Predicting Stock Movements Using Deep Neural Networks

Tristan Lee
tlee10@bu.edu

Abstract

We detail the use of a deep neural network (DNN) architecture to predict future stock movements given previous stock data. The DNN is mainly composed of transformer blocks that each use an LSTM to learn a positional encoding, multi-head self attention, and mixture of experts. We find that in the short to medium term, our models to not outperform simpler strategies of buying at regular intervals, and instead find evidence for long term investing being the more effective strategy.

1 Introduction

Profiting off predicting the stock market is an age-old problem. Much of the investing space suggests to simply "buy and hold" or "continuously buy" with only long-term returns in mind, essentially just following the market. Indeed, most strategies, except for those used in the world's largest and most sophisticated hedge funds, fail to beat the market in the long term. However, this leaves a possibility for strategies to beat the market in the short to medium term. In this paper, we describe a deep neural network (DNN) architecture that attempts to learn such a strategy. We model this as a supervised classification problem: given input instance x containing certain stock data over previous days, produce output instance y corresponding to a predicted stock movement. We directly use the predicted movement to take an action with the stock: sell, do nothing, or buy. The goal is for the model to predict actions in such a way that can perform as well or even beat the market, as measured by the gain/loss as both the raw value and a percentage.

2 Data

The data is an aggregation of data from different APIs. These APIs and their corresponding data are as follows:

- Yahoo Finance, accessed through the Python PyPI package `yfinance` (Aroussi, 2025). This gives historical stock price data for every day of the ticker's existence. We use data from years 2000 to 2023 for training, and 2024 onward for testing and evaluation (the last date for testing is April 25th, 2025). From this data we calculate various technical indicators: simple and exponential moving averages, crosses, MACD, stochastic oscillator, and RSI (see section 4.1.1 for a detailed description of each indicator and their calculation). `yfinance` also provides the sector of each ticker (technology, energy, etc.).

- Alpha Vantage, accessed through REST API (AlphaVantage, 2025). This provides historical fundamental data (reported earnings, analysts' expected earnings, etc.) for all tickers over the whole time range given above. We use quarterly earnings, specifically reported and expected earnings-per-share, and P/E ratio (see section 4.1.1 for a detailed description of each indicator and their calculation). We also use Yahoo Finance to access the most recent earnings data of the same type as just described.

This data was collected for the S&P 100 companies. See Figure 1 for a sample of this aggregated data.

3 Baseline Measurements

Let a *strategy* take as input a ticker's data over a time period as described previously, and produce as output a list of predicted actions, one per day of data: sell, do nothing, or buy, with the restriction that on any day i, the strategy makes use of no information from days j for $j > i$. Because we use close prices, this corresponds to making a decision after the closing bell, or alternatively, at a time t right before the closing bell, using the price at time

	Close	5_SMA	20_SMA	MACD	estimate_EPS
0	185.64	191.59	194.03	0.78	1.39
1	184.25	189.83	193.77	0.05	1.39
2	181.91	187.58	193.19	-0.70	1.39
3	181.18	185.10	192.64	-1.35	1.39
..	...	...	...	...	...
247	258.20	253.16	246.29	6.05	1.60
248	259.02	255.35	247.49	6.29	1.60
249	255.59	256.51	248.52	6.13	1.60
250	252.20	256.06	249.27	5.66	1.60

Figure 1: Select columns of aggregated data for ticker AAPL (Apple).

t as an estimate for the day's closing price. We evaluate the performance of a strategy by executing actions as follows:

- If "buy," buy 1 more of the stock at the current price.
- If "do nothing," do nothing.
- If "sell," sell all owned stock at the current price.

Using this method, we can record the profit/loss and cost for each ticker, and aggregate across all tickers. For example, this could look like "net profit of \$100 using \$1000," for a 10% gain. Note that we scale every ticker's prices such that their first day's price is \$1. This gives all tickers equal importance to the evaluation. Absent is any sort of options and derivatives, the only profit to be made here is by selling a stock for more than the buy price.

With this, we first evaluate several simple strategies on all 2024 data (we expand this time period to include the available data for 2025 after the candidate DNNs have finished trained, and re-evaluate the baselines). Note that we force a sell on the last day of the time period to close out any positions. We detail each strategy below, and give their performance in Table 1 (note that the performance of these strategies on each individual stock is viewable at this project's GitHub repository [1]):

1. Buy and hold: buy once at the beginning of the time period, and hold until the last day of the time period.
2. Buy: buy on every day of the time period.

[1] https://github.com/TristanLee187/Trading_Model/tree/main/plots/preliminary/baselines

Strategy	Gain/Loss(%)	Gain/Loss($)	Cost($)
Buy and hold	16.95	16.95	100.00
Buy	6.95	1792.08	27563.28
Random	0.24	21.71	9083.74
Momentum	1.20	5.04	419.78
Swing	3.86	44.33	1148.91

Table 1: Performance for each baseline strategy on the 2024 set.

3. Random: randomly execute buy, sell, and do-nothing actions. Multiple seeds were used and gave very similar results.
4. Momentum: buy when the 5 SMA crosses above the 20 SMA (indicating an upward shift in momentum), and sell when it crosses below.
5. Swing: buy when the RSI goes below 0.3 (indicating the stock is oversold), and sell when it goes above 0.7 (indicating the stock is overbought).

Note that cost is directly related to the number of buy actions taken, so in general higher costs mean more actions, which correlates to a more robust, tested strategy if the percentage gains are about equal. 2024 was an exceptionally good year for stocks, so the buy-and-hold strategy performed significantly better than any other in terms of percentage gain. This also made the strategy of just buying effective as well, notably making the most absolute profit. These strategies match advice that is widely given in the investment space, so a model that can match or even surpass these strategies' performance will be somewhat impressive.

4 Methods

With the baselines established, we now detail the exact structure of the data we use, the architecture of the deep neural network used to output predict actions, training settings, and performance on the 2024 data set. We model the problem as a supervised classification problem: given some input instance x of previous days' stock data for a particular ticker, predict an output instance y corresponding to a buy, do nothing, or sell action.

4.1 Data Structure

We first detail the exact structure of the input and output instances, as well as how they were computed.

4.1.1 Input Instances

Each input instance x consists of stock data for a particular stock ticker (as well some small amount of information about other tickers, see details below). Then we can split x into 2 distinct types of data: x_{seq}, sequential data where each token of the sequence corresponds with a particular day's data, and x_{meta}, "metadata" associated with the ticker as a whole rather than individual tokens of the sequence. x_{meta} consists of just the ticker's sector, given as a one-hot vector. x_{seq} is a sequence of 30 tokens, where each token consists contiguous days' worth of the various technical and fundamental indicators alluded to previously. The exact technical indicators used, many of which are taken from Investopedia's site of technical indicators (Chen et al., 2021), are detailed as follows:

- Normalized close price. Let m be the maximum close price in the sequence of 30 days in question. Then we divide all the close prices c_i by m, normalizing such that the new maximum value becomes 1.

- Proportional change. This is the difference from the previous day's close price to the current day's, divided by the previous day's.

- Normalized volume. Like the normalized close price, we simply divide the volume at each day by the maximum volume seen in the 30 day sequence.

- Normalized simple moving averages (SMAs) and crosses. First, we define the SMA on a given day of a given window size n as follows: the mean of the n consecutive close prices starting from $n-1$ days ago, and ending at the current day. So the 5_SMA on a given day is the mean of the 5 close prices starting from 4 days in the past and ending at the current day. We collect the SMA for window sizes 5, 20, 50, and 200, and normalize them by dividing by the same maximum close price m as defined previously. Finally, we compute crosses by subtracting one SMA from another. We collect the crosses between the 5_SMA and 20_SMA, 20_SMA and 50_SMA, and 50_ and 200_SMA, which are all normalized by m. SMAs and crosses are roughly used to identify uptrends and downtrends in the price. Note that we use SMA as a function on other values besides the close price (see RSI below); the SMA of a sequence in general should be thought of as a smoothing of the sequence using arithmetic mean.

- Normalized exponential moving averages (EMAs) and moving average convergence/divergence (MACD). We define the EMA on a given day of window size n using close price similarly as the SMA, but instead of the simple mean, we use a weighted mean where the weights exponentially decrease with distance from the present day (this emphasizes values closer to the present day). Specifically, the sequence is defined recursively as

$$\begin{aligned}\mathrm{EMA}_0 &= x_0,\\ \mathrm{EMA}_i &= (1-\alpha)\,\mathrm{EMA}_{i-1} + \alpha x_i,\end{aligned}$$

  where we choose $\alpha = \frac{2}{n+1}$. We collect the EMA for window sizes 5, 20, 50, and 200, and normalize by dividing by m like the SMA. Finally, we compute the MACD by subtracting 2 specific EMAs from one another, in this case the 12_EMA and 26_EMA, which is also normalized by m. EMAs and the MACD are another flavor of uptrend and downtrend identification, like SMAs and crosses. Like the SMA, we can use the EMA on other values as a way to smooth a sequence (see EPS in the fundamental indicators).

- Stochastic oscillator. Let h be the sequence of rolling maximum close prices for a given window size n (that is, h_i is the maximum close price of the n days from day $i-n+1$ to day i), and let l be the same but for the minimum close prices. Then the stochastic oscillator s is calculated as

$$s_i = \frac{c_i - l_i}{h_i - l_i},$$

  where c_i is the *non-normalized* close price of day i. Note that none of l and h are normalized, and neither is the stochastic oscillator, since this value is already between 0 and 1 as is. We use a window size of 14 here. The stochastic oscillator is roughly a measure of volatility in the price.

- Relative strength index (RSI). First we define a raw RSI over a window size n as follows: let g be the sequence of proportional

changes where the losses are zero-ed out, and let l be the sequence of proportional changes where the gains are zero-ed (g corresponds to gains, and l to losses). Let $\hat{g}$ and $\hat{l}$ be the SMA of g and l, respectively, with window size n. Then let rs be the element-wise quotient $\frac{\hat{g}}{\max(\hat{h}, 10^{-7})}$, where we use $\max(\cdot, 10^{-7})$ to prevent division by 0. The raw RSI is defined as the element-wise sequence $1 - \frac{1}{1+rs}$. Finally, the final RSI is the SMA of the raw RSI using window size n. This gives another "already normalized" value between 0 and 1. The RSI is roughly used to indicate momentum by identifying oversold and overbought conditions based on thresholding the RSI.

- Sector average proportional change. This is the one indicator that depends on other tickers. This is simply the mean of the proportional changes for each day of all tickers of the given ticker's sector. Performance of similar tickers may influence the future performance of a particular ticker.

Now we detail the fundamental indicators:

- Smoothed estimated earnings per share (EPS) and reported EPS. First, the raw EPS is simply the earnings per share in dollars. Expected EPS is given by an aggregate of certain big analyst firms, and the reported EPS is simply the ground truth EPS. These are updated whenever a company posts earnings, so we forward fill the sequence, such that at every day i, the EPS data for day i contains the most recent EPS data. Finally, we smooth each of these EPS using an EMA of window size 4. These values are not normalized.

- Smoothed surprise percent. First, the raw surprise percent is simply the proportional difference between the reported and expected EPS (we keep this as a raw value rather than a percentage value, despite the name). Then we smooth by once again using an EMA of window size 4.

- Normalized P/E ratio. First, we compute the raw P/E ratio by simply dividing the close price by the reported EPS. Then, we normalize by dividing this raw value by 100 and taking the sigmoid of the result. This gives a value between 0 and 1.

So, each token in x_{seq} has dimensionality 22: 18 from the technical indicators and 4 from the fundamental indicators. Thus x_{seq} has dimension 30-by-22. Also, the S&P 100 tickers used here span 11 sectors, so x_{meta} has dimension 11. These define a single input sequence x.

4.1.2 Output Instances

Each output instance y is simply a one-hot vector for each action: sell, do nothing, or buy, corresponding to $[1, 0, 0]^T$, $[0, 1, 0]^T$, and $[0, 0, 1]^T$, respectively. We determine these actions using constrained linear regression: suppose x is the input instance associated with y, where x spans 30 consecutive days. Let p_x be the last close price in x, and let $\mathbf{p}$ be the sequence of close prices for the following 15 consecutive days after the last day of x. Then let l be the regression line through values p_x and $\mathbf{p}$ (using least squares), but subject to the constraint that l must pass through the value p_x (this is a quadratic programming problem solved using the `scipy.optimize.minimize` function) (see Figure 2 for an illustrative example of this). We take the slope of this resulting line, multiply by 15, and divide by the p_x to get a proportional change δ over the 15 days based on the linear regression. We threshold δ to get our action label y:

$$y = \begin{cases} \text{sell} & \text{if } \delta \leq -0.05 \\ \text{do nothing} & \text{if } -0.05 < \delta < 0.05 \\ \text{buy} & \text{if } 0.05 \leq \delta. \end{cases}$$

We use these regression based labels in order to minimize noise: for instance, say we instead used the raw proportional change from the close price on last day of x to the immediate next day. Since day-to-day changes in market prices are noisy, such labels would also be extremely noisy, and a model would likely overfit. Using this regression approach hopefully reduces the amount of label noise by looking over a multi-day trend.

4.1.3 Generating the Training Data

Our training corpus consists of data, indicators, and features detailed above from the years 2000 to 2023 (note that for indicators like SMAs and EMAs that require large windows into the past, this indeed necessitates using data from before 2000. Also, for tickers whose data only begins after 2000, we use the oldest available data). With the definitions of x and y, generating a single (x, y) pair from the raw training data corpus requires iterating over

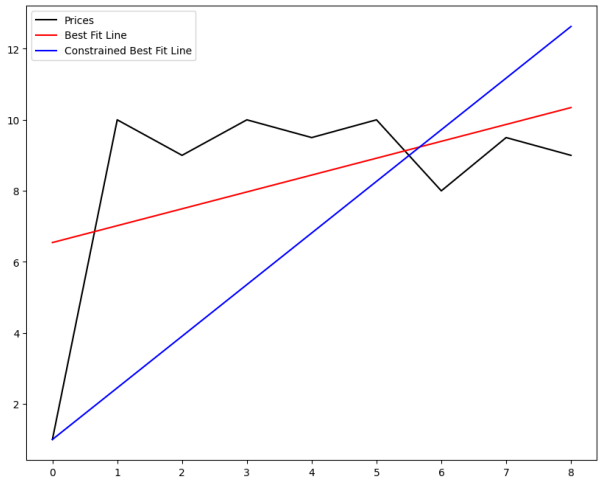

Figure 2: Regular best fit line vs. constrained best fit line. Using regression aggregates the price action over multiple days in the future. Constraining to the present day price ties the aggregation to current price action.

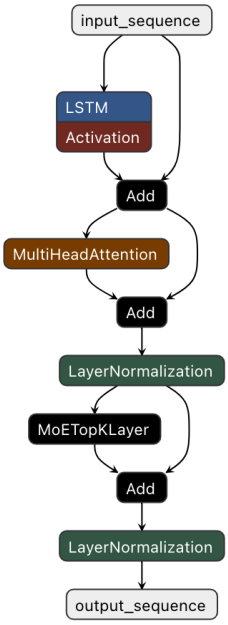

Figure 3: Architecture of a single transformer block. `input_sequence` and `output_sequence` have the same shape.

windows of 45 consecutive days' data: the first 30 define x, and the last 15 define y (along with the 30th day's close price). We also use a stride of 15 when iterating of the corpus, so that none of output labels for 2 different instances use overlapping data. The result is a transformed training set (X, Y) with 37215 examples, where X is composed of (X_{seq}, X_{meta}). Each $x_{seq} \in X_{seq}$ is a sequence with dimension 30-by-22, each $x_{meta} \in X_{meta}$ is a one-hot vector of dimension 11 for the ticker's sector, and each $y \in Y$ is a one-hot vector of dimension 3 for the predicted action. Y consists of roughly 17% sell actions, 60% do nothing actions, and 23% buy actions. Expectedly, do nothing actions compose the majority.

4.2 Model Architecture

With the structure of the data defined, we now detail the DNN architecture. See Figures 3 and 4 for diagrams, courtesy of Netron.app (Roeder, 2021).

4.2.1 Transformer Block

The main unit of the DNN architecture is a transformer block that processes sequence data, using the following components in order:

- LSTM positional encoding. We use a learnable recurrent model, specifically an LSTM, to learn a positional encoding, rather than using a fixed encoding. The LSTM outputs sequences of the same shape as the input, and we add the output to the original input to get a position encoded sequence. As a recurrent model, an LSTM seems suitable to learn a positional encoding.

- Multi-head self attention. We apply self attention to the new sequence using 2 heads and key dimensionality 8. The `Keras` implementation implements the query, key, and value matrices; the appropriate dot products; and the softmax operation. We use self attention to consider the entire context of the sequence to inform the predicted action, as opposed to recurrent models which may struggle with longer sequence lengths.

- Top-k mixture of experts (MoE). Rather than using a single feed forward network (FNN) after the attention output, we learn multiple FNNs with a gating network that routes to the top k most highly weighted of them. The motivation here is to learn "experts" that specialize in different "types" of sequences as determined by the router. Using MoE over a single FNN has seen success in LLMs (Cai et al., 2025), so it may be effective here.

We add a residual connection and apply layer normalization after both the attention and MoE layers to help with gradient flow and stabilize the activations. See Figure 3 for a diagram of a single transformer block we define here.

4.2.2 Full Architecture

Treating the transformer block as a primitive, we now define the full DNN architecture:

- Temporal and feature sequences. We maintain two versions of each input sequence x_{seq}: the original of shape 30-by-22, and the transpose of the sequence with shape 22-by-30. The original sequence is a temporal sequence where each token is associated with a single day, while the transposed sequence is a sequence of vectors of a single feature (technical or fundamental indicator). With this transposed, feature-based sequence, our transformer blocks may be able to learn the relationships between each feature more explicitly, as each token now consists of just a single feature but over multiple days, and can be compared to the other features via attention. We call these 2 versions the *temporal* and *feature* sequences.

- Transformer blocks. For both the temporal and feature sequences, we apply a stack of 4 transformer blocks as defined in the previous section. We apply multiple transformer blocks in the hopes of learning higher level features of the data, but not too many as to discourage overfitting and incur extra computation cost.

- LSTM aggregation. We add the outputs of both transformer stacks (transposing the feature sequence output back to shape 30-by-22) to aggregate their information, and apply an LSTM to reduce the 30-by-22 sequence to a single 22 dimensional vector. This is analogous to using a `<CLS>` token in traditional transformers, but we use a recurrent network instead to hopefully learn a richer aggregation.

- x_{meta} concatenation. Here we concatenate the aggregation with the one-hot sector vector to start using information about the ticker's sector. We reason that certain lower level trends and features can be learned from the sequence data without referring to the sector, and only once the higher level features are learned do we incorporate the sector.

- Fully connected layers and softmax output. With both the sequence data and metadata vectors aggregated, we apply multiple fully connected layers of gradually decreasing dimensionality (64, 32, 32) to apply non-linearities and reduce the dimensionality. The final layer applies softmax and produces a vector of size 3, corresponding to predicting one of the 3 actions.

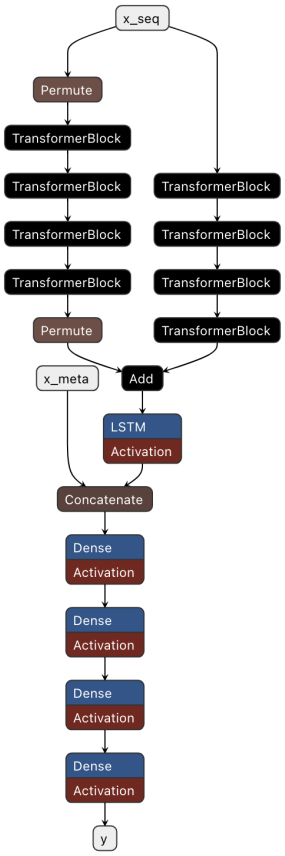

Figure 4: The full DNN architecture, where `TransformerBlock` is diagramed in Figure 3. Each `Permute` block transposes its input.

See Figure 4 for a diagram of the full DNN defined here.

4.3 Training

With the model architecture established, we define 3 different models based on 3 different loss functions. All 3 use a custom categorical cross-entropy loss function, using a custom weight matrix to penalize different types of errors differently. Let W be a 3-by-3 weight matrix for calculating categorical cross-entropy loss for our labels, where $W^{i,j}$ is the penalty for predicting j when the ground truth was i (recall that based on the defined one-hot encodings, 0 corresponds to sell, 1 to do nothing, and 2 to buy). Then a naïve W would be defined as follows:

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix},$$

where there is no penalty for correct predictions and a penalty of 1 for any incorrect prediction. Each loss function L_i will use a weight matrix W_i to penalize different types of errors differently, defining a model M_i.

- We start with the following simple intuition: errors that mistake sell actions for buy actions and vice versa should be penalized more than errors involving do nothing actions, as buy and sell actions are more "different" than buy and do nothing actions, or sell and do nothing actions. To discourage over buying or selling, we also penalize incorrectly predicting buy or sell actions when the ground truth actions was do nothing, more than the other way around. This gives the following weight matrix:

$$W_1 = \begin{bmatrix} 0 & 2 & 10 \\ 3 & 0 & 3 \\ 10 & 2 & 0 \end{bmatrix}.$$

- We edit W_1 using a perhaps more intuitive approach to the handling of do nothing errors: we should penalize the model for missing buy and sell actions more so than missing do nothing actions. Missing a buy or sell action should have a larger impact on the profitability of the strategy than buying or selling when the price action is stable. This gives the following weight matrix:

$$W_2 = \begin{bmatrix} 0 & 3 & 10 \\ 2 & 0 & 2 \\ 10 & 3 & 0 \end{bmatrix}.$$

- We edit W_2 under the following observation: there are fewer sell actions than buy actions in the training data. Additionally, based on my own general observations, when a stock price decreases, the "velocity" of their change is faster than when that stock price increases. That is, price action generally alternate between long periods of steady gains and short periods of sharp losses. Under these assumptions, we should penalize the model more for missing sell actions, to force the model to pay more attention to learning how to predict them. This gives the following weight matrix:

$$W_3 = \begin{bmatrix} 0 & 5 & 15 \\ 2 & 0 & 2 \\ 10 & 3 & 0 \end{bmatrix}.$$

These 3 weight matrices define 3 different loss functions and give 3 different models M_1, M_2, and M_3. All of them were trained using the same settings, given as follows:

- Adam optimizer, with a learning rate of 0.001, and a scheduler that multiplies the learning rate by 0.5 when the loss plateaus for 10 epochs. The gradients are also clipped to have a norm of at most 1.0.

- Validation data, using 20% of the training corpus.

- Noise augmentation. For each training instance $((x_{seq}, x_{meta}), y)$, we produce 15 copies where noise is added to x_{seq}, while x_{meta} and y are kept the same. The noise is drawn from a normal distribution with squared variance equal to 0.5 of the squared variance of each feature (this gives different noise levels for each feature). This gives a total of $15 * 37215 = 558225$ examples at train time. Adding noise to the training data should reduce overfitting by forcing the model to learn patterns that are robust to this noise.

- L2 regularization in nearly all model weights with $\lambda = 0.01$, and dropout regularization in the multi-head attention layers with dropout factor 0.2.

- Batch size of 64 and 50 epochs.

Training took around 6 hours for each model on the BU CS Linux server's CPU hardware.

4.4 Preliminary Results and Discussion

We use the models to predict on the 2024 data set, taking the argmax of the output one-hot vector to determine the actions. We use the same strategy as the baselines: if buy, buy 1 share, and if sell, sell all owned shares. In addition to the trained models, we also record the performance of the our constrained regression based labels. We append these results to Table 1 to get Table 2 (once again, performance on individual stocks can be viewed at this project's GitHub repository [2]).

Firstly, we see that the ground truth labels have less percentage gain than the best strategy of buy

[2] https://github.com/TristanLee187/Trading_Model/tree/main/plots/preliminary/model_performance

Strategy	Gain/Loss(%)	Gain/Loss($)	Cost($)
Buy and hold	16.95	16.95	100.00
Buy	6.95	1792.08	27563.28
Random	0.24	21.71	9083.74
Momentum	1.20	5.04	419.78
Swing	3.86	44.33	1148.91
Ground Truth	16.03	877.06	5471.49
M_1	11.45	413.65	3613.05
M_2	7.91	941.97	11904.03
M_3	2.22	150.10	6756.54

Table 2: Performance of each baseline strategy, the constrained regression based labels, and each trained model on the 2024 set.

and hold. This is a consequence of performing actions throughout the whole time period: if a stock price increases roughly monotonically (which is approximately true for most stocks in 2024), then the optimal strategy to maximize percentage gain would be to buy once at the beginning of the time period (for this reason, we can view the buy hold strategies and the ground truth labels as ideal, upper limits on percentage gain performance). This is made up for by the labels making substantially more absolute gains, perhaps a more meaningful performance metric, though still less than the buy strategy. We also see that every baseline strategy made some positive profit, though the random strategy only marginally so.

For the trained models, we see that all models also make some amount of profit. M_1 and M_2 do better than the buy strategy but not as well as the buy and hold strategy in terms of percentage gain, while the reverse is true in terms of absolute gain. Comparing M_1 to M_2, M_1 takes substantially less actions. This is expected from the design of the loss functions: W_1 was designed to discourage buying or selling too much, while W_2 penalizes buying and selling less. However, quite concerning is the fact that these models never predicted any sell actions (see the Appendix for illustrative examples, which show predictions for the whole 2024-2025 time period. Predictions made for just 2024 in this preliminary stage are the same as shown in those figures, since the same models were used for both time periods). Indeed, this matches the design of W_1 and W_2 penalizing both types of errors between buy and sell actions equally, even though the data has more buy actions than sell actions. This behavior may be favorable during times when stock prices are generally increasing, but in general we should want our model to learn to sell when prices fall. Moreover, the majority of the profit made for M_1 and M_2 came from a very small subset of tickers, chief among them TSLA (Tesla), accounting for 47.7% and 21.7% of the gain for the respective models. Coupled with the inability to predict sell actions, this makes M_1's and M_2's performance more unlikely to generalize for other market conditions.

Unlike M_1 and M_2, M_3 is capable of predicting sell actions. However, from the performance in Table 2, as well as looking by-eye at the executed actions (see the Appendix), this capability isn't used too favorably: M_3 does substantially worse than M_1 and M_2 in terms of both percent and absolute gains, and the sell actions don't correlate too well with the ground truth labeled sell actions. Nevertheless, this capability may be useful when stock prices don't increase as strongly as in 2024.

4.4.1 Discussion of Training Factors

Many combinations of the features and techniques given above were tried before settling on the final result. While we don't detail all of them here, we discuss some general observations of the effect (or lack thereof) of changing certain factors in the pipeline:

- Feature normalization. This had a huge effect on the coherence of the model; without normalizing the price, for instance, the model would predict all sell actions for tickers for simply being in some price range, and all buy actions for a different range. By normalizing these features, all tickers across all price ranges are treated equally, using scaled values and proportional differences.

- Using a stride when generating training instances. This had a large effect on the observed training, validation, and testing losses. With a stride of 1, the training and validation losses decreases very closely together, suggesting that the model was overfitting to points in the raw training data that are shared among many training instances because of the sliding window. The test time loss was often much greater than the other 2 losses with this setup. With a stride of 15, the training and validation losses more expectedly separated after some epochs, with a generally lower test time loss, though this was generally higher than both the training and validation loss.

Strategy	Gain/Loss(%)	Gain/Loss($)	Cost($)
Buy and hold	12.74	12.74	100.00
Buy	1.04	381.92	36822.68
Random	0.13	16.18	12180.65
Momentum	-0.04	-0.22	588.28
Swing	2.29	45.18	1977.10
Ground Truth	15.75	1134.27	7200.15
M_1	0.42	27.90	6708.02
M_2	-0.85	-143.39	16931.25
M_3	1.04	125.36	12006.38

Table 3: Performance of each baseline strategy, the constrained regression based labels, and each trained model on all 2024-2025.

- L2 regularization. Increasing the L2 regularization factor increased the F1 score (tracked alongside the loss) during training, suggesting that this reduced overfitting. This also increased the percentage gains for M_1 and M_2 by roughly 2-3 points.
- Feature sequences. As opposed to using just the original temporal sequences, incorporating the transpose to get the feature sequences improved model performance across the board, though much more so for M_1 and M_2 (sometimes 8 or more percentage gain points) than M_3.
- Adam vs. RMSProp. Both of these optimizers were tried while keeping other training variables constant, and gave models with nearly identical behavior and performance. I keep the Adam optimized models simply for Adam's ubiquity in the space.
- More epochs. Training seemed to stabilize after roughly 30-40 epochs for each model. Any number of epochs past 50 gave nearly identical models and behavior.

5 Results

We now expand the test set to also include 2025 up to April 25th. We use the baseline strategies, unchanged models, and ground truth labels to predict actions and measure performance, given in Table 3.

2025 brought much harsher market conditions than 2024, with several sharp price declines across nearly all tickers in March and April. This hurt the performance of both the buy and hold strategy and the buy strategy, making the ground truth labels better by all metrics. The buy strategy made the second most absolute gain. Curiously, the swing strategy does not deteriorate nearly as much, suggesting that the more volatile market conditions were less of an issue for this strategy.

For the trained models, M_1's and M_2's performance deteriorated drastically, erasing nearly all of their gains, and even turning negative for M_2. Expectedly, their inability to sell was disadvantageous during the harsher conditions of 2025.

M_3's performance also deteriorated, but not nearly as much as M_1 and M_2. It appears its ability to sell mitigated the damage taken in 2025, getting nearly the same percentage gain as the buy strategy, with lower absolute gain, though. Of all the trained models, M_3 performed the best by all metrics on the 2024-2025 time period.

6 Discussion

As noted previously, the buy and hold strategy and the ground truth labels should be viewed as ideal, upper limits on performance in terms of percentage gain: the first because it doesn't accurately reflect the nature of continuously taking actions over a time period, and the latter for obvious reasons. Of the baselines, the buy strategy attained the absolute gain, while the swing strategy attained the highest percentage gain. None of the trained models beat these baselines on both metrics, with M_3 performing the best. M_3's positive gains coupled with large number of actions makes it probably the most robust of the trained models in the short to medium term.

While the behavior of the models relative to each other mostly aligns with our design of the weight matrices W_i, we further speculate about the behavior of M_1 and M_2 just buying or doing nothing. These models failed to distinguish between buy and sell signals without sufficient penalty from the loss function, and likely chose the buy action because it represented a bigger share of the training data as observed in section 4.1.3. This can be interpreted in one of two ways: either there are underlying patterns in the data that truly distinguish between buy and sell that the model failed to learn, or such patterns don't exist and the models simply defaulted to choosing the more represented class. The behavior of M_3, which does output sell actions but not in a very profitable way, is evidence of the latter, at least using the model architecture defined here. However, we can take the lack of such short to medium term patterns to support the widely prevailing investment strategy of just buying at regular intervals

with only long-term performance in mind. Since these models were trained on 20+ years of data, if we take M_1's and M_2's behavior to be indicative of any market understanding, this would be evidence for this advice.

We also observe that M_1 and M_2 both predict more buy actions when the price action is more volatile in both the upward and downward price direction (see the Appendix). This is likely caused by the nature of the training data and our labels: volatile conditions are followed by proportionally more buy or sell actions than calmer conditions. Coupled with these models' inability to predict sell actions, this means they simply predicted more buy actions.

7 Conclusion

We find that the data structures, model architectures, loss functions, and training settings used to define these models do not produce strategies that reliably perform well in the short to medium term, relative to other simpler strategies. While these models showed varying promise in preliminary training and testing, they fail to generalize well to more diverse market conditions. On the contrary, we can interpret the behavior and performance of our models that were trained with short-medium term goals to actually support the effectiveness of long term strategies, such as simply buying frequently at regular intervals, which also performed reasonably in the time period observed.

References

AlphaVantage. 2025. Alpha vantage.

Ran Aroussi. 2025. yfinance. Version 0.2.54.

Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang. 2025. A survey on mixture of experts in large language models. *Preprint*, arXiv:2407.06204.

James Chen, Charles Potters, and Yarilet Perez. 2021. Technical indicator: Definition, analyst uses, types and examples.

Lutz Roeder. 2021. Netron.app.

Appendix

See the following figures for examples of the models' predicted actions on given tickers on the 2024-2025 time period, compared to the constrained regression ground truth labels (more figures can be found at this project's GitHub repository[3]). We see that M_1 and M_2 are incapable of selling, while M_3 is capable of selling. We also see that the models generally predict more buy/sell actions, or just buy actions for M_1 and M_2, in more volatile market conditions like TSLA (Tesla) than more calm conditions like most of GS (Goldman Sachs) (note that the plots in Figures 5 and 6 scale the price axis to the minimum and maximum price of the ticker over the time period. The proportional changes in GS are actually much smaller than those in TSLA).

[3] https://github.com/TristanLee187/Trading_Model/tree/main/plots/final/model_plots

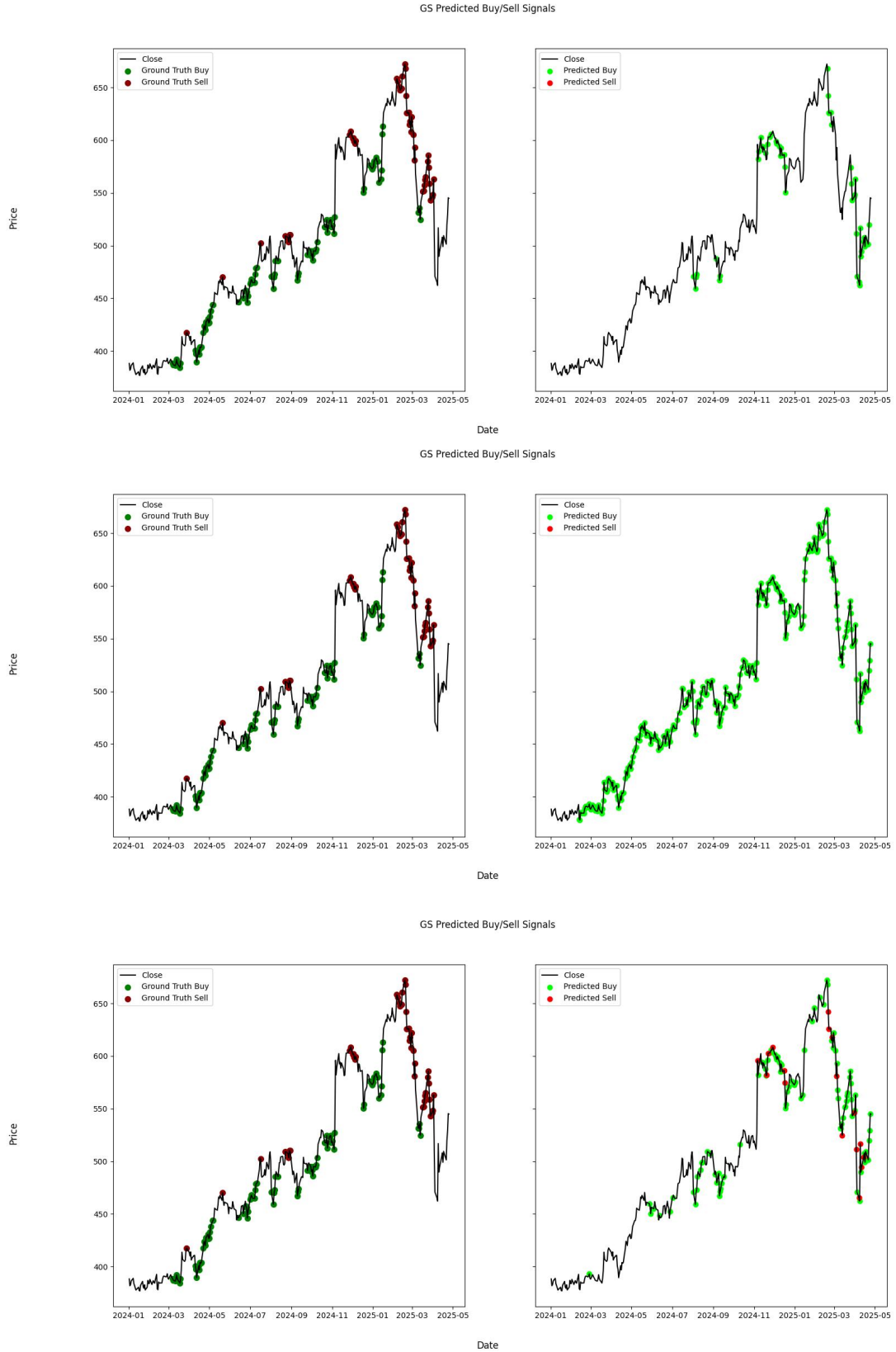

Figure 5: Performance of, from top to bottom, M_1, M_2, and M_3 on ticker GS (Goldman Sachs) over the whole 2024-2025 time period. In each row, the left plot shows the same ground truth labels and the right plot shows the model's predictions.

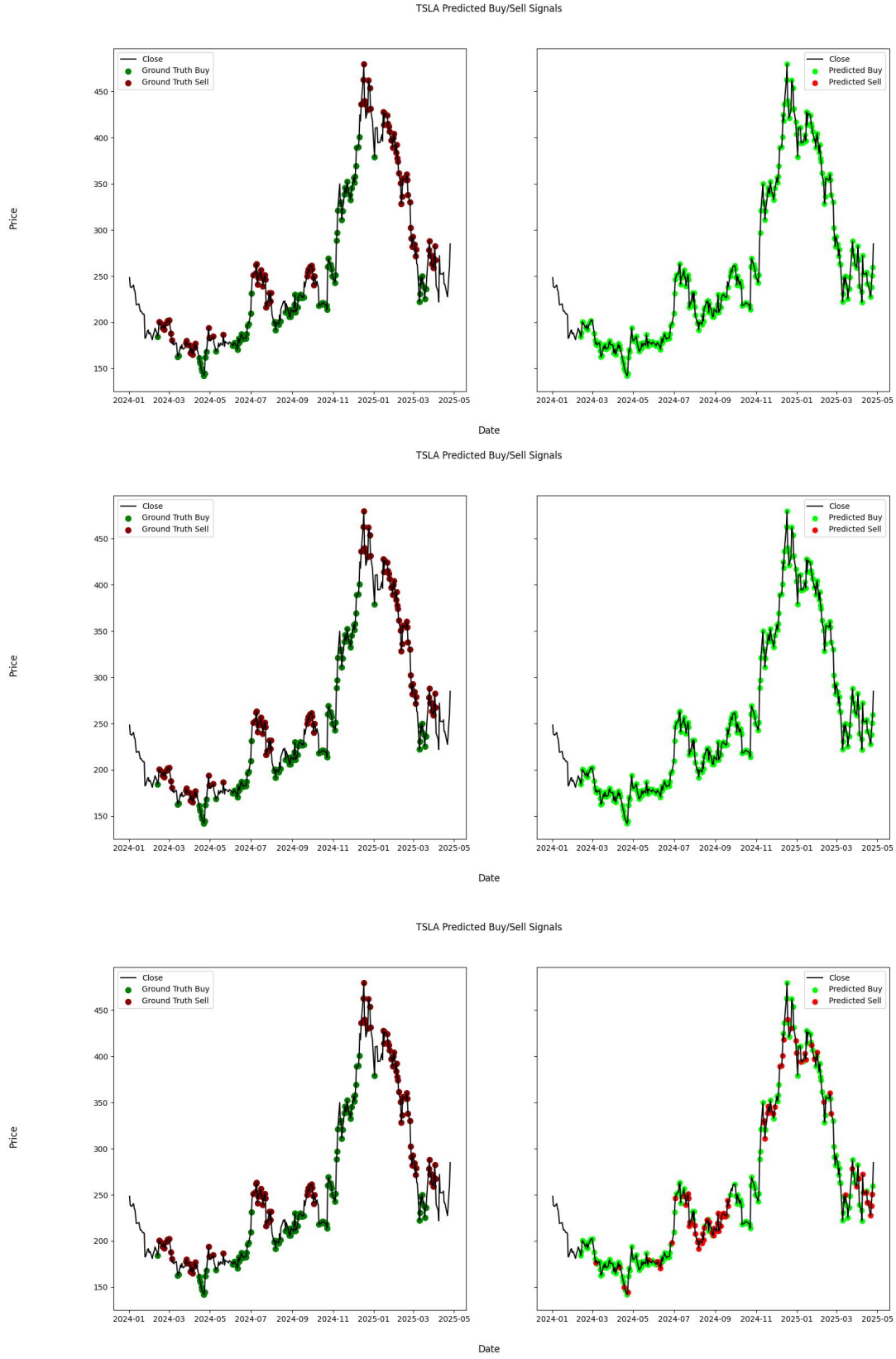

Figure 6: Same as Figure 5, but for ticker TSLA (Tesla).