Scientific Research Publishing

Journal of Data Analysis and Information Processing, 2020, 8, 86-98
https://www.scirp.org/journal/jdaip
ISSN Online: 2327-7203
ISSN Print: 2327-7211

Meta-Learning of Evolutionary Strategy for Stock Trading

Erik Sorensen, Ryan Ozzello, Rachael Rogan, Ethan Baker, Nate Parks, Wei Hu

Department of Computer Science, Houghton College, Houghton, NY, USA
Email: wei.hu@houghton.edu

How to cite this paper: Sorensen, E., Ozzello, R., Rogan, R., Baker, E., Parks, N. and Hu, W. (2020) Meta-Learning of Evolutionary Strategy for Stock Trading. *Journal of Data Analysis and Information Processing*, **8**, 86-98.
https://doi.org/10.4236/jdaip.2020.82005

Received: January 12, 2020
Accepted: May 22, 2020
Published: May 25, 2020

Abstract

Meta-learning algorithms learn about the learning process itself so it can speed up subsequent similar learning tasks with fewer data and iterations. If achieved, these benefits expand the flexibility of traditional machine learning to areas where there are small windows of time or data available. One such area is stock trading, where the relevance of data decreases as time passes, requiring fast results on fewer data points to respond to fast-changing market trends. We, to the best of our knowledge, are the first to apply meta-learning algorithms to an evolutionary strategy for stock trading to decrease learning time by using fewer iterations and to achieve higher trading profits with fewer data points. We found that our meta-learning approach to stock trading earns profits similar to a purely evolutionary algorithm. However, it only requires 50 iterations during test, versus thousands that are typically required without meta-learning, or 50% of the training data during test.

Keywords

Meta-Learning, MAML, Reptile, Machine Learning, Natural Evolutionary Strategy, Stock Trading

1. Introduction

Accessibility to large data stores for developing effective machine learning models is a valuable source of profit and insights for numerous industries and areas of research. However, the current requirements to have a lot of data for training deep learning models are a prevalent crux of applying machine learning to problems where little data is available or there are restrictive windows of relevant data on which to be trained. An example of this is the stock industry where the most relevant data for stock forecasting are the past few days or weeks. Meta-learning was introduced to solve this problem [1]. In meta-learning, the initial step is for the algorithm to train on samples of a distribution of similar tasks such that it generalizes to those tasks. After building the meta-model, the algorithm can learn to solve a specific task with only a few data points and a few iterations [2].

We have applied two different meta-learning algorithms, Modal-Agnostic Meta-Learning (MAML) [3] and Reptile [4], in the area of stock trading. While both MAML and Reptile use a gradient in the evolutionary strategy (ES) algorithm to train on each task, they differ in how they do meta-learning. MAML takes the second-order gradient whereas Reptile uses the first-order gradient. Rather than using a typical gradient descent approach in reinforcement learning, we employed the evolution strategy [5], which works with a population of possible best-fit solutions to find the maximum reward. Using ES with a meta-learning approach towards stock trading, we show how once a meta-model is trained using MAML or Reptile, afterwards training on a specific task requires significantly fewer iterations and less data for accurate and profitable trading.

While machine learning has been applied to the stock trading in the past [6] [7] [8], there is much exploration required to discover how meta-learning performs in this area. In our research, we use industry sector stock data to train a meta-model for stock trading. The goal of our algorithms is to meta-model a collection of stocks (trading data) well enough to make profitable purchases and sales of new stocks (test data) with an initial amount of money that is allotted for investing.

1.1. Reinforcement Learning Overview

Reinforcement learning (RL) teaches an agent to learn how to act from its interactions with an environment. The goal of the agent is to learn an optimal policy that maximizes its long-term rewards. A policy refers to the way that the agent takes an action in a given state [9].

1.2. Evolution Strategies as a Scalable Alternative to Reinforcement Learning

Evolutionary strategies are an alternative to stochastic gradient descent that is used in reinforcement learning to optimize the policy. These strategies were created from the family of evolutionary algorithms [10]. We apply evolutionary strategy to optimize the objective function $f(x)$ in reinforcement learning for stock trading. Natural gradient (therefore NES) works with a probability distribution space parameterized by θ, $p_\theta(x)$. It searches for the steepest direction within a small step in the distribution space. To find an optimal solution to the function, the following iterations are employed [5]:

1) Generate a population of samples $D = (x_i, f(x_i))$ where $x_i \sim p_\theta(x)$.

2) Evaluate the "fitness" of samples in *D*.

3) Select the best subset of individuals and use them to update θ, generally

based on fitness or rank.

In order to evaluate the "fitness" of our samples in step 2, we apply a "jitter" to each of the samples from step 1. The "jitter" is created through random variance based on a hyper parameter, sigma, which controls how much random variance we have in "jittering". Then, based on the samples chosen with this random variation we can do step 3 and select the samples that are best optimized to the function $f(x)$.

It has been shown that ES scales very well, which allows us to train deep networks to solve complicated reinforcement learning problems. ES has also been found to be a great black box optimization technique when the objective function does not have an analytic form. These factors make ES an effective alternative to deep reinforcement learning and also simpler to implement as there is no need for backpropagation [10].

1.3. Meta Reinforcement Learning as a Response to Overfitting in Traditional Reinforcement Learning

Common in RL algorithms is their tendency to overfit [11]. RL struggles when it is faced with similar environments and needs to completely relearn in order to adjust. Meta-reinforcement learning (meta-RL) aims to fix this problem by programming a computer on how to learn in more general environments so it does not have to completely relearn [12]. For example, say we have a two-armed bandit, where one arm gives a reward and the other doesn't. RL is easily able to learn which hand gives the reward but will fail if we introduce it to another two-armed bandit where the rewarding arm has switched. It had grown accustomed to pull a specific arm rather than considering the possibility that the situation had changed. Meta-RL would learn how to approach a general two-armed bandit problem rather than a specific one, giving it the ability to adjust when the bandit arm is different. In meta-RL, the algorithm first trains on a collection (distribution) of similar tasks, and then it can learn a new task fast [12].

2. Methods for Stock Trading with Meta-Learning

The real-world stock data we used was taken from Yahoo Finance beginning on January 2nd, 2001. All the stock data was from the consumer cyclical sector (CC sector), where the stocks were priced above \$50 in 2019. We chose the CC sector because those stocks are highly dependent on business cycles and economic conditions. We chose stocks of prices above \$50 because we wanted to be consistent in the data that we were using and because stocks with higher prices tend to be less erratic. One thing to note is that the stocks used were priced above \$50 in 2019, but a lot of the data used included stocks that were priced below \$50 before 2019.

One possible drawback of sticking to a particular sector (the CC sector) is that the learning algorithms may struggle with stocks that are not consumer cyclical.

The stocks in the CC sector are typically industries such as automotive, housing, entertainment, and retail. Non-cyclical stocks tend to do relatively better when the economy is doing poorly since they do not depend on how well the economy is doing (an example of a non-cyclical stock is utilities)[1].

Most of the stock data (50 stocks) is used for meta-learning. This is called the training dataset. We test the meta-training with a test dataset, which is the data of a single stock. This is done by using the meta-model developed from the training dataset and the first 70% of the test dataset to predict the last 30%. We used both MAML and Reptile methods for stock trading. Our state is a window size of n days of stock closing prices. Based on these prices, our action is to trade on the following day. The NES algorithm[2] trains our neural network to learn the optimal policy with a function that rewards a profitable trade and that punishes loss. Our reward function is the amount of profit or loss over our initial starting money and is defined as follows:

$$\text{reward} = \left(\frac{(\text{current} - \text{initial})}{\text{initial}}\right) * 100 \tag{1}$$

We use the latest window with the meta-learned model to buy, sell, or hold stocks.

2.1. MAML Method for Stock Trading

Our method was constructed using an adapted version of the NES trading algorithm to make it a meta-learned model based on the MAML algorithm seen in Algorithm 1 [3]. The NES algorithm acts as the learner for trading one stock. The meta-model's goal is to learn to trade on a collection of stocks,

Algorithm 1. Model-agnostic meta-learning.

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
2: while not done do
3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4:     for all do
5:         Evaluate $\nabla_\theta L_{\mathcal{T}_i}(f_\theta)$ with respect to K
           examples
6:         Compute adapted parameters with
           gradient descent: $\theta'_i = \theta - a\nabla_\theta L_{\mathcal{T}_i}(f_\theta)$
7:     end for
8:     Update $\theta \leftarrow \theta - \beta\nabla_\theta \Sigma_{\mathcal{T}_i \sim p(\mathcal{T})} L_{\mathcal{T}_i}(f_{\theta'_i})$
9: end while
```

[1]Further information on the CC sector can be found here at https://www.investopedia.com/terms/c/consumer_cyclicals.asp and https://www.thebalance.com/understanding-cyclical-and-non-cyclical-stocks-3141363.
[2]Our NES implementation uses Husein Zolkepli's work at https://github.com/huseinzol05/Stock-Prediction-Models/blob/master/agent/updated-NES-google.ipynb.

updating the weights (θ) according to the action and reward received in the environment. We then model updating θ using the MAML algorithm. To do this, we first randomly initialized our θ. We then batched the tasks T_i, where each task is an individual stock in our chosen stock industry of Consumer Cyclical. We then loop through each of the batched individual stocks or tasks and use the NES algorithm to perform natural evolutions, which updates θ based on training from the environment on individual stocks from T_i. After all θ is updated, we loop through each θ to update the global θ using the NES algorithm. Now, θ will be updated according to the trained θ for each of the individual stocks, thereby learning the general trends of all the tasks or stocks in the Consumer Cyclical industry.

The MAML algorithm was meta-trained to build a meta-model using a dataset of 50 stocks within the Consumer Cyclical industry with 180 days (actual dates: 1/2/2001 to 9/24/2001) of company closing prices starting January 2nd, 2001. Our test stock was called ABC and was not present in the training. During meta-test, the NES algorithm was trained on 70% of the dataset with the help of meta-model and then did its actual trading on the next 30% of the data.

The parameters in the NES algorithm that we used were a population size of 15, a learning rate of 0.03, and a sigma of 0.1. We limit the algorithm to buying or selling 5 shares at a time to stabilize training, with an initial investment value of $10,000.

When implementing the NES algorithm with MAML, we set up a single experiment. Our goal was to see the difference in rewards between the NES classical version versus the NES with MAML over the same number of epochs during the training process. We ran both algorithms over 10,000 epochs during their training phase. However, the NES classical trained only on 70% the ABC stock while the NES with MAML meta-trained on 50 stocks within the Consumer Cyclical industry. Additionally, we trained the NES with MAML with only 50 epochs on the ABC stock. We then graphed the results of simulated trading for both algorithms on the last 30% of the ABC stock, which was never used in training, and was therefore unseen data to both algorithms. The final comparison we visualized was the performance in trading, or the algorithm's ability to decide between the actions to buy or sell the stock at precise time steps.

2.2. Reptile Method for Stock Trading

Reptile is a first-order derivative approximation of MAML, which requires a second-order derivative [13], that performs similarly but with fewer calculations, allowing it to be more efficient in its applications. Reptile works by initializing ϕ, the vector of initial parameters, and then iterating through these three steps [4]:

1) For each task, T, evaluate the loss, L_T on the weight vectors, W.

2) Compute $W = SGD(L_T, \phi, k)$.

3) Update $\phi \leftarrow \phi + \epsilon(W - \phi)$.

Essentially, Reptile uses gradient descent to find a set of weights that has been generalized over a set of tasks. When the network is initialized with these weights and is applied to a test set, it can quickly learn that task with much less data than a typical machine learning algorithm and with less computational intensity than MAML.

Our Reptile implementation for stock trading used the same stock data for meta-training as in section 2.1. We define one task as trading on one kind of stock (*i.e.* how to trade WDC stock) based on the closing prices of some number of days in the past. Following the meta-learning philosophy, we trained the NES with Reptile on 45 of CC sector stocks such that the algorithm generalizes its approach to trading within the whole sector and is able to trade on new stocks in the same sector after only a few more iterations of training on a specific stock.

The whole workflow of our meta-learning trading algorithm can be seen in Figure 1. The primary structure of our Reptile code was taken from [14]. Our contribution consisted of applying Reptile to the NES algorithm to find the maximum profit from trading. After initializing the model, we defined a set of global weights by which to track the meta-learning process as the machine trained briefly on each stock. After this meta-training, we did meta-testing by picking 70% of one new stock to train the NES model and tested it using the

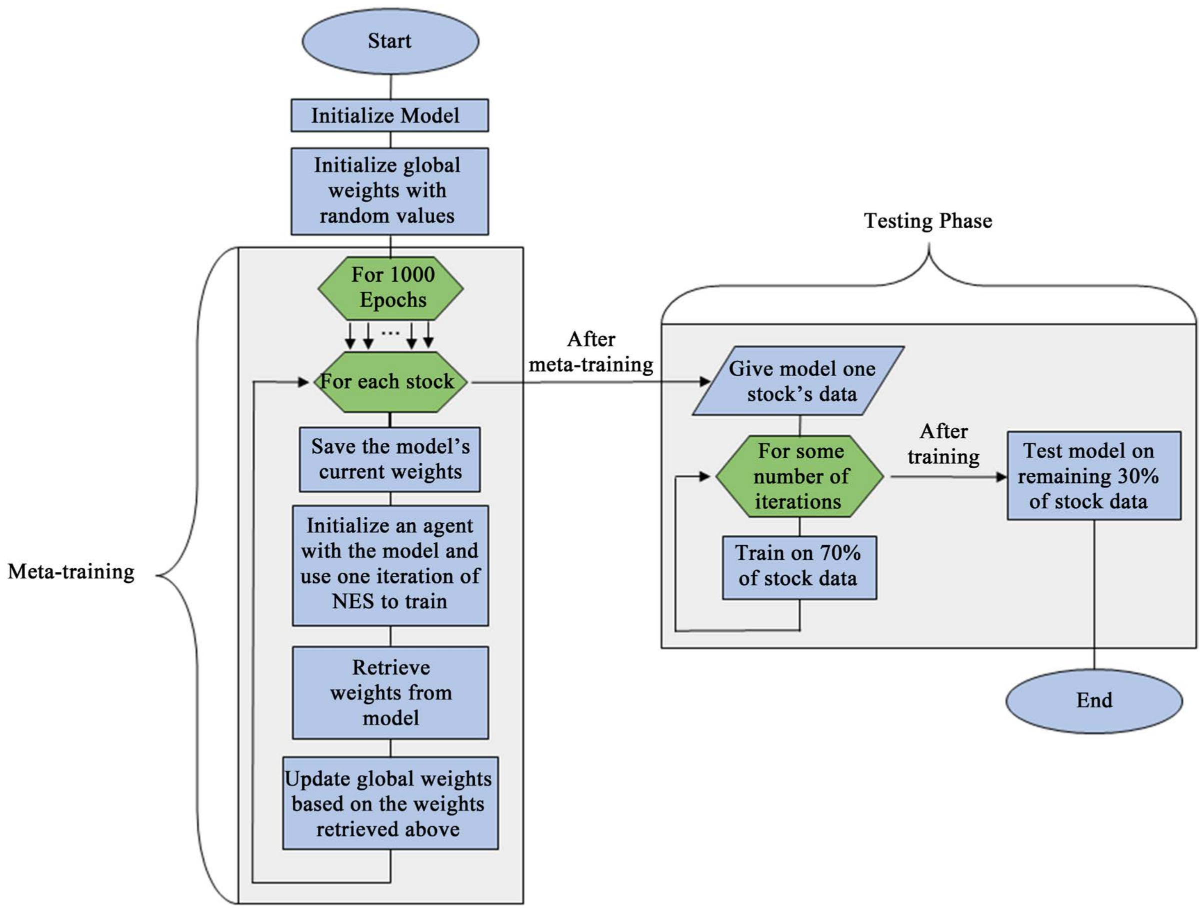

Start
Initialize Model
Initialize global weights with random values
For 1000 Epochs
For each stock
Save the model's current weights
Initialize an agent with the model and use one iteration of NES to train
Retrieve weights from model
Update global weights based on the weights retrieved above
Meta-training
After meta-training
Testing Phase
Give model one stock's data
For some number of iterations
Train on 70% of stock data
After training
Test model on remaining 30% of stock data
End

Figure 1. Meta-learning trading flowchart.

remaining 30% to see how it performed. For each time we meta-trained the algorithm, and ran this testing portion 10 times. We ran the meta and testing portion as a whole 10 times for each set of parameters we used to generalize our findings. Our neural network model consisted of four layers, each with a size of 500. For the agent, we started with $10,000 with which to trade, a maximum of 5 shares in a single day (to prevent extreme behaviors of buying and selling), and an incrementation rate of 1 day for the sliding window of closing prices. We also used a population size of 15, a learning rate of 0.03, and a sigma of 0.3. We found that the key to effective meta-training is to use many epochs and few iterations of the NES algorithm to ensure that the model does not overfit to a single stock. Therefore, although we used one iteration of the NES algorithm for each individual stock of the 45, we trained the model on these 45 stocks with 1000 epochs. This was done to maximize the model's understanding of the entire sector and to give it more flexibility and accuracy during testing.

Since we wanted to see if using the NES strategy alongside Reptile would perform better than the NES algorithm alone at very few iterations, we set up two experiments. First, in order to determine if including Reptile increases performance, we ran the testing portion with and without prior meta-training. This would help us see if the meta-training helped the algorithm make better trading decisions at few iterations with it rather than without it. Along with the parameters described above, we trained and tested on 60 days' worth of closing prices (actual dates: 1/2/2001 to 3/28/2001), used a window size of 10 days, and set the number of pre-trading training iterations to 50. Thus, our meta-training portion trained on all 60 days, and, during test, the model trained on 42 (70%) days while 18 (30%) days were held for trading. For the trial that included the meta-training, we used these same parameters for the meta portion.

Our second experiment was to test whether the NES strategy with Reptile can perform the same as only the NES algorithm given less data. For this, like mentioned above, we compared how the algorithm performed with and without meta-training. When we meta-trained the algorithm, we implemented the testing portion with half as much data as we gave the testing portion of the trial without the meta-training, but we used the same number of iterations for the testing portions of both. Specifically, for both trials, we used 100 iterations of training for the test portion with a window size of 15 days. For the run with the meta-training, we trained on 80 days (actual dates: 1/2/2001 to 4/26/2001). For test, our model with Reptile trained on 24 days before trading, while the run without the meta-training trained on 58 days before trading. We then traded on 24 days to see how it performed.

3. Results of MAML and Reptile Meta-Learning Strategies

We found that the NES with MAML resulted in smoother training and that NES with Reptile led to profitable trading with a fewer number of test iterations and data. Both of these results suggest the applicability of meta-learning strategies to improve stock trading with the NES algorithm.

3.1. MAML Results for Stock Trading

Our NES algorithm with MAML has an advantage over the base NES algorithm in that it can learn how to trade over multiple stocks in the same sector. Therefore, its training was smoother than the NES algorithm without MAML as depicted in Figure 2. However, it did not perform as well as NES did even with the same parameters for training. This is most likely because the NES algorithm is overfitting to the ABC stock, whereas the meta-model is training on 50 different stocks at a time.

Additionally, shown in Figure 3, when running our back-test trading on new stock data, the base NES algorithm performed better than that with MAML. The equation we used to calculate the market value of the stock is the difference in

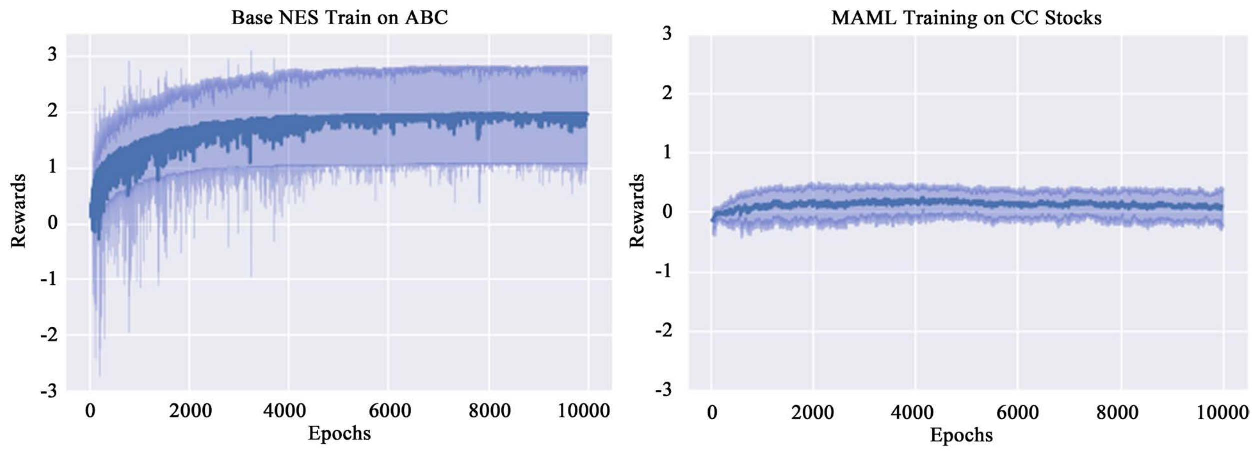


Figure 2. Compare the training of the NES alone (left plot) and NES with MAML algorithms (right plot). The graphs show the mean and standard deviation of the rewards over 10 iterations of training. NES trains on just the ABC stock while NES with MAML trains on 50 stocks in the CC sector, the same sector as the ABC stock. As we can see NES with MAML has less variance (0.2 std vs 2 std).

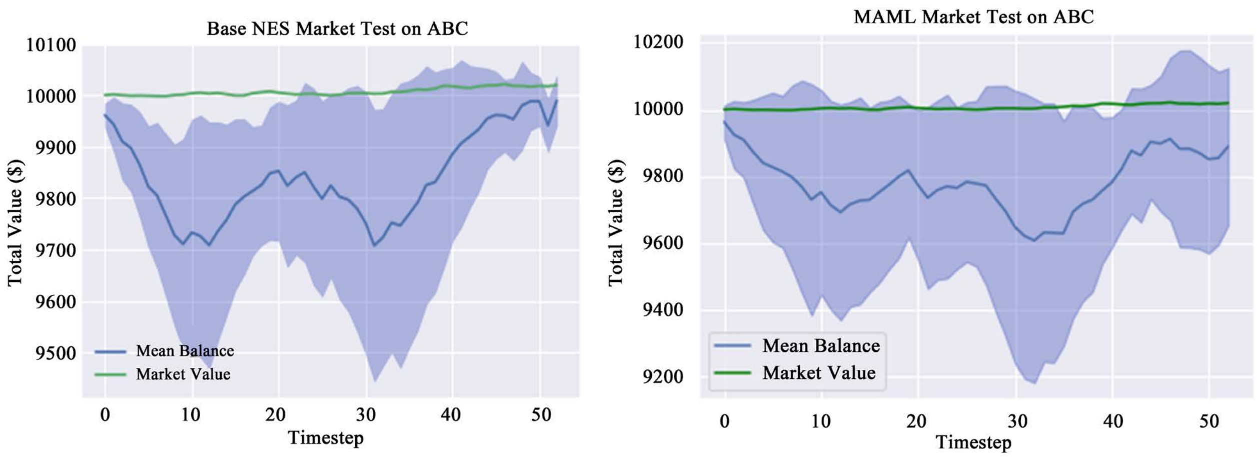


Figure 3. Comparing the market testing of the NES (left plot) and the NES with MAML (right plot) algorithms. The graphs show the upper bound of the market value of the ABC stock without any trading (green) and the mean balance (blue) and std (light blue) of our algorithm over 10 tests. The test was done over 53 (30% of 180) days.

the value of the stock from the first day of trading multiplied by the maximum number of shares that can be bought in a single day (in our case 5 shares) plus our initial investment amount of $10,000. The equation to calculate the upper bound of the market value of a stock on a given day is defined as follows:

$$\text{market value} = (\text{current} - \text{initial}) * 5 + 10{,}000 \quad (2)$$

After 53 (30% of 180) days of trading on the ABC stock, the ABC market value, starting at $10,000, ended up at a market value of $10,019 after simulating buying 5 shares at the beginning and holding. This emulates the same limit of buying stocks to which our algorithm is held. This ensures that the trading is on the same scale when plotting. The NES algorithm without MAML achieved an average value over 10 tests of $9989 and the NES algorithm with MAML achieved an average value of $9889.59. The blue lines in these graphs track the total balance of money throughout the trading process, so they fluctuate with stocks bought and sold. Since the algorithm's goal is to maximize profit by ending with as high of a balance as it can, these graphs reveal to us the trading process and the final balance in comparison with the market value of 5 shares.

3.2. Reptile Results for Stock Trading

We found that using the NES with Reptile for stock trading increased performance with very few test stage iterations when compared to using the NES alone. Regarding our first experiment, we wanted to emphasize that using NES with Reptile before training on a specific stock increased the performance on trading that stock. When we trained the algorithm with one stock at 50 iterations before testing, the model that had prior meta-training with Reptile performed $1.24 better than the model that only trained with the NES approach. To measure performance in our first test using only 50 iterations of training before testing, we compared the profits made during trading over 18 days of Western Digital Corp (WDC) stock with the NES algorithm with and without Reptile. With Reptile, averaging ten tests, our model made $1.47 while without Reptile, it made $0.23 (**Figure 4**).

When we tested the performance between NES with Reptile with half of the testing training data and NES alone at 100 iterations, the NES with Reptile still performed $1.09 better. To measure the performance of NES with Reptile using 50% of the training data during the test stage, we compared the training of the NES algorithm using 100 iterations on 48 days to the Reptile-enhanced NES with the same number of iterations but only trained on the previous 24 days. Trading over 24 days, the average results of 10 experiments with reptile on half the data averaged $2.99 profit, while NES alone training and testing as normal averaged a profit of $1.90 over ten trials. Therefore, using NES with Reptile with half as much training data available during testing, our model performed $1.09 better and made a profit (**Figure 4**).

We can see in **Figure 4** that during trading in both tests our algorithm decided to buy stocks at the point it predicted the market value of the stocks to

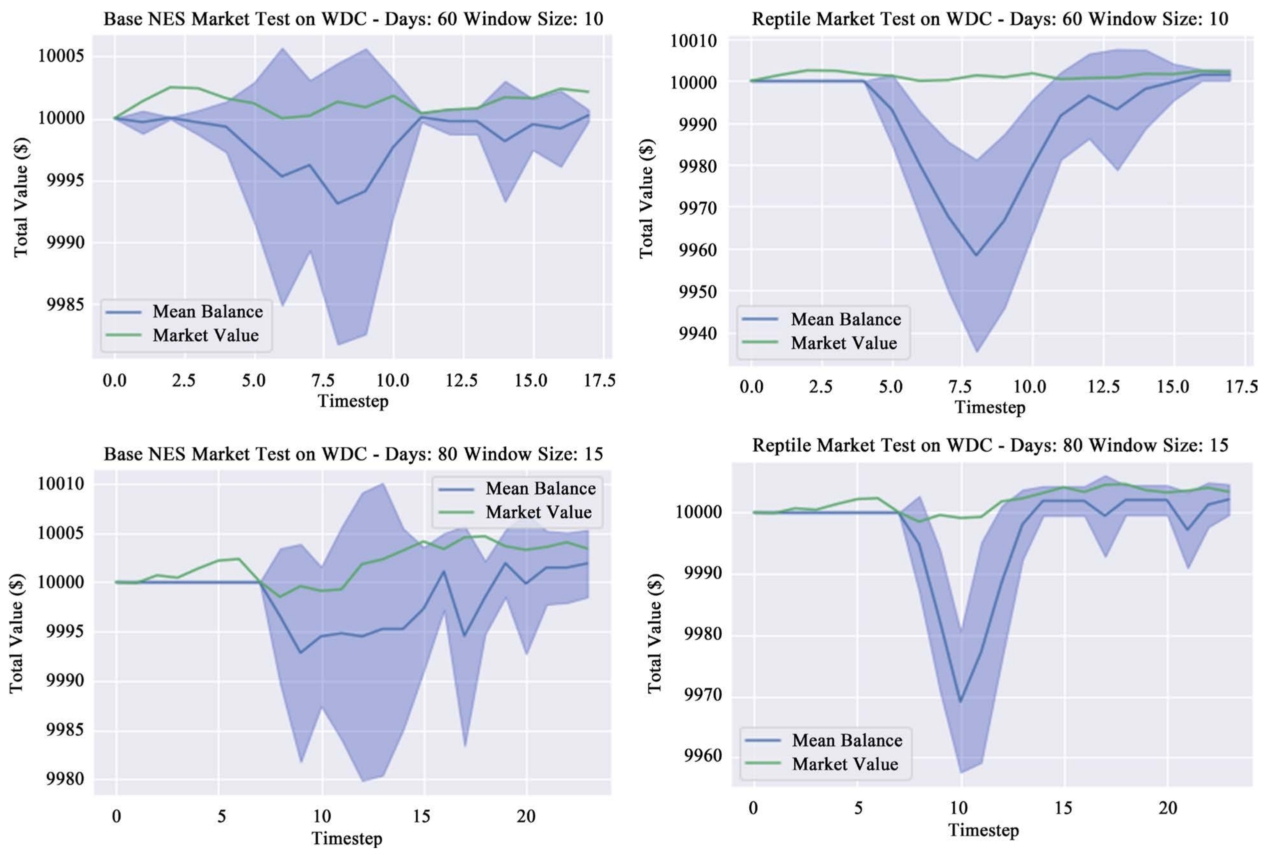


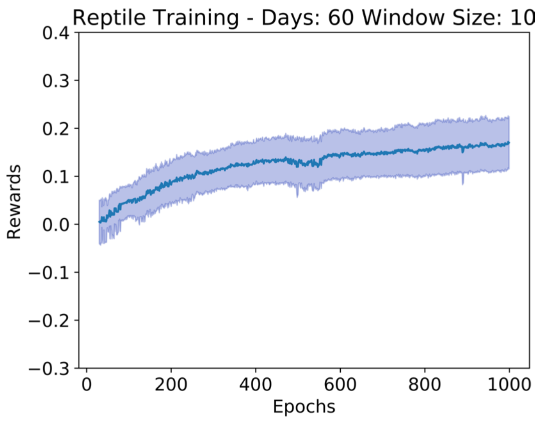
Figure 4. The mean balance and market price of WDC stock over the 18 and 24 days for both of our tests (base NES market test on the left, Reptile market test on the right). Notice how many stocks were bought the 4th day of trading for 60 days of stock and the 7th day for 80. We can see the decreasing balances which represent stock purchases after the market prices reach local maximum so that the algorithms can make a profit. In both tests, the NES algorithms with Reptile performed better ending with high balances and earning a profit.

be lower and sold once the market value began to rise. The blue line in these graphs depicts the mean balance held over 10 trading tests for each of our experiments. The green line represents the market value of the stocks over the 18 and 24 days of trading. In both experiments, our mean money held ended as more than the initial $10,000 with which we started, meaning we made profit. In our first test, our meta-learning algorithm trained on 60 days with a window size of 10 and ended with $10,001.47, meaning that it made a $1.47 profit. In our second test, our meta-learning algorithm trained on 80 days with a window size of 15 and ended with $10,002.99.

To observe the performance of our meta-training, NES with Reptile was run on 45 different stocks over 1000 iterations. The model performance consistently increased to $0.18 and $0.23 of profit for our two experiments of different window sizes. Averaging 10 trials, we plotted the mean reward and standard deviation of our training curves as seen in Figure 5.

4. Conclusion

It has been a great challenge for deep learning methods to learn from small



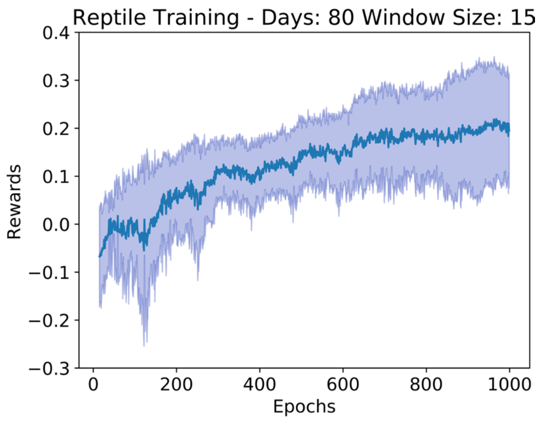


Figure 5. The meta-training curve of our experiments of different window sizes, where the models were trained successfully after 1000 epochs reaching average rewards of \$0.18 and \$0.23 over 45 same-sector stocks.

amounts of data. Meta-learning offers a promising technique to learn from previous tasks to enable efficient learning of new tasks. As far as we know, this paper is the first to study applying meta-learning to stock trading. To implement this, we used two common meta-learning algorithms, MAML and Reptile, to improve the NES algorithm to simulate an agent buying and selling on one stock in a particular sector of the stock market. When we employed NES with MAML, the model was trained more consistently than the evolutionary strategy alone, though it did not profit on the market as much as it did when it was trained only with NES. To increase our profit using MAML, we may need to, in the future, increase the number of epochs used in training the model on specific stocks. We can also explore the MAML method's potential by testing on multiple stocks.

Applying NES with Reptile yielded profit, even with a relatively small number of epochs for the meta-learning portion. We saw that the NES algorithm with Reptile made more money than without. Additionally, NES enhanced by Reptile outperformed NES alone when given fewer data points on which to train for a single stock. Although the NES with Reptile was an overall success, there is still much to be explored with this strategy. Increasing the number of epochs for the meta-portion of this algorithm as well as the number of days on which the model trains and tests may increase performance and profit.

Overall, we have seen that meta-learning can contribute to machine learning of stock trading. Certainly, our research can be extended into other areas within the stock market besides trading. However, our implementation of the NES strategy with MAML and Reptile is a good starting point for continued research in this area.

Acknowledgements

We thank the anonymous reviewers for their critical reading of our manuscript and their valuable comments and suggestions.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Hochreiter, S., Younger, A.S. and Conwell, P.R. (2001) Learning to Learn Using Gradient Descent. In: *Lecture Notes on Computer Science* 2130, *International Conference on Artificial Neural Networks* (*ICANN*-2001), Springer, Berlin, 87-94. https://doi.org/10.1007/3-540-44668-0_13

[2] Li, D., Yang, Y., Song, Y. and Hospedales, T. (2017) Learning to Generalize: Meta-Learning for domain Generalization.

[3] Finn, C., Abbeel, P. and Levine, S. (2017) Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *International Conference on Machine Learning*, Sydney, 6-11 August 2017, 1126-1135.

[4] Nichol, A., Achiam, J. and Schulman, J. (2018) On First-Order Meta-Learning Algorithms.

[5] Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J. and Schmidhuber, J. (2014) Natural Evolution Strategies. *Journal of Machine Learning Research*, **15**, 949-980.

[6] Guang, L., Xiaojie, W. and Ruifan, L. (2019) Multi-Scale RCNN Model for Financial Time-Series Classification.

[7] Hegazy, O., Soliman, O. and Salam, M. (2013) A Machine Learning Model for Stock Market Prediction. *International Journal of Computer Science and Telecommunications*, **4**, 17-23.

[8] Patel, J., Shah, S., Thakkar, P. and Kotecha, K. (2015) Predicting Stock Market Index Using Fusion of Machine Learning Techniques. *Expert Systems with Applications*, **42**, 2162-2172. https://doi.org/10.1016/j.eswa.2014.10.031

[9] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013) Playing Atari with Deep Reinforcement Learning. *NIPS Deep Learning Workshop*, Lake Tahoe, 9 December 2013.

[10] Salimans, T., Ho, J., Chen, X., Sidor, S. and Sutskever, I. (2017) Evolution Strategies as a Scalable alternative to Reinforcement Learning.

[11] Song, X., Jiang, Y., Du, Y. and Neyshabur, B. (2019) Observational Overfitting in Reinforcement Learning.

[12] Lake, B.M., Salakhutdinov, R. and Tenenbaum, J.B. (2015) Human-Level Concept Learning through Probabilistic Program Induction. *Science*, **350**, 1332-1338.

https://doi.org/10.1126/science.aab3050

[13] Antoniou, A., Edwards, H. and Storkey, A. (2018) How to Train Your MAML.

[14] Ravichandiran, S. (2018) Hands-On Meta Learning with Python. Packt Publishing, Birmingham.