Deep Learning and Fintech

Supervisor

KWOK James Tin Yau ([jamesk@cse.ust.hk](mailto:jamesk@cse.ust.hk))

Group Members

KIM, Zi Won ([zwkim@connect.ust.hk](mailto:zwkim@connect.ust.hk)) - DQN

SU, Heng ([hsuab@connect.ust.hk](mailto:hsuab@connect.ust.hk)) - LSTM

MA, Qiurui ([qmaai@connect.ust.hk](mailto:qmaai@connect.ust.hk)) - DQN

YANG, Yifeng ([yyangbf@connect.ust.hk](mailto:yyangbf@connect.ust.hk)) - Convolutional Autoencoder

[**Abstract**](#_61id1ociyvu6) **1**

[**Deep Q Networks (DQN)**](#_fdj814ctotvu) **2**

[Background](#_o2txaio9qfwq) 2

[Experiment](#_yxf53cmwq6f2) 2

[Results](#_2bruejtpndxa) 3

[Future work](#_ixukt2t5s2mo) 4

[**Long Short-Term Memory (LSTM)**](#_drd8pinnt87h) **4**

[Background](#_jisfkg5nmkvm) 4

[Experiment](#_9cvuu2uou2b9) 4

[Results](#_jt91am22ok4n) 5

[**Convolutional Autoencoder**](#_8g7hzmza2chp) **7**

[Introduction](#_vibm69hoh93y) 7

# Abstract

In recent years, deep learning has proven itself as a powerful technique for various applications. These applications range from object classification from images to forecasting events to natural language processing. In this research project, we try to apply machine learning and deep learning techniques for predicting stock prices from historical data only. Specifically, we utilize deep Q networks, a form of reinforcement learning, long short-term memory recurrent neural networks, a combined approach with both LSTM and RNNs, and convolutional neural networks to tackle the problem of predicting stock prices. Without any professional experience, amateur investors and analysts may potentially use this information to make decisions when performing trades.

# Deep Q Networks (DQN)

## Background

Deep Q Network (DQN) is a technique developed by Google’s DeepMind in 2015, which combined Deep Neural Networks with Reinforcement Learning (Q-Learning) for the first time. DeepMind has demonstrated that with DQN, it can train machines to adapt its behavior, or policy, continuously without any human intervention by learning itself. As a result, an agent that has been trained with DQN on Atari games have achieved higher performance marks than humans.

With this new combination of Deep Learning and Reinforcement Learning, it brings up the question, “Can a computer agent trade better than humans in the stock market?”. Many attempts have been made with deep reinforcement learning: Moody et al. first explored the single asset with discrete position size, and then continuous quantities of multiple assets with a new differential sharp ratio [1]; Zheng et al. explores portfolio optimization with DQN and modern portfolio theory [2]; David Wu, an analyst in Merrill Lynch discovers that replacing the vanilla Q-network with LSTM boosts the robustness and feasibility of the system [3].

## Experiment

Data preprocessing:

The raw data is in the form of n\*20, with n as trading days in 30 years from 1987 to 2017, and 20 as the dimensions of indicators including ‘close, high, low, average, MACD, …’. Two kinds of data formats are explored: data of the recent 10 days are stacked together as one state (assumption: the state of the next date fully depend on the past 10 days before it, MDP); data of one single day is used as state (POMDP). In the experiment, the stacked frames does not yield noticeable better result compared with using one single day data, possibly because indicators like ‘average, MACD, etc.’ already incorporate the moving trend of the price, and therefore is a MDP already.

Normalisation was first performed on the 20 year data to guarantee that the training set covers the similar states as the test sets, the growing time value of money would otherwise push up the price and other indicators throughout the 20 years. The data is then wrapped in a game environment, which mocks the set up of that in OpenAI gym: an episode of predetermined number of steps (100, 200, 400, 1000) forms one episode, namely one game. The game restarts and shifts to next period when the end of the episode is reached. One concern for breaking the periods down is that too long a period may induce a pure buy-and-hold strategy instead of a frequent trading strategy.

Setup:

There are three actions available for the system to choose from, sell (asset = -100), hold, buy (asset = 100), only when there are changes in the asset will a trade be executed. Reward function is set to be the incremental gain of the asset and cash on hand. In the last action in a game, the asset is traded back to cash again. The total gain of an episode is calculated at the end of each episode, representing the performance of the agent.

Explore:

The Q-network is set to be a simple 3-layer feedforward network. The recurrent reinforcement learning, changing the vanilla Q-network to LSTM was also explored, but little perceivable boost of performances follows. Exploration and exploitation ratio declines linearly with number of epochs. To break the correlation between continuous steps, a replay buffer of size 10,000 was employed to be randomly sampled from. There is a huge performance boost after the memory is employed. A further comparison is made between using double network against single network, the former proves to yield better result, presumably due to the fact it converges better.

## Results

Performance of the agent is measured with profits made throughout the testing data with the length of an episode. The performance is measured against the standard buy-and-hold strategy. In the most cases, the agent outperform the passive buy-and-hold but do not make fully optimized performances.

## 

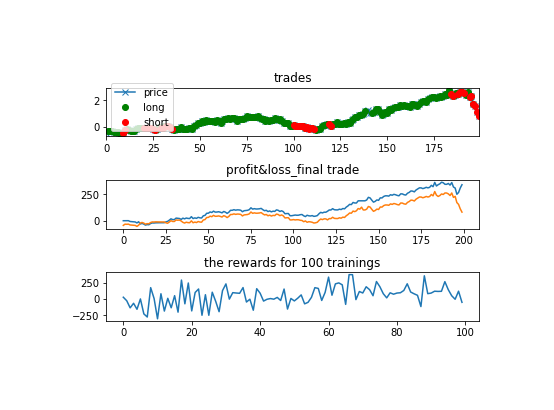
## 

## 

## 

## 

Double Network with episode steps of 200. Upper Left: Red: short, position = -100; green: long, position = 100; Middle Left: the profit for the episode illustrated in the left; Middle Down, Right: reward fluctuation for each episode during the training processes.

Single network agent fails even more miserably to make the right choices.

## Future work

Current network structure is not proven to be optimal. Explore on the deeper networks and dropout, batch-normalization layers.

Incremental gains as reward may not fully justify the risk involved in the return. Sharp ratio as reward function may better fitted in the real trading world. Propose to follow the structure brought up by Moody et al.

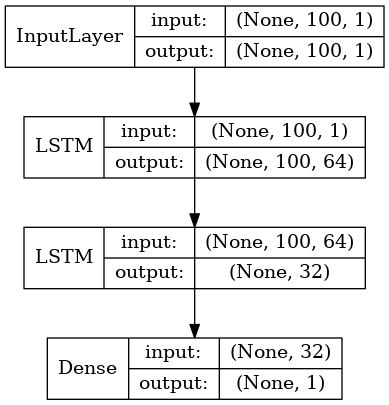
The current action space is discrete: buy, sell, hold only. It is more favorable to have an agent that have continuous action space. Propose to use DDPG(deep deterministic policy gradient) to explore.

# Long Short-Term Memory (LSTM)

## Background

Recurrent neural networks, or RNNs, are a specific type of neural network that specialize in dealing with temporal data. It’s structure is similar to a normal feedforward network, with the exception of an additional “hidden state,”, which allows for information to be preserved analogous to human memory. Long short-term memory networks (LSTMs) are a variation of the traditional RNN. LSTMs have proven to be more powerful than vanilla RNNs for generalizing long term dependencies.

## Experiment

In this section of the project, I utilized a network with two LSTM layers and a fully connected layer with 1 output to predict the next stock closing price. A diagram of the network is provided. This model was decided upon via trial and error, and was based on existing architectures found in the literature. I experimented with adding dropout layers to try and prevent overfitting, but the results showed no significant improvement. Therefore, I left them out in the final version of the model. I separately trained this model on four separate datasets (CL\_Monthly, CL\_Weekly, CL\_Daily, and CL\_60min) to compare and contrast the performance of the network on different amounts of data. For CL\_Monthly, the first 60% of the data was used for training, and the last 40% for testing. In the other 3 datasets, the first 80% of the data was used for training, and the last 20% for testing. The data was first pre-processed by scaling it to between 0 and 1. It was found that this normalization allowed the network to perform better, as it had difficulty generalizing values it had never seen before without this normalization. The training data was generated by creating a “sliding window” of size 100, and using the first 99 elements as input with the 100th element as the expected output. The training was done in a Google Colaboratory environment.

## Results

The root mean square error was used to analyze the results of each experiment. As expected, the network with the largest amount of training data performed the best on the test set, and had the lowest RMSE. For the monthly data, it seems that due to lack of data, the network was unable to generalize any sort of meaningful pattern from the data, as evident by its poor performance in the test data. The weekly model seemed to have picked up on the trend, but was not as precise as the daily and 60min model. Below is a summary and comparison of the 4 models, and plots of their performance on the testing set.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CL\_Monthly | CL\_Weekly | CL\_Daily | CL\_60min |
| # train, test samples | 28, 20 | 572, 144 | 3202, 801 | 90719, 22680 |
| Epochs | 10 | 1 | 1 | 1 |
| Training time | 0:00:13 | 0:01:15 | 0:05:14 | 3:13:37 |
| RMSE | 0.0543 | 0.0269 | 0.0106 | 0.00268 |

|  |  |
| --- | --- |
| Original dataset | Scaled dataset |
|  |  |
| CL\_Monthly | CL\_Weekly |
|  |  |
| CL\_Daily | CL\_60min |
|  |  |

# Convolutional Autoencoder

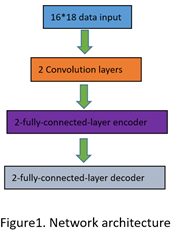
In this session, we will discuss how to use convolutional autoencoder to extract useful information from stock sequences and then compare the similarity between sequences.

## **Introduction**

Autoencoder is a common technique used to learn useful representation of data. Due to its low-dimension latent space, it is also frequently used in dimensionality reduction. Stock market is a fast-paced market and many sophisticated models have been invented to detect evanescent profitable chances. In order to catch these chances as fast as possible, we seek to reduce data dimension by autoencoder. Moreover, to further make use of the price trend, which is similar with local pattern detection in picture recognition, we also employ convolutional neural network to detect local pattern.

**Method**

We use daily data with 18 attributes and 4104 samples. We combine the consecutive 16 days to produce a 16\*18 matrix, which is regarded as a sample in training the network. In such way, we are able to produce 4087 samples. We construct a network consisting of two parts. The first part is a 2-layer convolution layers with max-pooling, the filters shape of whose are (4,4,1,20), (4,4,20,40), respectively. These numbers mean filter width, filter height, number of input channels, number of output channels. Max-pooling with stride=2 is performed after each convolution step. The output of convolutional network is input into autoencoder, which consists of encoder part and decoder part. Both encoder and decoder part consist of two fully-connected layers, whose dimension are 150, 40, respectively. After training the network, we input all historical samples into the model to obtain the output in latent space. Then, we also obtain the latent representation of a test sample. We search for the closest sample sequence by Euclidean distance. The following figure shows the architecture of the network(Figure 1).



**Results and Discussion**

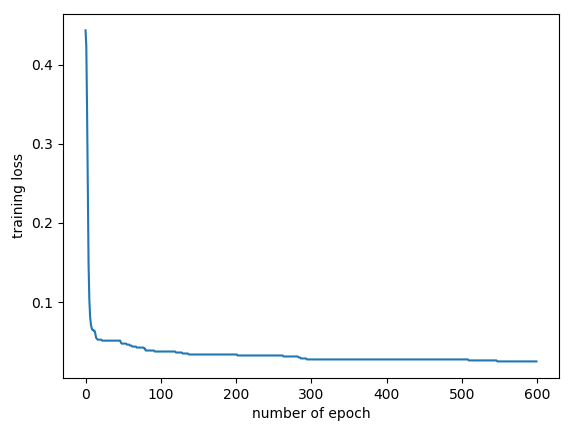
The following graph shows the training loss against number of epochs(Figure 2).

Figure2. Training loss

We tried different shapes of filters, dimension of each layer. Even though the training loss reduces as the number of epochs increases, we are not successful in getting meaningful results. We find that the output of convolutional layers are all zero. So we cannot differentiate input samples.There are several reasons. First, the number of samples in daily data is quite small to train a network. Although we have hourly data, it is quite hard to train the model without GPU. Second, the network may be comparatively too deep.

There are some further possible improvements. For example, we can use hourly data as its sample size is much larger than daily data. In addition, we may also add some deconvolutional layers at the end of the decoder.