Capstone Project **Machine Learning Engineer Nanodegree**	Cristóbal Garib July 20th, 2016

Definition

Project Overview

In this project I am going to teach a program to trade stocks. To do this I will start with a dataset containing all the daily prices of the S&P 500 stocks. With this information I will choose some random range of time and I will tell my program to buy, sell or hold stocks during this time. At the end I will contrast the return of the traded portfolio with the SPY index (ETF of the S&P 500) and see how the program did. With this result I will train the program to not repeat its error and to repeat what he did right. Then I will repeat this process several times in different time ranges. And finally I will test the program on the last 3 months of data to see how I did.

To train the program I will take the same approach one would take to train a dog; if the program does some right (get a good portfolio return) I will give him reward and if the program does something wrong (bad portfolio return) I will punish it. This approach is called reinforced learning.

The novel approach of this project is that I will not teach the program what to do in each occasion because the number of possible combinations of stock prices is infinite. Instead I am going to give the program a brain, or a network of computer simulated neurons, and I am going to train the brain so it can infer what to do in each different possible combination of prices.

Problem Statement

In my last project in this Nanodegree I had to train a cab to learn its optimal policy for driving using Q-Learning. My conclusion in the project was:

"The main problem with this model is that it is too hard to see all the possible cases, so there are many states that just don't have a Q value . This is why I thought that maybe a good solution would be to learn these relations with another machine learning model. We could replace Q with a regression model, thus instead of calculating Q we would predict it. The inputs would be the different variables of the state and the action, and the output would be the value of Q

With this approach not only the current state/action value of Q would be updated, but also the values of other Qs that the model believes to be related to.

This looks like a great idea for my capstone project ☺"

After doing the course on Google´s course on Deep Learning I got interested in the subject so I read some papers of Google Deep mind and I came across Deep Q Learning and I realized it uses the same conclusion I arrived in my last project. They were using a Deep Convolutional Network to learn Q. I then also did the course on Machine Learning for Trading (because my last job was in Asset Management), which at the end talks about how to use Q learning for stock trading. I put all of this together and decided to do this project.

So what I am going to do is that I will use Deep Q learning to learn the optimal policy to trade stocks. I will train the model using several trade runs where I am going to simulate that the program is buying and selling stocks, and I am going to use the return on this trades as rewards. Instead of traditional Q Learning I am going to train a Deep Neural Network to predict the values of Q.

I am going to use this approach because the number of possible cases in combinations of prices of stocks is almost infinite and normal Q learning could never find an optimal policy.

The dataset is going to be all the stocks in the S&P 500. The goal is to have a model that haves a portfolio return well above the return of the SPY, the ETF of the S&P 500 stocks.

Metrics

My main metric will be the total portfolio return. Then I will contrast this number against the total return during the same period of time of the SPY. This will provide me insight in whether my program is really outperforming the market.

My secondary metric will be **% of good trades**. To calculate this metric, I will classify see in which stocks that I bought and then sold I made a positive return and I will call the **hits**, the I will find the stocks where I had negative returns and I will call the **miss**. Thus **% of good trades = hits/(hits +miss)**. This metric would be useful to see if the agent is consistently choosing good stocks.

My final metric will be a **random trader**. Here I will do 100 trade runs over the test data with that agent that only uses random actions, then I am going to average the portfolio returns and the variance. This metric is useful because is going to show if my program is really learning or if it just randomly guessing. My portfolio return should be above the **mean portfolio return plus the variance of the random agent**.

Analysis

Data Exploration

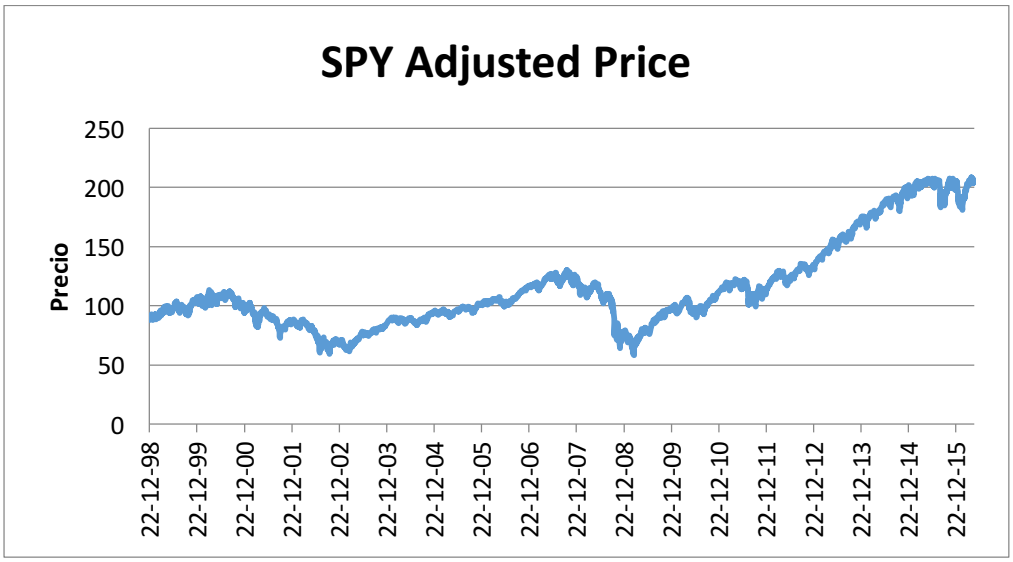
The data consists of the daily prices of 503 stocks that compose the S&P 500. This data was taken for Yahoo finance. Each stock has the following information (we are going to use GOOGL stock as example):

Date	Open	High	Low	Close	Volume	Adj Close
05-09-16	726,70	734,29	723,50	729,13	1.898.900	729,13
05-06-16	712,20	72,60	711,95	725,18	1.982.500	725,18
05-05-16	715,00	717,55	709,45	714,71	1.479.800	714,71

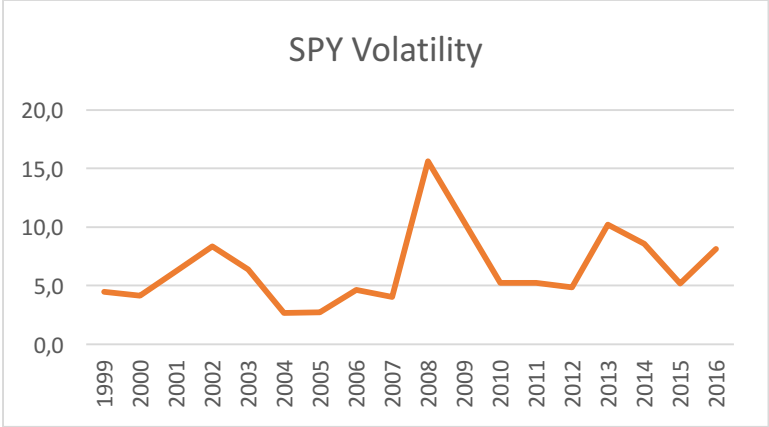
The **Date** is the date where that data was recorded, as we can see is sorted in descendent order. **Open** is the price that the stock had when the market opened. **High** is the maximum value of the stock during the day, and **Low** is the lowest value. **Close** is the price of the stock at the end of the day. **Volume** is the amount of money used to buy or sell the stock during the day. Finally, **Adj Close** is the price at the close of the market but is adjusted to take in account dividends paid and stock splits during the life of the stock. This last feature is very significant because it gives as the true picture of how the stock has performed during its life.

Most of the data spans from 1998 to 05-09-2016 (which is the last date that Yahoo finance lets you scrub their data for free). Some stocks start later the last one starting on 03-04-2010. Also is important to notice that many dates are missing from the dataset, this happens when the stock was not traded that day, either because the market was closed that day or the stock was not traded. The average number of day for each stock is 4.126. And the total number of rows of information collected is 2.075.378.

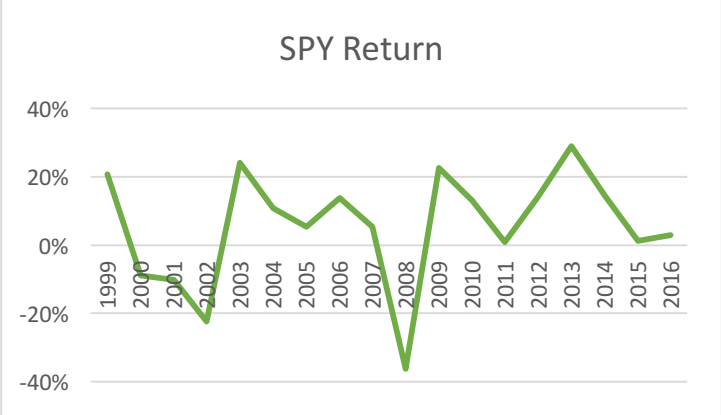
In order to get meaningful statistical data, we can use the SPY index because this index is the average of all stocks in the dataset. Here is a graph of the SPY:



Here we can see the internet bubble burst of 2002, the crisis of 2008 and high volatility of the markets this past years. The volatility of the SPY is showed by year in this graph:



Again we can see the volatility in 2002, 2008 and these past years. Finally, for the SPY, here we can see yearly returns of the SPY:



Algorithms and Techniques

The algorithm is heavily based in the one used in famous paper "Playing Atari with Deep Reinforcement Learning" by Google´s Deep Mind. It basically feeds a deep neural network with a preprocessed state φ_t which in turns outputs the predicted value of $Q(s_t,a_t)$ In their case they use a Convolutional Neural Network because their input was pixels, in our case we will use a normal Neural Network with hidden layers and a lineal output layer.

In order to train this DNN the loss function we are going to use is $(y_j - Q(\varphi_j , a_j))^2$ where $Q(\varphi_j , a_j)$ is our predicted value of Q and y_j is our expected value of Q using the classic Q Learning formula $y_j = r_j + \gamma \max_a Q(\varphi_{j+1}, a)$, γ been the future reward discount.

In their paper, Deep Mind introduces "experience replay" or all the different transitions $(\varphi_t, a_t, r_t, \varphi_{t+1})$ are stored in in a set D, some time passes a random minibatch is subsampled

from D to train the model, breaking similarities between each transition of states. In our case we take a not so different approach, we create M different trade runs where we simulate that our agent is trading stocks between a random number of consecutive periods. I take this approach because the state φ_j includes the current portfolio, and in other to simulate the different combinations of portfolio I have to simulate a trade run. Still do it in random periods of times with random length allows the model to break the similarities between state transitions.

Finally we use ε-greedy for exploration just like Deep Mind uses in their algorithm, and we make the ε decay over time.

Algorithm Deep Q Trader

Initialize preprocessed states $\varphi_t = \varphi(s_t,\varnothing)\ \forall\ s_t$
Initialize empty portfolio P to max capacity N
Initialize action-value function Q with random weights
for trade_run = 1, M **do**
  Select random range (T_i, T_f)
  Initialize set D to store samples
  for t = T_i, T_f **do**
    -With probability ε select a random action a_t, otherwise select a_t = maxarg Q* ($\varphi(s_t,P)$, a)
    -Execute action a_t buying, selling or holding designated stock and updating portfolio P.
    -Calculate portfolio return for period t and store as reward r_t
    -Update preprocess $\varphi_{t+1} = \varphi(s_{t+1}, P)$
    -Store transition (φ_t, a_t, r_t, φ_{t+1}) in D
    -Update decaying ε
  Set $y_j = r_j + \gamma$ maxarg Q(φ_{j+1}, a) for each sample in D
  Perform a gradient descent step on $(y_j - Q(\varphi_j , a_j))^2$

Benchmark

As mentioned before I am going to use the total return during the same period of time of the SPY. The SPY is the ETF the groups the return of all the S&P 500 stocks, which is the stocks we are using to trade in this model. In essence is the average return of the market. So by contrasting our results against the SPY we can see if we are actually outperforming the market or not.

Methodology

Data Preprocessing

The input selection and preprocessing is heavily based in Udacity´s "Machine Learning for Trading" course.

When exploring the data, we noticed that not all the stocks start at the same date. Also some of the stocks don´t have transaction for some of the dates and consequently don´t have a price. Moreover, not all of dates have transactions because the market where closed.

The best way to deal with this problem is to use the dates associated with SPY, because this ETF is one of the most traded instruments in the world so every date where it has a price is a date where the market was open.

The underlying dynamics of the markets change over time. What defined the prices of stocks today is very different than what defined then 10 years ago because the companies are different. This is why we are only going to use prices from 2010 and on. This is a good starting prices because it eliminates the noise of the crisis of 2008 but it still is a lot of data to train our model with, exactly 803.794 data points.

So, we first extract the dates from SPY since 2010. Then we get the prices of each stocks for each of these dates. If there is no price for a given date, we use the last known price of the stock to avoid including in the dataset future knowledge of the prices.

The different scales of the prices of each stock makes them unsuitable to use them as inputs for a DNN. In order to scale them "Machine Learning for Trading" suggest dividing the pricing with the windowed mean. This measure is usually used in technical stock analysis. The size of the window is one of the parameters we are going to use on the model. We are going to subtract the windowed mean by one to center the values to 0.

The other two features that Machine Learning for Trading suggest using are Bollinger's bands and a feature to indicate if the stock is present in the portfolio.

Bollinger's bands are two bands that go along the windowed mean. The lower Bollinger band is the windowed mean minus two times the windowed variance, and the upper band is the windowed mend plus two times the variance. In order to reduce the dimensionality of the input we are going to give a value of 1 if the stock price is over the upper Bollinger band, a values of -1 if the stock prices is under the lower Bollinger band and a value of 0 if it is within both bands.

We are going to represent a stock being present in the portfolio with a value of 1 and 0 if it´s not.

As we can see all the features will have approximately values between -1 and 1 and will be centered in 0. Perfect for a DNN.

Implementation

The states are going to be represented by a list containing the Adj_Price/Windowed_Mean, the Bollinger´s bands and a list that represents witch stocks we have in our portfolio that we are going to call "holds". I also want to include in the states the information for the las n days, n is going to be a parameter in the model. So our state will be composed of n Adj_Price/Windowed_Mean, n Bollinger´s bands and a list that says if the stock is in the portfolio or not.

The state is passed as an input for the DNN. The DNN is created using Tensorflow. I built two different DNN to find the best result:

DNN 1

	Input	Hidden 1	Output
Size		RELU	LINEAL
Activation	STATE_SIZE	(STATE_SIZE+NUM_ACTIONS)/2	NUM_ACTIONS

DNN 2

	Input	Hidden 1	Hidden 2	Output
Size		RELU	RELU	LINEAL
Activation	STATE_SIZE	STATE_SIZE-(STATE_SIZE-NUM_ACTIONS)/3	STATE_SIZE-2*(STATE_SIZE-NUM_ACTIONS)/3	NUM_ACTIONS

The out of the DNN is our predicted $Q(\varphi_j , a_j)$. There are three possible actions buy, hold all sell for each stock so there is a total of 503*3 = 1.509 different actions, or outputs in our DNN.

To train the model simulating M training runs. Each trade run will start in a random point in time and it will last a random number of days between 30 and 120. Initially the portfolio starts empty and "holds" will start with 0. The value of ε starts in 1. Then we start out simulation.

Each day an action is chose. Depending on the value of ε we either choose a random valid action or we look for the valid action that has the largest predicted value of Q. Choosing a valid action is important in order to not waste time learning Q values for actions that do nothing like selling or holding a stock that we don´t have in the portfolio, buying a stock we already have or buying a stock when the portfolio is full. Here is the code for choosing an action:

```
def _choose_next_action(self,state):
    new_action = np.zeros([self.ACTIONS_COUNT])
    self.action_index=0
    if random.random() <= self._probability_of_random_action:
```

```
        # choose an action randomly
            self.action_index = self._get_random_valid_action(state)

    else:
        # choose an action given our last state
        readout_t = self._session.run(self._output_layer,
feed_dict={self._input_layer: [state]})[0]
        ordered_index= np.argsort(readout_t)
        for i in range(len(ordered_index)):
            self.action_index=ordered_index[len(ordered_index)-i-1]
            if self._check_valid_action(state, self.action_index):
                break

    new_action[self.action_index] = 1

    return new_action
```

After choosing the action the portfolio is updated. Each action in the portfolio has the same weight and the portfolio can hold no more the 20 stocks. Also only one action is performed for each day.

With the updated portfolio we can calculate the next state by getting the new prices and calculating the new holds with the updated portfolio. Also we decay the value of ε, from 1 to 0.1, we do this so that the value of ε is 0.1 in the last 10 trades.

Finally, we store the transition $(\varphi_t, a_t, r_t, \varphi_{t+1})$ in D. And when we end our trade run we pick all the transitions in D and we train our DNN. We use stochastic gradient decent to train the DNN.

The biggest complication of this implementation is that the states are very big so it took a lot time to calculate the next state given an action. To solve this in my implementation I precalculate all the possible states, this gave me a significant boost in speed.

Refinement

After checking that everything worked we are going to see what parameters optimize the model.

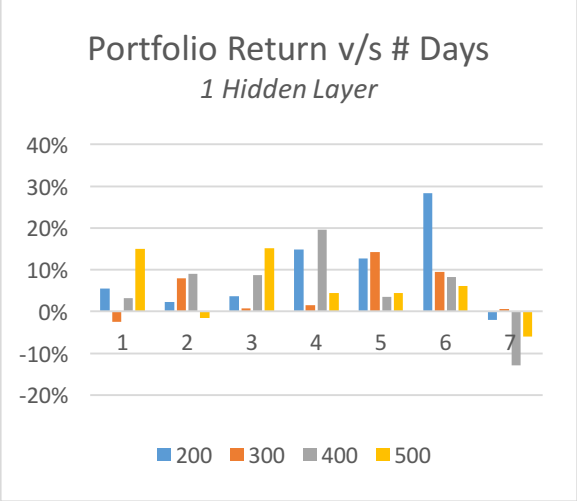
First we are going look for the best combination between number of hidden layers, number of days in state and the number of trade runs as this are probably the most important parameters in the model. We are going to choose the four best models according to the portfolio return we get from doing a test run through 01/01/2016 to 05/09/2016.

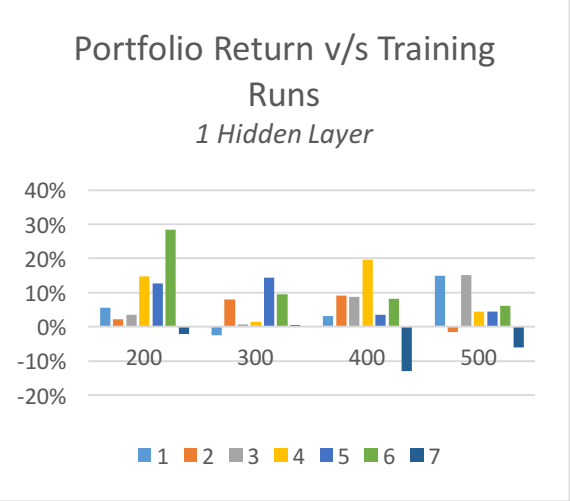
Then we are going to iterate though the learning rate and the window size for the Bollinger´s Bands and the windowed average. Here we are going to use another test set from 08/27/215 to 12/31/2015 so we don´t indirectly overfit the model by choosing the parameters.

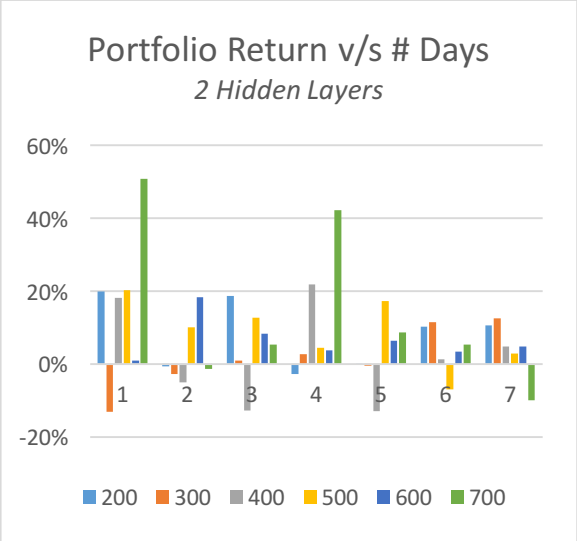
Finally, we are going to choose the best 4 models and we are going to do a Roll Forward Cross Validation by dividing the dataset in 18 folds. And we are going to divide the number of trade runs in 18 so at the end we would train the agent its specific optimal number of trade runs. The final model will be the one with the highest average portfolio return of the last 10 folds, because the first ones will have too few trade runs to be train with.

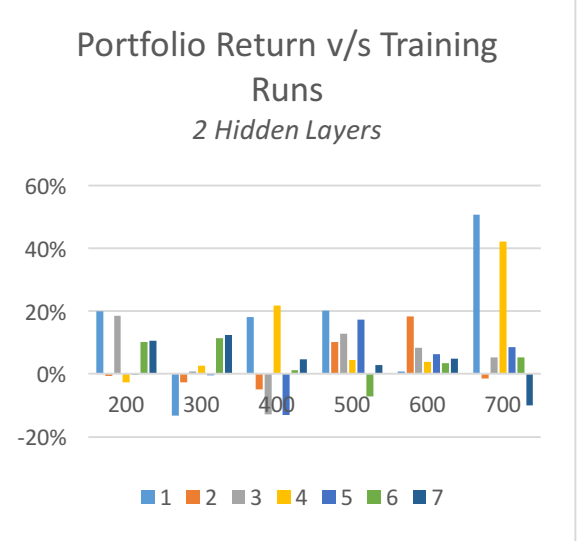
We are going to select the model using this approach because each complete training session, depending on the number of features and number of trade runs, lasts between 20 to 50 minutes. So I have to optimize the number of runs I will be doing.

The results for the first round of iterations are:









For this dates the SPY return was 2,78% so every result over that value is a success.

Going through the different strategies taking by the agent I can see that remarkably it chooses to either a buy and hold trader or a timing trader. Two popular strategies when trading stocks.

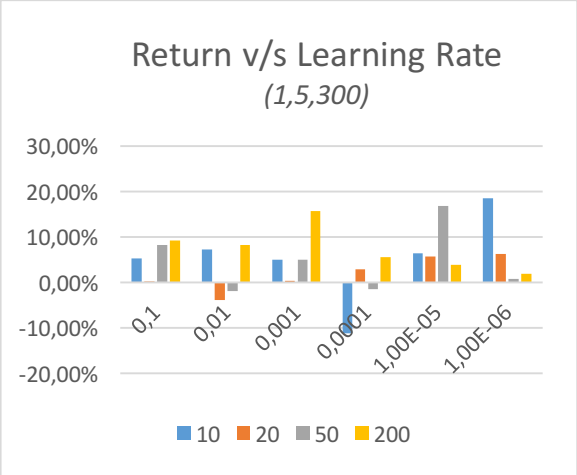
The buy and hold strategy, is a long term strategy. It picks just a couple of stocks in the beginning of a trade run and it keeps them till the end. The average number of stocks that a buy and holder agent has in its portfolio in these runs is 4,83. And in average is having an 8,64% return over portfolio. The one with best results has a 50,77% return and is doing so remarkably well because it chooses two stocks that were in their minimum price and then they have a huge rally over the year. The fact that the model predicted this is remarkable.

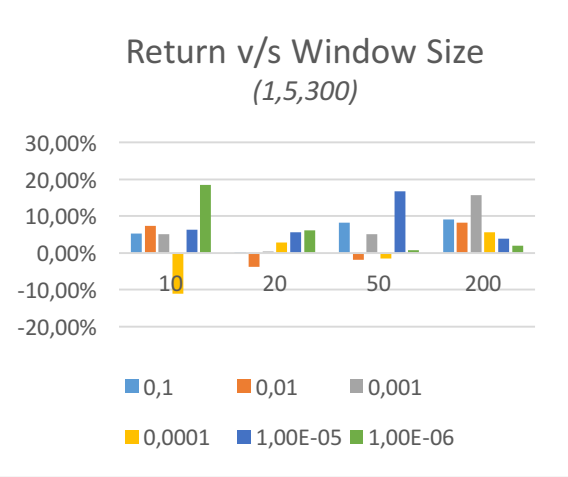
The timing strategy is based on continuously moving the portfolio, by trying to buy and sell in the right moment. The average number of stocks these agents have in the total runs is 45,6. And in average is having a 5,04% return over portfolio. This strategy usually has very good returns but the average is lower because sometimes is that really poorly, the worst value is -13,18%.

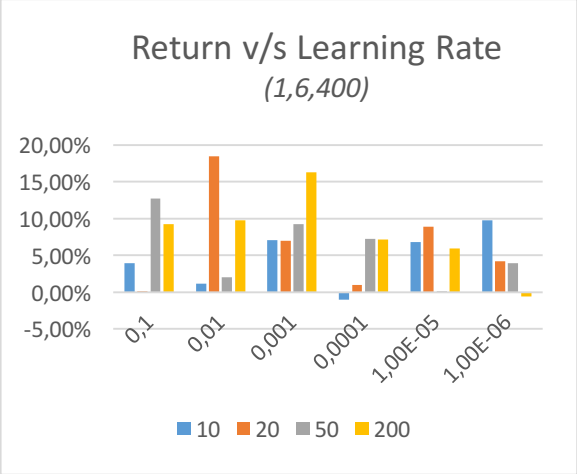
With the one hidden layer chooses the buy and hold strategy 42% of the times and the two hidden layer agents choose this strategy 56% of the term.

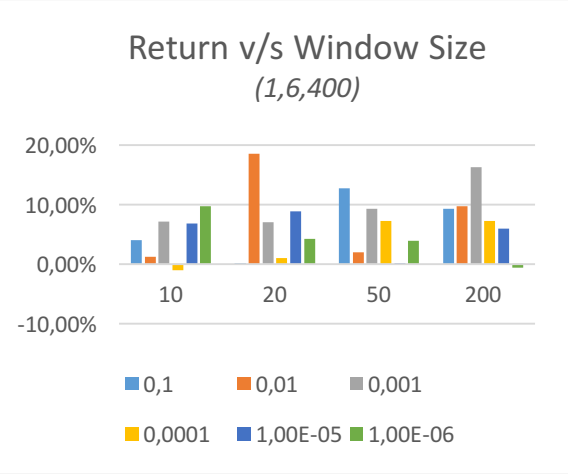
As candidate we are going to choose from 1 hidden layer (5,300) and (6,400), this are one timing strategy and one buy and hold strategy. From the 2 hidden layer agents we are going to choose (1,500) and (4,700). I choose this values because they gave the most consistent good results.

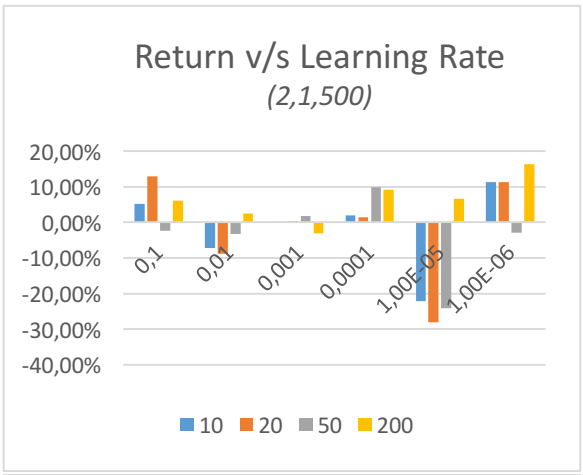
For the second iteration we had the following results:

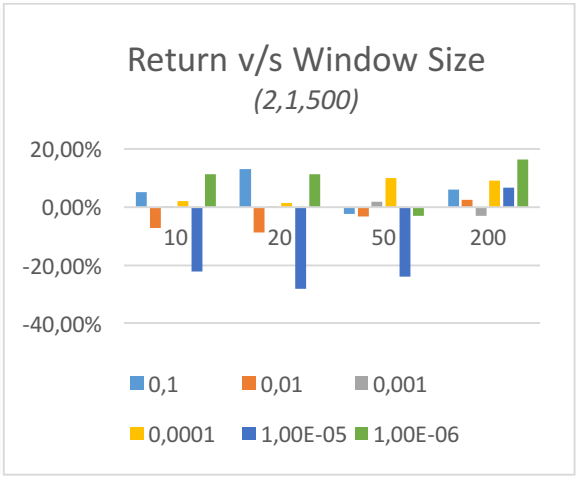


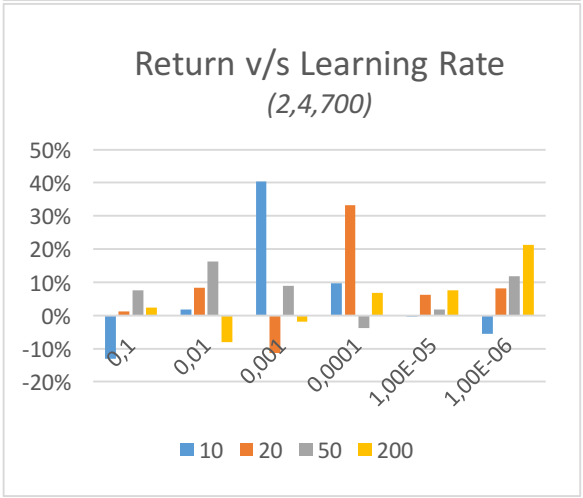


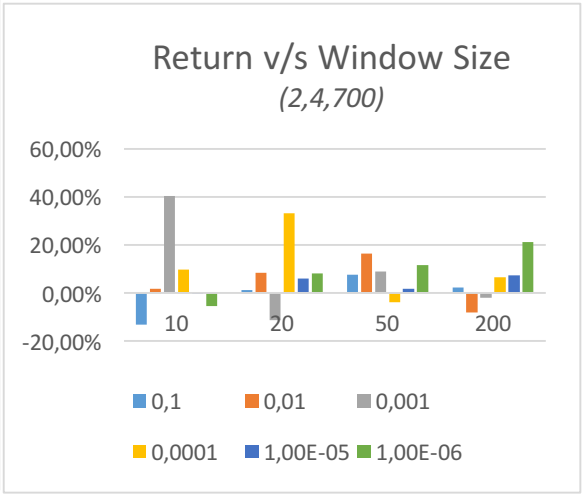












The SPY return for this period is 6,17% so every result over that value is a success.

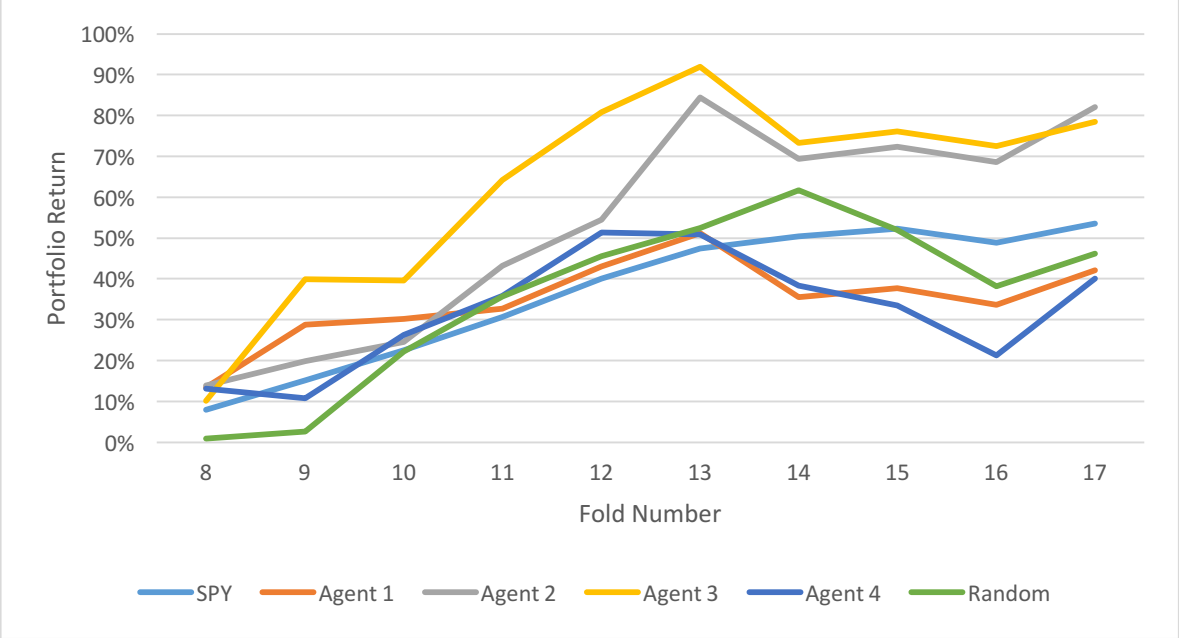
A couple of trends are apparent. The agents with one hidden layer tend to prefer a buy a timing strategy and tend to have better performance with higher learning rates. So they tend to be more impulsive and have a lot variability in their results. A higher learning rate is consistent with spiky behavior. Meanwhile the agents with two hidden layer tend to prefer a buy and hold strategy and has better performance with lower learning rates. The extra layer seems to give the agents a better capacity of recognizing which stocks have a good potential of growth. Both type of agents do better with the 200 day window size.

Looking at the results it is evident that agents (1,6,400) and (2,4,700) are the best. Agent (1,6,400) took a timing strategy 54% of the time, and had the highest scores at (0.01, 20) and (0.001,200), having a return of 18,46% and 16,29% respectively. Agent (2,4,700) took a buy and hold strategy 58% of the time, and had the highest scores at (0.0001, 20) and (1.00E-06, 200), having a return of 33,31% and 21,30% respectively. Agent 1 (1,6,400,0.01,20), agent 2 (1,6,400,0.001,200), agent 3 (2,4,700,0.0001,20) and agent 4 (2,4,700, 1.00E-06, 200) are going to be our last candidates.

We will discuss the final results in the next section.

Results

This are the results for the Roll Forward Cross validation of the refined agents:



As we can see agent 2 and 3 greatly outperform the other agents and also our benchmark SPY and the random trader. This table summarizes the final result:

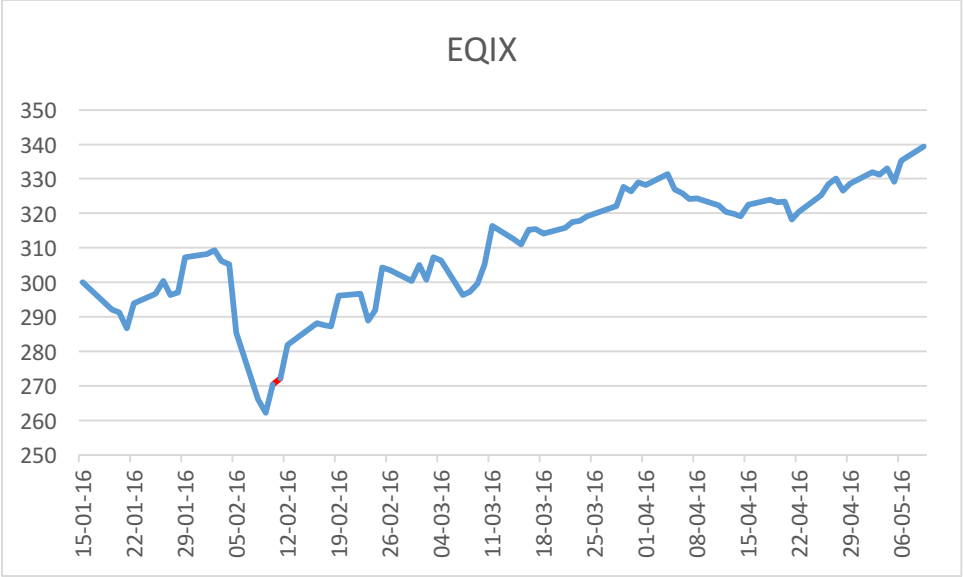
	Return
Agent 1	42,14%
Agent 2	82,12%
Agent 3	78,50%
Agent 4	40,04%
SPY	53,53%
Random	46,09%

So our final selected agent is Agent 2 (1,6,400,0.001,200). It not only did outperform all the other models in the final return it also had the most consistent results. It did better than the SPY 70% of the time. Agent 3 meanwhile was only 50% of the time.

Of this underperformance period the most characteristic is fold 14 where most of the agents has losses of an average of -9% while the SPY grew 2%. This was between 11/23/2014 and 03/31/2105. Here the volatility was really high and all of the agents picked a timing strategy that gave very bad results.

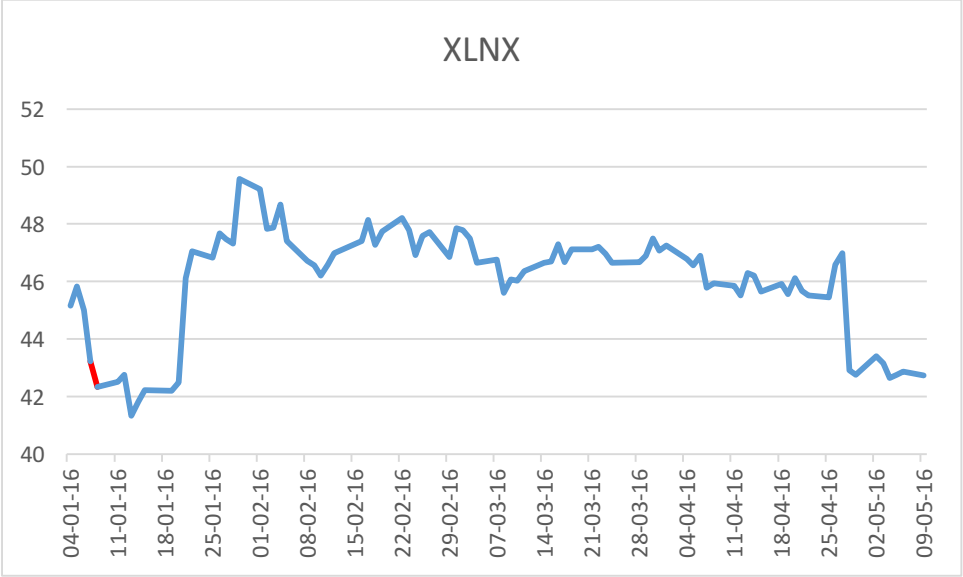
Agent 2 has a hybrid type of strategy. It starts with a Buy and Hold strategy, buying an average of 15 stocks but then it consistently buys and sells the same stock. Sometimes it sells them the next day or a few days apart. In most cases the stocks where it has this behavior are stock that are having

a high growth rate. For example, in the last fold Agent two bought 17 stocks and then in 02/10/2016 decided to buy EQUIX. Here is the chart for EQUIX.



As we can see the model decided to trade in almost the precise point where the stocks start to have a great increase in value.

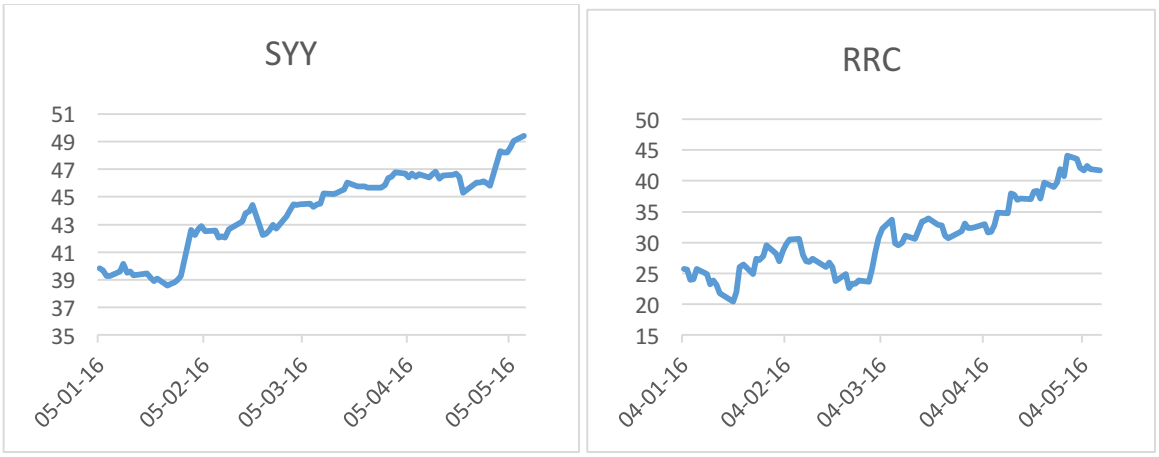
Other cases are not so clear, for example it decided to buy XLNX on 01/08/2016.



As we can see it picked a very good moment to buy, but after a big growth in price it when down slowly, and at the it went down very quickly. The agent never sold the stock so it lost money.

Finally, I did a last check by running this agent one more time with the first test we use, all the prices of 2016 and the model got incredible results. It got a portfolio return of 50,8%. Here it use a

simple buy and hold strategy buying only two stocks at the beginning January 2016. It bought and then hold SYY and RRC. Here are the graphs.

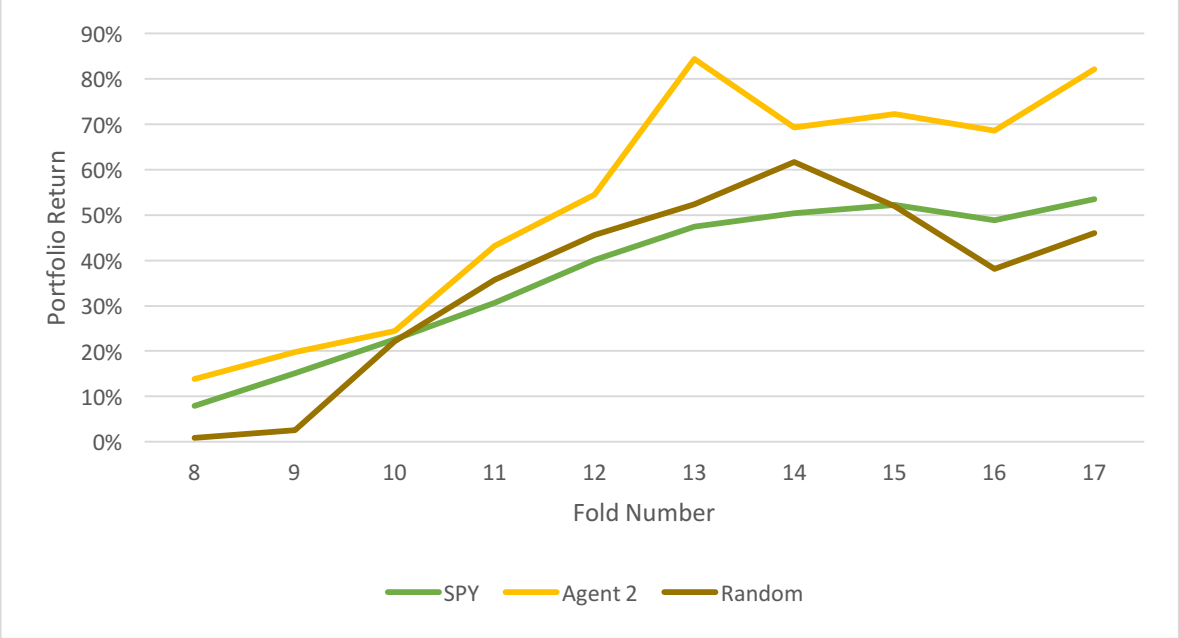


As we can see Agent 2 decided to buy this stocks in their minimum and then they grew considerably, confirming the predicted power of the agent.

Agent 2 outperforms every benchmark and provides consistent and attractive returns. This result is validated with the fact that we test it over 18 different periods and it got consistent good results over both benchmark proposed. Also by viewing the trade history we can see that the agent is a powerful predictor of stocks that will grow in the future.

Conclusion

Final results are summarized in this graph:



We have mixed two popular models in Machine Learning, Q Learning and Deep Neural Networks to produce an agent that can automatically trade stocks. We did this by creating a DNN that predicts the value of Q. We trained the agent by simulating several trade runs and then we tested our portfolio return in a test set of recent data. We then refine the model doing a grid search over the most relevant parameters. And finally after we selected our best models we did a Roll Forward Cross Validation to test the model over 18 different periods of time. We compared this results with the SPY return and a random trader and we got significantly better results, 28,6% over SPY and 36% over random trader. Moreover, this result where consistent over time.

What I found most interesting about the project is how the agent used different types of trading strategies depending on the current situation. Some models tend to prefer one strategy over the other, but they had a healthy mix of both proving to be very flexible to different situations.

I was also impressed on how well the model picked the perfect time to buy a stock, that is really hard to do in real life.

One of the biggest difficulties I had in this project is that the state dimensionality and the action dimensionality was very big, so I had to do a lot of train sessions. Each of this sessions toke several minutes and refining the model by performing a grid search over the parameters toke a couple of days. This didn´t let me to look further into the model, for example I only saw how the model behave with two hidden layers, maybe with more layers I could have gotten other results.

The other problem with this approach is that it is very hard to see what the model is really doing, it behaves more like a black box.

Overall the model did meet my expectations of creating an agent that gets great portfolios return and that can really be used as tool to pick stocks that have good growth potential. But further improvements can be made to the model. I would use it in real life as tool for assisting me in selecting a stock but it still needs further improvement in order to let it automatically trade.

Improvement

There some ideas that could really improve the model and outperform its current results.

The first idea is to include the current return since purchase of each stock. This is useful because the current network has no idea if it is winning or losing money in a trade. Usually when trading one has *stop loss* rules or *cash in* rules that tells us when to stop, probably if the agent develops this sort of strategies it can get a big boost in performance. Let´s take the case of stock XLNX which had a great performance in the beginning but then it loosed value gradually, maybe if the agent was aware of the consistent drop in return it would have sell it.

The second idea is to combat the problem of high dimensionality by using some sort of feature selection. Maybe we could select just the 10% of the stock of the S&P500 according to some criteria. Maybe we could pick the ones that are most traded or the ones that have higher correlation. This could help the model converge faster.

Lastly we can add other information to the model. Stocks prices are very influenced by macroeconomic factors that the current model is not taking in account. Other instruments like options or bonds are a good predictor of the market sentiment towards a stock. Finally, a sophisticated agent could read the news and see if the current new could affect the price of a stock.