Midterm Report

Pavee Phongsopa

December 2019

1 A-Term

1.1 Background

Following the successes of machine learning in different industries in the recent years, the goal of our project is to apply machine learning to area in finance specifically focusing on deep learning in option trading. We started off with an example of neural network on linear function which consists of 2 layers of 2 neurons that employed ReLU function to help approximate.
Sample pseudo code neural network in Python:

$$Output = inSequence(NeuralNetwork(input, ofneurons)ReLU()NeuralNetwork(ofneurons, ofneurons)N \tag{1}$$

1.2 Non-Linear approximation with ReLU

$$f(x) = max(0, x) \tag{2}$$

In order to approximate something significantly more completed like a graph of option price we needed to find out if the neural network setup can stay accurate. To do so, I increased the degree of the polynomial of the testing function. The result was inaccurate due to the insufficient number of neurons and layers, so I started increasing number of both. One by one the predicted graphs got closer and closer to the test graph while the loss slowly decreased. From what I find, number of neurons plays a larger role than number of layers in determining the accuracy of the predicted graph. What I also found interesting was that different degree polynomials reacted differently to the same neural network setup. Lastly, just to see how close I can get to 0 loss, I left the computer running for a few hours with over 10 000 neurons. The result was accurate but not very practical due to the amount of time that it takes to calculate.

1.3 Non-Linear approximation with Sigmoid

$$s(x) = \frac{1}{1 - e^(-x)} = \frac{e^x}{e^x + 1} \tag{3}$$

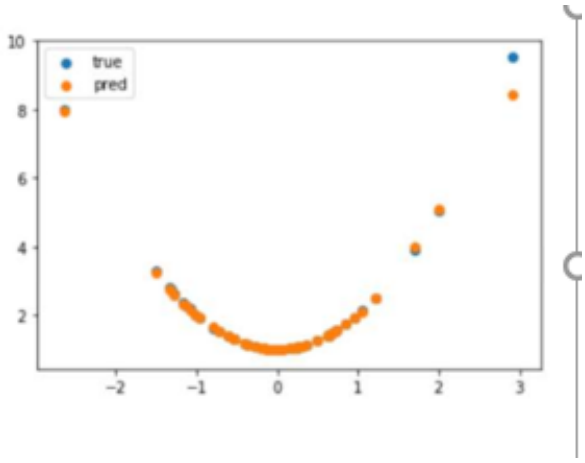

Figure 1: Approximated Graph vs Real Graph with ReLU()

Next, I try the same neural network approximation but with Sigmoid function instead of ReLU. The result obtained showed the same trend in a sense that number of neurons greatly affect the loss function and that different degree polynomials worked better with different setups. However, one thing that I notice is different is that with Sigmoid function, the code is slightly faster but left with more loss at the same number of neurons and layers when compared to ReLU. With Sigmoid, graph tends to diverge on both ends which probably contribute to the increased loss in loss function.

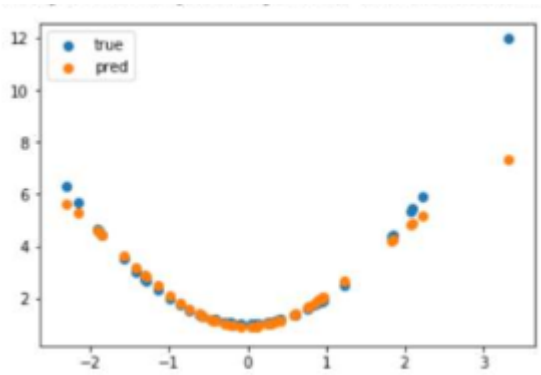

Figure 2: Approximated Graph vs Real Graph with Sigmoid()

1.4 Neural Network Setup Optmizer

From the two non-linear approximation sessions, it was clear to me that the number of neurons and layers played significant roles in the convergent to zero of the loss function as well as the speed at which it converges. Additionally, different graphs have different optimal setups. Thus, I thought it would be a good idea to include a setup optimizer even if it might not be needed in our project at that moment. To use this optimizer, you will be required to put in the maximum number of neurons and layers that you want to test, and the program will run every combination starting from 2 neurons and 2 layers. The code will then return 2 setup statistics. One will be the setup for lowest loss and the other will be the setup for the fastest code under acceptable loss (loss ¡ 0.01).

2 B-Term

2.1 CRR Model

Moving on to the step, our team split existing financial models that we wanted to test the neural network with into 3 parts. The CRR model code, which I was responsible for, was based off the instruction and pseudo code I found of wiki page. With the help of the reference materials, I was able to code CRR European Option and American Option in python.
Bionomial Tree Calculation
Given: steps(n), time(T), stockprice(S), strikeprice(k), interest(r), volatility(σ), *dividend*(q)

$$t = T/n \tag{4}$$

$$U = e^{(\sigma\sqrt{t})} \tag{5}$$

$$D = \frac{1}{U} \tag{6}$$

$$P_1 = \frac{Ue^{(-qt)} - e^{(-rt)}}{U^2 - 1} \tag{7}$$

$$P_2 = e^{-rt} - P_1 \tag{8}$$

2.2 CRR Model with Neural Network

The first iteration of CRR model code (November 18th) had one fault in it which was the efficiency for each of the testing and adjusting epoch. This drawback persisted because I was unable to convert the data structure of the tensor table which left me with the limitation of being able to only test 1 value per try. Fortunately, it seems to work at first glance.

Later that week I was able to find the solution and the code was back to being able to test 100 values per epoch with a total number of 10 000 epochs. The result looked promising and it looked like neural network is performing well in approximating the American option function as you can see with the loss function.

2.3 Symposium Poster

Symposium posters created with the help of the information on Vital's and Nick's GitHub pages.

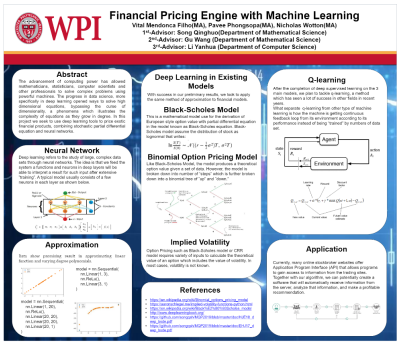

Figure 3: Symposium Poster

3 C-Term

Depending on how much we manage to accomplish in this B-term and maybe during the break, we planned to tackle q-learning which use the concept of real time reward as per our professors' suggestion. We also hope to finish all our writing section of our MQP within this period.

3.1 Q-Learning Sample

Following this Q Value equation.

$$NewQ(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma maxQ'(s',a') - Q(s,a)] \tag{9}$$

Below is one of the simplest q-learning examples that I could find and tried. I planned to study and apply the same idea and structure to our code if circumstances allow. The goal of this code is to allow the machine to learn from its moves to find and memorize the quickest route to a vertex containing number 7 when given a graph. For example, Given this graph

The machine was able to learns and memorizes the most efficient route as you can see on this reward vs time graph

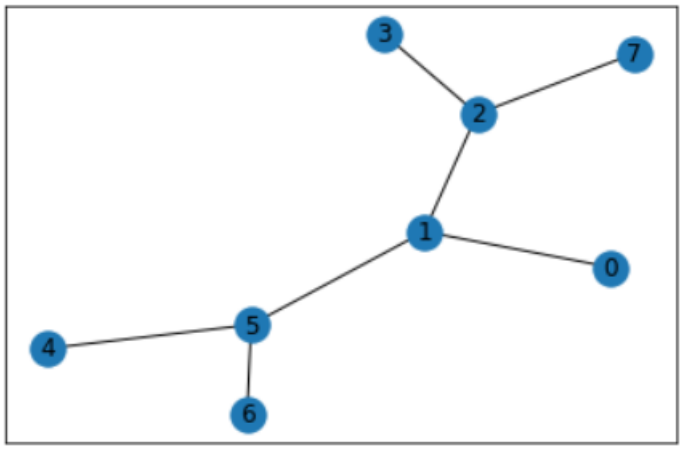

Figure 4: Sample Edge/Vertex Graph

```
Most efficient path:
[0, 1, 2, 7]
```

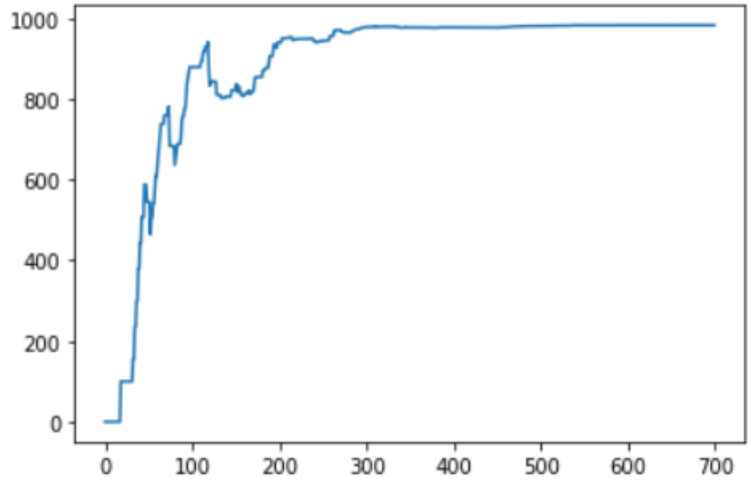

Figure 5: Reward vs Attempt Graph

3.2 Interactive Broker API

Small order API code that might be useful later in C-term if we ever decide to mess around with automatic trading.

References

https://github.com/songqsh/MQP2019/blob/master/Pavee/test
https://github.com/songqsh/MQP2019/blob/master/Pavee/Test
https://github.com/songqsh/MQP2019/blob/master/Pavee/Optimizer.ipynb
https://en.wikipedia.org/wiki/Binomial$_{o}ptions_{p}ricing_{m}odel$
$https://github.com/songqsh/MQP2019/blob/master/Pavee/CRR_{M}odel.ipynb$
$https://github.com/songqsh/MQP2019/blob/master/Pavee/Deep_{l}earning_{f}or_{C}RR.ipynb$
$https://github.com/songqsh/MQP2019/blob/master/Pavee/Complete_{C}RR_{N}eural_{N}etwork.ipynb$
$https://github.com/songqsh/MQP2019/blob/master/Pavee/symposium.pdf$
$https://github.com/songqsh/MQP2019/blob/master/Pavee/qlearning_{s}ample.ipynb$
$https://github.com/songqsh/MQP2019/blob/master/Pavee/InteractiveBroker_{A}PI.ipynb$