Attentive Hierarchical Reinforcement Learning for Stock Order Executions

Karush Suri*
University of Toronto
karush.suri@mail.utoronto.ca

Shashank Saurav
University of Toronto
shashank.saurav@mail.utoronto.ca

Abstract

High frequency stock order executions require prior trading knowledge over longer periods in order to tackle volatility and address recency. This calls for domain-specific skills and awareness of price variation. We propose a novel stock order execution pipeline for S&P 500 stock sequences combining attention with Hierarchical Reinforcement Learning (HRL) for high-frequency market trading. The agent, consisting of two sub-agents, learns price prediction and order bidding hierarchies by attending over past values. Additionally, the agent learns to greedily exploit present market conditions for long-run payoffs. We compare our approach with a traditional HRL agent trained on the top 6 S&P 500 stock tickers during the COVID-19 outbreak. Competitive performance of the HRL agent utilizing attention depicts suitability of our algorithm over other modern-day RL methods.

1 Introduction

Long-term memory [7] has presented several successful used cases in price prediction [8, 10]. Utilizing sequential memory aids in evaluating trends and seasonality [9] as the recurrent cell receives inputs corresponding to past price values. Irrespective of its effectiveness, the model is biased towards sequential recency which calls for a gated architecture. We make use of this property of LSTMs to predict long-term prediction of prices. However, sequential memory is not robust to adverse changes in pricing and market behavior [22]. Moreover, supervised setting for order execution is not incentive dependent and thus, it requires attention from a reward-based standpoint.

A typical reinforcement learning setup is a reward-based scheme which allows the model to learn through experience [17]. Such training schemes can be combined with unsupervised methods to model markov processes as non-deterministic objectives [18]. Moreover, experiential tasks can be broken down into various sub-tasks which allows multiple models to learn various levels of skills that can be transferred to solve the problem task [3]. This results in domain-specific knowledge of agents capable of solving complex hierarchical tasks. Hierarchical Reinforcement Learning (HRL) can be further abstracted temporally between agents at the model [12] and state level [11]. Such an approach of distribution of tasks in cooperative and competitive games is called Multi-agent reinforcement learning [20]. We leverage the multi-agent hierarchical aspect in combination with LSTMs to model stock bidding as a long-horizon Markovian process.

2 Background

2.1 Long-Term Short-Term Memory

Fundamental RNN architectures are not very good at remembering dependencies because of the problems of exploding and vanishing gradients.The Long-Term Short-Term Memory (LSTM) architecture was designed for modeling memory-dependent information over long time periods. The memory cell,

being the entity capable of remembering information over time, has a linear activation function and a self-loop which is modulated by a forget gate which takes values between 0 and 1. The forget gate computes a linear function of its inputs, followed by a logistic activation function. If the gate is on (taking the value 1), the memory cell remembers its previous value, whereas if the forget gate is off, the cell forgets it. The connection from the input unit to the memory cell is gated by an input gate, which has the same functional form as the forget gate. The block produces an output, which is the value of the memory cell, passed through a tanh activation function. It may or may not pass the value on to the rest of the network. The connection is modulated by the output gate which has the same form as the input and forget gates. When we implement an LSTM, we have multiple vectors at each time step representing the values of all the memory cells and each of the gates. Mathematically, the computations are as follows:

$$\begin{pmatrix} i^{(t)} \\ f^{(t)} \\ o^{(t)} \\ g^{(t)} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} x^{(t)} \\ h^{(t-1)} \end{pmatrix}$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ g^{(t)}$$

$$h^{(t)} = o^{(t)} \circ \tanh(c^{(t)})$$

where $i^{(t)}$, $f^{(t)}$, $o^{(t)}$,$g^{(t)}$ represent the outputs of input gate,forget gate,output gate and memory cell respectively. The output of the LSTM cell is denoted by $c^{(t)}$ with $h^{(t)}$ being the hidden state.

2.2 Causal Self Attention

The attention module is a practical work-around from the bottleneck of information in sequential memory. The module allows LSTM cells to have access to the previous cell states and hidden states directly. Cells learn to address content based on annotations rather than position. We define a context vector $c^{(i)} = attention(Q, K, V)$ where Q, K and V are queries, keys and values respectively and are given by $Q = hW_i^Q$, $K = hW_i^K$ and $V = hW_i^V$. The weight coefficients α_{ij}, also known as attention weights, are computed using the squashing operation $\alpha_{ij} = softmax(\frac{QK^T}{\sqrt{d_k}})$ where $\sqrt{d_k}$ is the embedding dimension. Thus, the final expression for context $c^{(i)}$ becomes $c^{(i)} = softmax(\frac{QK^T}{\sqrt{d_k}})V$. In causal attention the current query attends to their past and current keys only.

2.3 Multi-Agent Reinforcement Learning and Q-Networks

We review the reinforcement learning setup wherein an agent interacts with the environment in order to transition to new states and observe payoffs by following a sequence of actions. The problem is modeled as a Stochastic Markov Game $(N, S, A^i, r^i, P^i, \gamma)$ where N is the number of players/agents indexed using i where $i \in [1, n]$, S is the state space, A^i is the action space corresponding to each agent, r^i is the reward observed by each agent such that $r^i : S \times A^i \rightarrow R$, P^i the transition model presenting the probability of transition from a given state to the nest state and γ is the discount factor. In the case of a fully-cooperative Markov game, all agents observe the same reward function, $r^1 = r^2 = ...r^n = r$. We consider a policy $\pi_{\theta^i}(a^i|s)$ for each agent as a function of our model parameters θ^i. At a given time t, the agent samples an action $a_t^i \sim A^i$ according to its policy $\pi_{\theta^i}(a_t^i|s_t)$ where $s_t \in S$, observes the reward $r(s_t, a_t^i)$ and transitions to the next state s_{t+1} according to the transition model $P(s_{t+1}|s_t, a_t)$. The agent aims to maximize the expected discounted payoff $\mathbb{E}_{\pi^i}[\sum_{j=1}^{\infty} \gamma^t r(s_t, a_t^i)]$ as a function of the parameter θ^i. The action-value function for an agent i is represented as $Q^{\pi^i}(s_t|a_t^i) = \mathbb{E}_{\pi^i}[\sum_{j=1}^{\infty} \gamma^{j-t} r(s_j, a_j^i)|s_j = s_t, a_j^i = a_t^i]$ which is the expected sum of payoffs obtained in state s_j upon performing action a_j^i by following the policy π_i. We denote the optimal policy $\pi^{i,*}$ such that $Q^{\pi^{i,*}}(s|a^i) \geq Q^{\pi^i}(s|a^i) \forall s \in S, a \in A$. In the case of multiple agents, the joint optimal policy can be expressed as the Nash Equilibrium [16] of the Stochastic Markov Game as $\pi^* = (\pi^{1,*}, \pi^{2,*}, ...\pi^{n,*})$ such that $Q^{\pi^*}(s|a^i) \geq Q^{\pi}(s|a^i) \forall s \in S, a \in A, i \in [1, N]$. Q-Learning [19] is an off-policy, model-free algorithm suitable for continuous and episodic tasks. The algorithm uses semi-gradient descent to minimize the Temporal Difference (TD) error [4]: $\mathbb{L} = \mathbb{E}[(Q(s_t, a_t) - y_t)^2]$ where $y_t = r_t + \gamma \max_{a' \in A} Q(s_{t+1}, a')$ is the TD target. Q-Network [15] is an

application of Q-Learning by using neural network as a function approximator for complex control tasks.

3 Hierarchical Attention for Stock Order Executions

3.1 Multi-Agent Hierarchical Setup

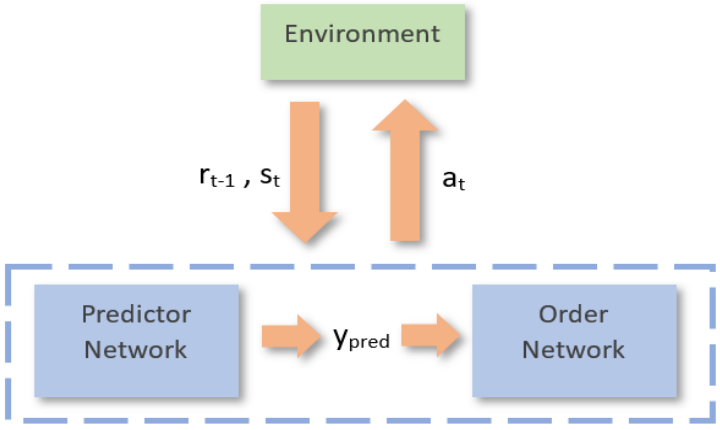


Figure 1: Multi-Agent Hierarchical setup between predictor and order networks

1 presents the proposed architecture for the multi-agent setup. For a goal oriented Markov Game $S, T, A^i, \gamma, P^i, r^i_\beta$ and an agent with parameters θ^i, our objective is to maximize $\mathbb{E}_{\pi^i}[\sum_{t=0}^{\infty} \gamma^t r^i_\beta(s_t, a^i_t)]; \quad \forall \beta \in T$. However, since the trading problem is modeled as a fully-cooperative game, $r^i_\beta = r^i = r; \quad \forall \beta \in T$, i.e- all tasks share a common reward function for all agents interacting in the environment. The set of tasks T consists of two individual tasks, $T = \{predict, order\}$. At a given time-step t, the predictor network ϕ^{pred} parameterized by θ^{pred} observes state s_t from the custom-designed trading environment (details provided in A.1). Reward r_{t-1} corresponding to the previous time-step $t-1$ along with past state-action pairs forming a trajectory τ are stored in an experience replay-buffer. Upon observing state s_t, ϕ^{pred}, following $\pi_{\theta^{pred}}(y_{pred}|s_t)$ predicts price y_{pred} of the stock sequence corresponding to next time-step. ϕ^{ord} observes s_t and the prediction offered by ϕ^{pred}, y_{pred}, in order to sample an action $a_t \sim \pi_{\theta^{ord}}(a_t|s_t)$ where $\pi_{\theta^{ord}}(a_t|s_t)$ is the policy of ϕ^{ord} parameterized by θ^{ord} at time-step t. Action a_t is executed in the environment by the dual-agent system which results in a transition to s_{t+1} via transition probability $p(s_{t+1}|s_t, a_t) \in P$.

3.2 Attention-based Order Executions

Attention has demonstrated significant improvement in sequential tasks such as sequence-to-sequence modelling [21], neural machine translation [2] and image captioning [23]. The attention module is a practical work-around from the bottleneck of information in sequential memory. The module allows LSTM cells to have access to the previous cell states and hidden states directly. Cells learn to address content based on annotations rather than position. We use causal attention in order to make the current hidden states attend to their past and current values. This allows the model to obey causality and prevent hindrance from future time-step values.

In the case of price prediction and order executions, we produce price context vectors consisting of past price values from the price sequence s_p. The price context vector, denoted by $c^{(i)}_p$ is obtained by scaling hidden states Q_p, K_p and V_p and then performing attention. Thus, $c^{(i)}_p = attention(Q_p, K_p, V_p) = softmax(\frac{Q_p K^T_p}{\sqrt{d_k}})V_p$. In our case, d_k is price embedding

dimension obtained from sequential processing of s_p. Upon implementation of attention, ϕ_{pred} yields y_{pred} corresponding to the next time-step $t+1$ which is then observed by ϕ_{ord}. ϕ_{ord} now attends to s_p and y_{pred}, i.e- $(s_p|y_{pred})$, where $(a|b)$ is the concatenation operation between a and b along the non-singleton dimension. This results in the execution of a_t from our dual-agent setup. Attention modules in ϕ_{pred} and ϕ_{ord} aid in improved gradient flow and aid the model to learn high frequency price changes. 0 presents the proposed HRL+Attention algorithm.

Algorithm 1 HRL+Attention

Initialize $s_0 \leftarrow env$,θ^{pred}, θ^{ord}, α, $batch\_size$
Initialize $\epsilon_{start} \leftarrow 1$,$\epsilon_{final} \leftarrow 0.01$, $buffer \leftarrow \{\}$,$step \leftarrow 3e4$, $epochs$
for all $epochs$ **do**
    $\epsilon \leftarrow \epsilon_{final} + (\epsilon_{start} - \epsilon_{final}) \exp{(\frac{-epoch}{step})}$
    $y_{pred} \leftarrow \phi_{pred}(s_t, \epsilon)$
    $s_t^{'} \leftarrow (s_t|y_{pred})$
    $a_t \leftarrow \phi_{ord}(s_t^{'}, \epsilon)$
    $s_{t+1}, r_t \leftarrow env(a_t)$
    append (s_t, a_t, r_t, s_{t+1}) to buffer
    $s_t \leftarrow s_{t+1}$
    if $buffer \geq batch\_size$ **then**
        $\theta^{pred} \leftarrow \theta^{pred} + \alpha[r_t + \gamma \underset{a}{max} Q(s_{t+1}, a_{t+1}, \theta^{pred}) - Q(s_t, a_t, \theta^{pred})] \nabla Q(s_t, a_t, \theta^{pred})$
        $\theta^{ord} \leftarrow \theta^{ord} + \alpha[r_t + \gamma \underset{a}{max} Q(s_{t+1}, a_{t+1}, \theta^{ord}) - Q(s_t, a_t, \theta^{ord})] \nabla Q(s_t, a_t, \theta^{ord})$
    end if
end for

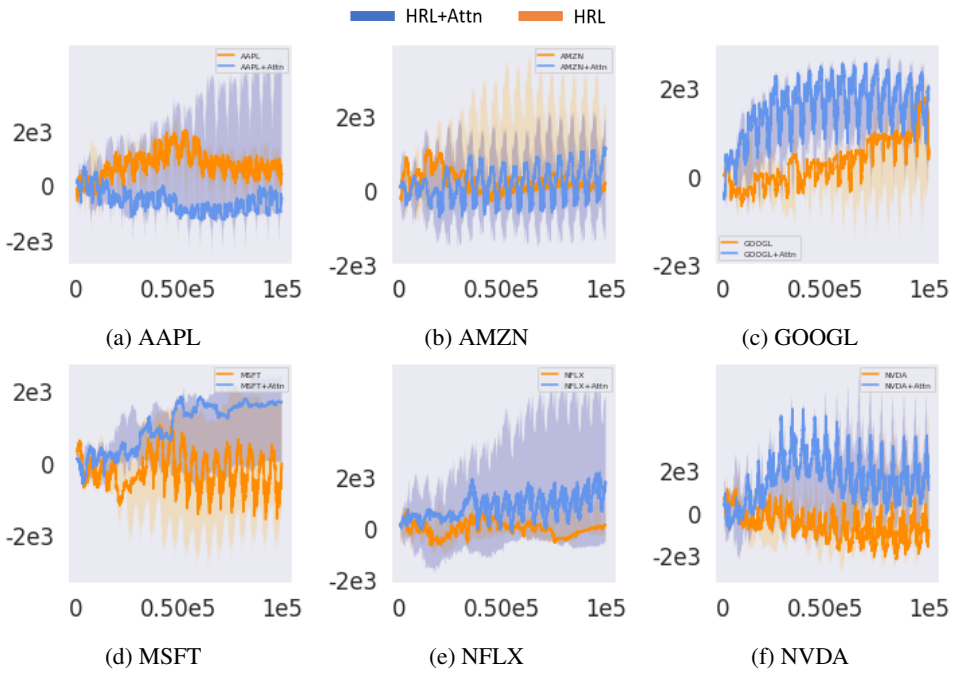
4 Experiments

Trading experiments were carried in a custom-designed trading environment (A.1). A total of six stock tickers from the S&P 500 [6] index, namely AAPL, AMZN, GOOGL, MSFT, NFLX and NVDA were considered for training and validation. Sequence corresponding to each ticker consists of price quotes at 2 minute intervals. These quotes represent the price of ticker at various instances in USD (Opening, Closing, High and Low). An additional sequence of Volume, which is the number of shares corresponding to the specific price, are also provided as input. Ticker sequences were used for the months during the COVID-19 outbreak, from January to February 2020. However, validation of performance was carried out for March 2020.

The agent is trained on normalized price values obtained from the environment at 2 minute intervals. Experience replay[15] was used for storing state-action trajectories for training. An exponential ϵ decay strategy [14] for the first 50K steps. Following exploration, the model is trained for 100K steps 4 times. At each time-step, ϕ_{pred} takes in s_t and yields y_{pred}. ϕ_{pred} consists of gated LSTM cells with causal attention modules followed by a fully-connected layer. y_{pred} is concatenated with s_t which serves as input state $s_t^{'}$ for ϕ_{ord} which provides a_t using the same architecture as ϕ_{pred} in conjunction with a softmax non-linearity. $a_t = 0$ denotes buying of shares whereas $a_t = 1$ represents a sale order. Each execution of a_t in the environment results in r_t (price at s_t) and the next-state s_{t+1}. Q-learning updates are carried out upon completion of each time-step. A complete implementation of the algorithm along with the hyper-parameters can be found at the project website[2]. A demo of the algorithm can be found in the tutorial[3].

[2]https://github.com/karush17/Hierarchical-Attention-Reinforcement-Learning

[3]https://nbviewer.jupyter.org/github/karush17/PyTorch-Tutorials/blob/master/Tutorials/Reinforcement%20Learning/2516_Project.ipynb



(a) AAPL (b) AMZN (c) GOOGL

(d) MSFT (e) NFLX (f) NVDA

Figure 2: Comparison of average rewards with and without attention (average over 4 inferences for 100K steps)- Out of the 6 sequences, agent learns to optimize profits for 5 of them by attending to combined weighted sequences rather than individual past vectors (as in the case of HRL) and learning seasonal variations. In the case of AAPL, agent fails to learn seasonality as a result of abrupt changes in the market.

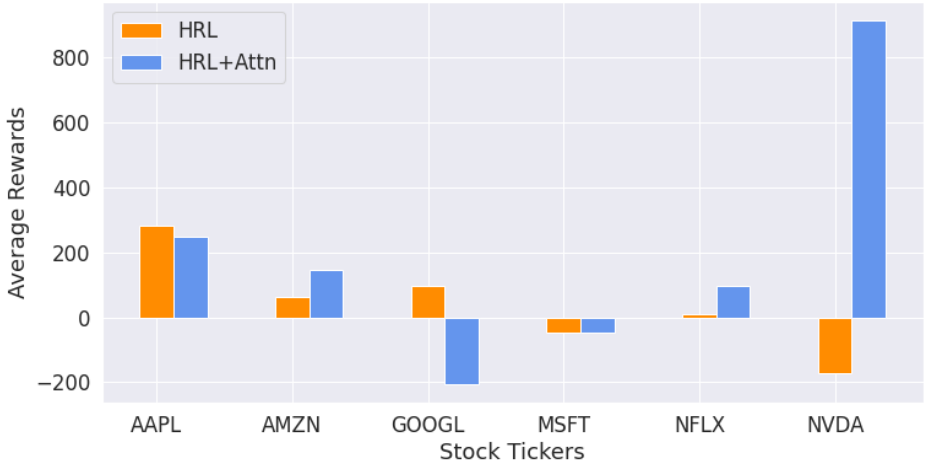


Figure 3: Average Reward over S&P 500 tickers for 4 validation inferences over 25K steps for March 2020- Price context vectors from attention enable robustness to volatility and facilitate accurate predictions and order executions. In the case of GOOGL, agent fixates its attention to adverse price changes, which is a sharp market dip, as a result of lack of seasonality.

2 presents the average rewards obtained during training by the agent for 6 S&P 500 sequences. Agent observes higher payoffs when coupled with attention for price context vectors $c_p^{(i)}$. Accurate predic-

tions yielded by ϕ_{pred} are attended by ϕ_{ord} which results in efficient action-selection and reduction in sparse rewards. Moreover, the agent presents robust performance for 4 out of 6 sequences as a result of attending to high price changes over a small time frame. Validation of agent's performance can be gauged by 3. The agent does significantly well for sequences in March 2020 which depict similar behavior to previous months. For instance, in the case of NVDA and NFLX, the agent estimates a continuous rise in price values and greedily produces buy order bids. On the other hand, agent estimates a sharp price fall for GOOGL which is found to be absent. This leads the agent to sell shares and hence, significantly degrade its performance. Detailed statistics and overall performance of the model is presented in A.2.

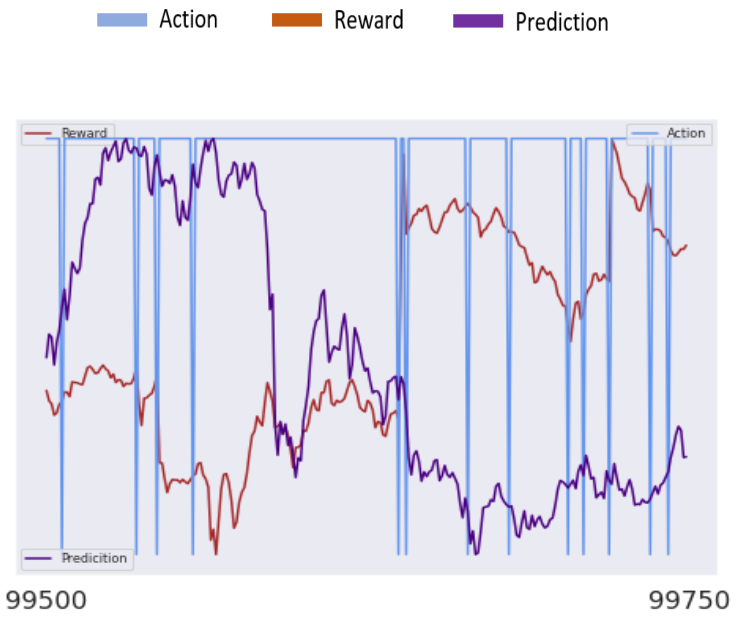


Figure 4: Intuition of Hierarchical Attention by action/prediction-reward mapping (1:Sell, 0:Buy)- Hierarchical breakdown of task processes allows agent to estimate price variations and learn greedy-trading strategies. By making use of attention, agent exploits sharp price falls and manipulates its actions to obtain higher payoffs.

4 presents the action-map for NFLX over 250 test steps. y_{pred} estimated by ϕ_{pred} can be mapped with a_t sampled from $\pi_{\theta^{ord}}(a_t|s_t)$. When $\pi_{\theta^{ord}}(a_t|s_t) = \pi^{ord,*}$ is the optimal greedy-wise trading strategy followed by ϕ_{ord}, the agent exploits sharp price variations by prediction estimations learned by ϕ_{pred} in order to produce a_t^* and observe higher rewards. The intuition can be broken down into 3 parts. Firstly, ϕ_{ord} observes estimated price hikes for a considerable amount of time and hence, continuously sells shares to obtain long-run rewards. Secondly, even small price dips predicted by ϕ_{pred} govern the agent to buy shares at lower prices in order to observe higher payoffs by selling them at later time-steps. Lastly, adverse price drops predicted by ϕ_{pred} force the agent to repeatedly buy shares by exploiting lower prices and greedily selling shares when observing price hikes.

5 Related Work

A number of implementations have been carried out in literature for stock order executions using Reinforcement Learning. [22] make use of DDPG algorithms in order to trade on a custom portfolio of 30 stock sequences. Policy-based approaches for low granularity sequences are scalable but do not present significant payoffs for sequences with high granularity and high volatility. [5] presents different forecasting and trading scenarios on a range of tasks such as portfolio management and forex-based transactions. These consist of different architecture requirements and variable horizons and present a suitable test-case for hierarchical problem modeling. While, major focus lies on

improving algorithmic design and long-horizon performance in the case of adverse state changes, [1] presents optimization strategies as scalable measures by making use of task-aware backpropagation methods in order to enhance the robustness of the agent. On the other hand, various approaches diverge from policy-based implementation to semi-supervised [24] and supervised [7] settings for a range of different price sequences [13] which are computationally effective for prediction tasks.

6 Conclusion

Hierarchical learning of the agent in combination with attention allows the agent to learn underlying market behaviors and adapt greedily to price variations. In the presence of attention, agent leverages its hierarchies to exploit present ticker prices for long-run payoffs. The attentive HRL pipeline is a suitable replacement for the traditional HRL trader. However, evaluating the approach for multiple granularities such as 5 minute or 30 minute price updates may yield better performance by the agent. Moreover, increasing the action space or hierarchies of the agent may lead to more human-like decisions such as holding a share for a fixed duration of time. This may result in greater long-term rewards. We leave this as our future work.

References

[1] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE Transactions on Neural Networks and Learning Systems*, 28:1–12, 02 2016.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13(1):227–303, Nov. 2000.

[4] I. C. Dolcetta and H. Ishii. Approximate solutions of the bellman equation of deterministic control theory. *Applied Mathematics and Optimization*, 11:161–181, 1984.

[5] T. G. Fischer. Reinforcement learning in financial markets - a survey. FAU Discussion Papers in Economics 12/2018, Erlangen, 2018.

[6] U. Habibah, S. Rajput, and R. Sadhwani. Stock market return predictability: Google pessimistic sentiments versus fear gauge. *Cogent Economics & Finance*, 5(1):1390897, 2017.

[7] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[8] B. Huang, Q. Ding, G. Sun, and H. Li. Stock prediction based on bayesian-lstm. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, ICMLC 2018, page 128–133, New York, NY, USA, 2018. Association for Computing Machinery.

[9] H. Jia. Investigation into the effectiveness of long short term memory networks for stock price prediction. *CoRR*, abs/1603.07893, 2016.

[10] H. Y. Kim and C. H. Won. Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models. *Expert Systems with Applications*, 103:25–37, 2018.

[11] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *CoRR*, abs/1604.06057, 2016.

[12] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, page 464–473, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

[13] H. M, E. Gopalakrishnan, V. Menon, and S. Kp. Nse stock market prediction using deep-learning models. *Procedia Computer Science*, 132:1351–1362, 01 2018.

[14] A. Maroti. Rbed: Reward based epsilon decay, 2019.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[16] J. F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[17] J. Randlov. Learning macro-actions in reinforcement learning. In M. J. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 1045–1051. MIT Press, 1999.

[18] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. P. Reichert, N. C. Rabinowitz, A. Barreto, and T. Degris. The predictron: End-to-end learning and planning. *CoRR*, abs/1612.08810, 2016.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[20] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente. Multiagent cooperation and competition with deep reinforcement learning. *CoRR*, abs/1511.08779, 2015.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[22] Z. Xiong, X. Liu, S. Zhong, H. Yang, and A. Walid. Practical deep reinforcement learning approach for stock trading. *CoRR*, abs/1811.07522, 2018.

[23] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044, 2015.

[24] X. Zhou, Z. Pan, G. Hu, S. Tang, and C. Zhao. Stock market prediction on high-frequency data using generative adversarial nets. *Mathematical Problems in Engineering*, 2018:1–11, 04 2018.

A Appendix

A.1 Trading Environment Design

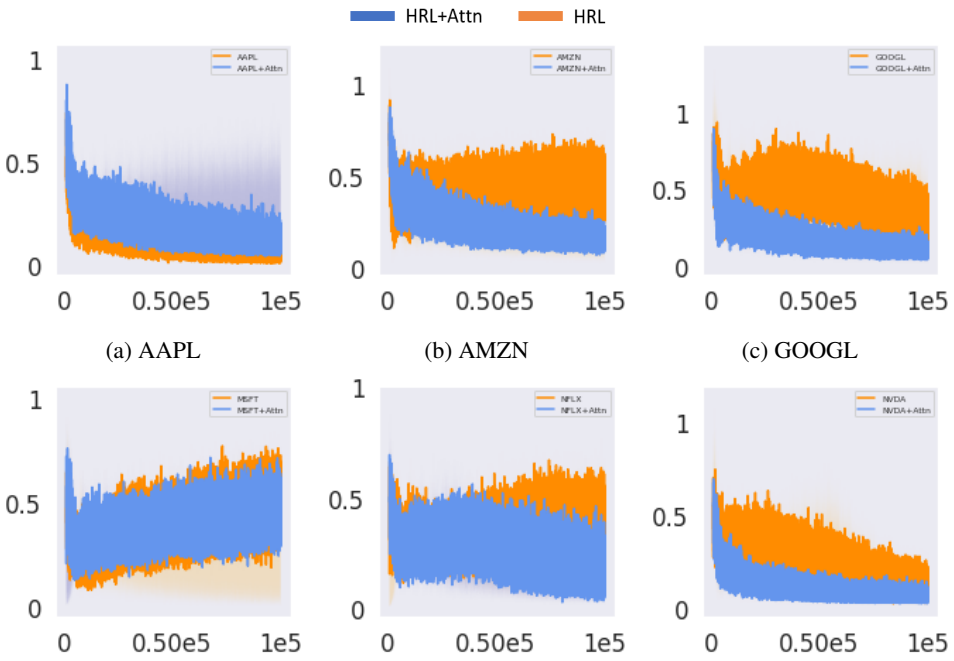
Algorithm 2 TradeEnv

Input: S, a_t
Output: r_t, s_{t+1}
Initialize account_balance, max_account_balance, max_num_shares, max_share_price, max_open_positions
Initialize net_worth, shares_held, shares_sold, cost_basis, sales_value
/ / / / / / / / / / / / / FETCH NEXT OBSERVATION FROM SEQUENCE
function OBSERVATION(S)
    $obs \leftarrow \frac{S[-5:]}{max\_share\_price}$
    $obs \leftarrow (obs | [balance, net\_worth, shares\_held, shares\_sold, cost\_basis, sales\_value])$
    return obs
end function
/ / / / / / / / / / / / / EXECUTE ACTION BY BUYING/SELLING SHARES
function STEP(S, a_t)
    if $a_t == 0$ **then**:
        $shares\_bought \leftarrow \frac{balance}{S[-1]} \times amount$
        $prev\_cost \leftarrow cost\_basis \times shares\_held$
        $add\_cost \leftarrow shares\_bought \times S[-1]$
        $balance \leftarrow balance - add\_cost$
        $cost\_basis \leftarrow \frac{prev\_cost + add\_cost}{shares\_held + shares\_bought}$
        $shares\_held \leftarrow shares\_held + shares\_bought$
    else if $a_t == 1$ **then**:
        $shares\_sold \leftarrow shares\_held \times amount$
        $balance \leftarrow balance + shares\_sold \times current\_price$
        $shares\_held \leftarrow shares\_held - shares\_sold$
        $sales\_value \leftarrow shares\_sold \times current\_price$
    end if
    $net\_worth \leftarrow balance + shares\_held \times current\_price$
    $reward \leftarrow balance$
    $obs \leftarrow OBSERVATION(S)$
    return reward, obs
end function
/ / / / / / / / / / / / / RESET ENVIRONMENT TO A RANDOM STATE
function RESET(S)
    $balance \leftarrow account\_balance$
    $net\_worth \leftarrow account\_balance$
    $shares\_held \leftarrow 0$
    $shares\_sold \leftarrow 0$
    $sales\_value \leftarrow 0$
    $cost\_basis \leftarrow 0$
    $obs \leftarrow observation(S)$
    return obs
end function

A.2 Model Performance

Primary experimentation for price prediction and trading was carried out on the S&P 500 dataset using Q-Learning. The first sub-agent performs the task of predicting the price at next time-step by taking the input as past 6 prices in the sequence. The predicted output, concatenated with the original state (past 6 prices) is passed to the second sub-agent which outputs an action.

Long-horizon optimization of the agent is an essential component for profit maximization in the long-run. 5 presents average loss of the agent over various sequences.

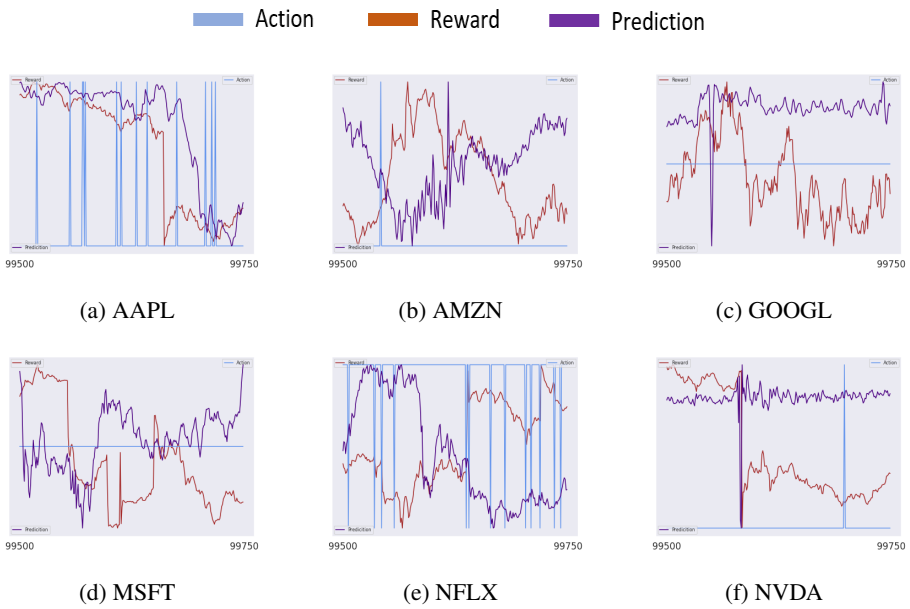


(a) AAPL (b) AMZN (c) GOOGL

(d) MSFT (e) NFLX (f) NVDA

Figure 5: Comparison of average loss with and without attention (average over 3 inferences)- Attentive agents learn to perform better following the exploration stage and converge to a minimum objective. In the case of MSFT, some overfitting is observed which is primarily a case of frequent price changes towards the later half of the sequence.

Action selection of agents using attention can further be assessed by comparing the actions selected by the second sub-agent against the predictions offered by the first sub-agent. This, in total, can be mapped with the reward achieved at the given time-step.

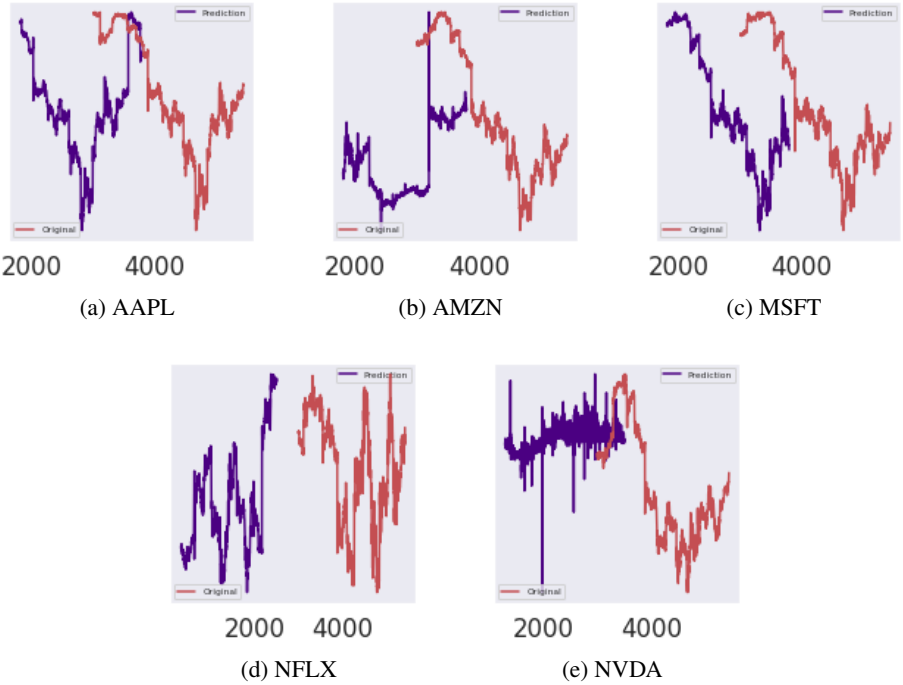


(a) AAPL (b) AMZN (c) GOOGL

(d) MSFT (e) NFLX (f) NVDA

Figure 6: Action maps of the agent corresponding to actions selected as a result of price predictions (1: Sell, 0: Buy)- Agent is able to make decisions based on future predictions. For instance, in the case of NFLX, the first sub-agent predicts a sharp drop in pricing as a result of which it sells its shares multiple times and achieves higher rewards.

Attention of the first sub-agent can further be assessed by comparing the price forecasts against the original market sequence.7 presents the comparison of price prediction against the original sequence.



(a) AAPL (b) AMZN (c) MSFT

(d) NFLX (e) NVDA

Figure 7: Predictions offered (in purple) by the agent under high volatility of the sequence (presented in brown)- Agent attends to sharp dips, sudden increases and abnormal changes in price variation over shorter time intervals. However, predictions may deviate from their true value at certain instances.

		HRL				HRL+Attention			
Ticker Name	std	Avg. R	Peak R	Mean Loss	Sharpe Ratio	Avg. R	Peak R	Mean Loss	Sharpe Ratio
AAPL	15.31	**282**	2841	0.05	6.99	248	**3320**	0.10	6.15
AMZN	113.65	64	758	0.27	0.21	**145**	**1013**	0.18	0.48
GOOGL	61.49	**96**	**1893**	0.37	0.59	-207	1817	0.12	2.08
MSFT	8.82	-46	**1427**	0.38	3.23	**-46**	212	0.50	3.23
NFLX	14.24	9	713	0.27	0.39	**95**	**1910**	0.33	4.13
NVDA	20.12	-172	3300	0.10	5.13	**916**	**3510**	0.05	28.22

Table 1: Comparison of Attentive Hierarchical performance with conventional Hierarchical framework