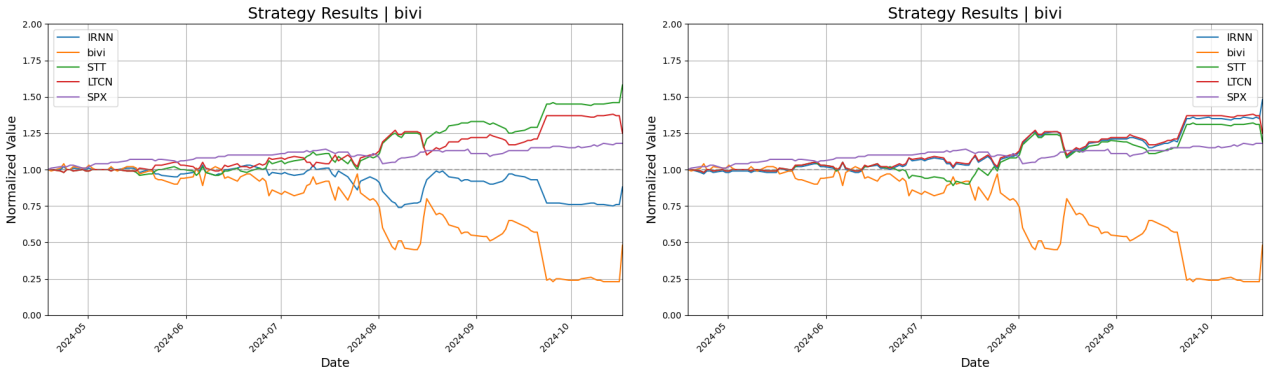

Figure 2. Comparison of pretraining/fine-tuning ablation; 75% (left) and 99% (right) pretraining selected for further visualization here.

[2] Yang Li, Hong-Ning Dai, and Zibin Zheng. Selective transfer learning with adversarial training for stock movement prediction. *Connection Science*, 34(1):492–510, December 2022. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/09540091.2021.2021143.

[3] Yahoo! Inc. Yahoo finance. https://finance.yahoo.com.

[4] Ran Aroussi. yfinance: Yahoo! finance market data downloader. https://github.com/ranaroussi/yfinance, 2019.

[5] Jinan Zou, Qingying Zhao, Yang Jiao, Haiyao Cao, Yanxi Liu, Qingsen Yan, Ehsan Abbasnejad, Lingqiao Liu, and Javen Qinfeng Shi. Stock Market Prediction via Deep Learning Techniques: A Survey, 2023. _eprint: 2212.12717.

[6] Technical Indicators | ChartSchool | StockCharts.com.

[7] Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015.

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

[9] Daniel Boyle and Jugal Kalita. Spatiotemporal Transformer for Stock Movement Prediction, May 2023. arXiv:2305.03835 [cs].

[10] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2Vec: Learning a Vector Representation of Time, July 2019. arXiv:1907.05321 [cs].

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. arXiv:1810.04805.

[13] Ramin Hasani, Mathias Lechner, Alexander Amini, Daniela Rus, and Radu Grosu. Liquid Time-constant Networks, December 2020. arXiv:2006.04439 [cs, stat].

[14] Lechner Hasani, R., M., and A. et al. Amini. Closed-form continuous-time neural networks. *Nature Machine Intelligence*, 4:992—-1003, 2022. https://doi.org/10.1038/s42256-022-00556-7.

[15] Ramin Hasani, Mathias Lechner, Tsun-Huang Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.

[16] Pavel Nakaznenko. Step-by-step guide to building an ltc liquid neural network from scratch, 2024. https://github.com/KPEKEP/LTCtutorial/blob/main/LNN_LTC_Tutorial_Eng.ipynb.

[17] Mathias Lechner et al. ncps, 2024. https://github.com/mlech26l/ncps/.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[19] Yin Cui, Menglin Jia, Tsung-Yi Lin, Yang Song, and Serge J. Belongie. Class-balanced loss based on effective number of samples. *CoRR*, abs/1901.05555, 2019.

Student Name	Contributed Aspects	Details
Sakae Watanabe	Implementation, Experimentation	Data & feature engineering; training base code; STT model build/tune; trading simulation
Ashish Narasimham	Implementation, Paper Writing	Dataset sourcing; training code generalization; pre-training feature; RNN model build/tune; paper writing
Michael Walker	Implementation, Experimentation	Refactor code and configuration for reusability; experiment setup; LTCN model build/tune; grammar fixes and minor text updates

Table 5. Contributions of team members.

5. Work Division

The team felt that team members' contributions met a minimum bar, and no group member was absent from the project work. Sakae wrote the majority of the framework code around the models - data engineering, feature engineering, experimentation, and trading simulation; Ashish and Michael contributed to refactors of that code for generalization and reusability. We each contributed our own model development and tuning, and all had ideas for experimentation/analysis.