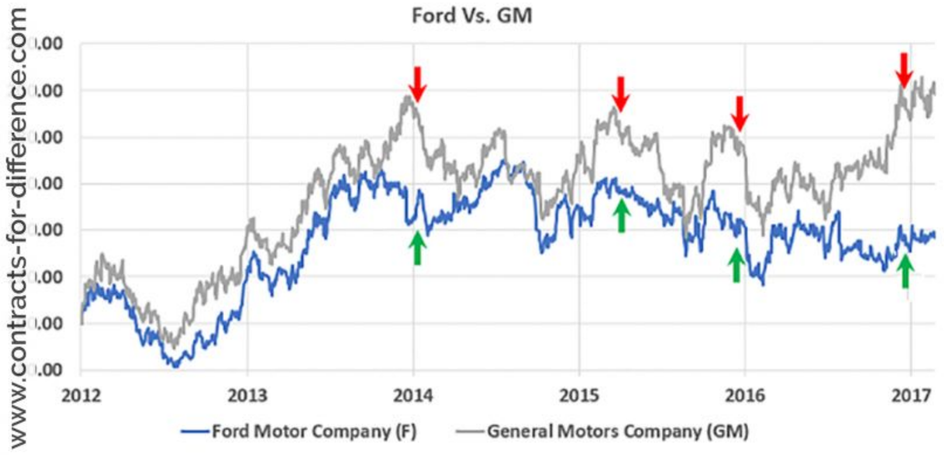
Pairs Trading using Reinforcement Learning

Madeleine Lim, Robin Markwitz, Ruan Scheepers, Asaf Silman

What is pairs trading?

Pairs trading is a trading strategy which identifies a pair of stocks which exhibit similar price movements in order to exploit financial markets that are out of equilibrium.



Pairs Trading with Machine Learning

KO Chung Wa, WONG Wen Yan, THAM Brendan Guang Yao
Supervised by: Prof Nevin L. ZHANG

Introduction & Objectives

Pairs Trading Concept

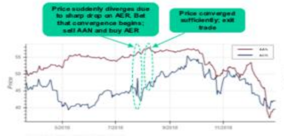


Figure 1: Normalized Price Charts of Sample Stock Pair Using AAN-AER

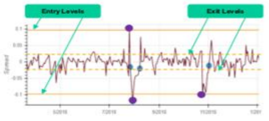


Figure 2: Spread of AAN-AER (red line). This chart indicates the trade actions (purple circles) to bet price convergence.

Pairs Trading is a statistical arbitrage trading strategy that bets on the relative pricing of two assets which exhibit similar price movements. Due to idiosyncratic factors such as supply and demand, prices of similar assets may temporarily diverge due to market imbalance, causing the price of one asset to move much more than its pair. To illustrate the example, consider the following pair of stocks Microsoft (MSFT) and Amazon (AMZN). MSFT and AMZN operate in the same sector, have similar company size, and similar company strategy. Therefore, it is likely that their prices move similarly.

However, suppose AMZN has announced underwhelming results in its business while MSFT announced results that met expectations. As an initial response, the price of AMZN will decrease faster than MSFT since there is a net surplus of sellers, causing the prices to **diverge**. Over time, the initial overreaction causes the **spread** of MSFT-AMZN to increase, and when it is sufficiently high (when AMZN is *too undervalued* relative to MSFT), we can place a **bet that the spread will converge**. In summary, we buy the relatively undervalued asset and sell the overvalued one at some optimal entry levels **to bet on convergence**; we then exit when spread converges. *Figures 1 & 2* illustrates the concept using a different pair (AAN and AER stock), both listed in NYSE Technology Sector.

What is the spread? – The spread generally refers to the difference in prices of two assets. Intuitively, an ideal pair is one that has a **mean-reverting** spread.

Method	Spread definition
Distance	$Y_t - X_t$
Cointegration	$\log Y_t - \alpha \log X_t - \beta$

Figure 3: Definitions of common pairs trading spread definition

Objectives

- Implement existing pairs trading strategies as baselines for comparison (Distance, Cointegration – Rolling OLS, Cointegration – Kalman Filter).
- Develop a profitable pairs **trading agent using Reinforcement Learning** (RL) and generalize the model to other sectors or assets.
- Implement a User Interface (UI) to monitor and analyze performance of RL Trading Agent.

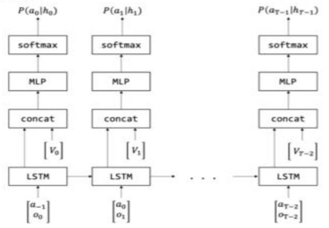
Methodology - Reinforcement Learning Agent

Problem Definition

The pairs trading problem can be understood as: *Given one spread at a time, how do we decide when to **buy the spread, sell the spread, exit the spread.*** We modeled this problem as a Partially Observable Markov Decision Process (POMDP):

Observation Space O	$\mathbb{R}^{1+2+1}$ where the first dimension is the agent portfolio value, the second and third dimensions are the price of two given stocks, and the last dimension represents the spread of the normalized log price estimated by rolling window regression.
State Space S (partly unobserved)	$\mathbb{R}^{1+3}$ where the first dimension is the agent portfolio value and the last 3 dimension together is the one-hot encoding of the classes in {"spread is over-priced", "spread is under-priced", "spread is in the normal price level"}
Action Space A	One-hot encoding of the classes in {"selling the spread", "buying the spread", "having no position on the spread"}.
Reward Function r_t	$r_t = PV_{t+1} - PV_t$, where PV_t is the portfolio value at time t.

Model Architecture


$P(a_0|h_0)$
$P(a_1|h_1)$
$P(a_{T-1}|h_{T-1})$
softmax
MLP
concat
$[V_0]$
$[V_1]$
$[V_{T-2}]$
LSTM
$\begin{bmatrix} a_{-1} \\ o_0 \end{bmatrix}$
$\begin{bmatrix} a_0 \\ o_1 \end{bmatrix}$
$\begin{bmatrix} a_{T-2} \\ o_{T-2} \end{bmatrix}$
Figure 4: The policy network architecture of RL agent. T is the final time step in every episode. $o_t \in O, a_t \in A, V_t$ is portfolio value.

To solve this POMDP, we adopted the Recurrent Policy Gradient method. Our proposed Policy Network architecture (as shown in Figure 3) takes in a sequence of historical actions and observations as inputs and outputs a probability distribution over all possible actions.

Results

Experiment Setup

All **technology stocks** available in the New York Stock Exchange (NYSE) that have no missing data point from 2015-01-01 to 2019-01-03 were selected. This amounts to 116 stocks, which provides 6670 possible pairs. We have three experiments where the training and testing data are described below.

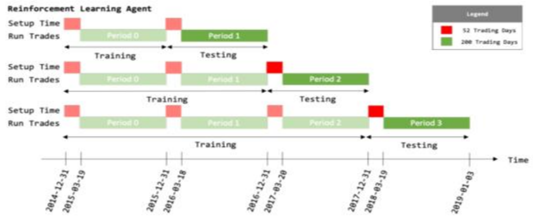


Figure 5: Timeline of backtesting

Performance Review

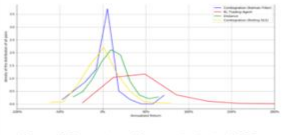


Figure 6: Comparison of returns distribution for RL trading agent vs baseline strategies (in period 3)

Strategy/Portfolio	Period 1	Period 2	Period 3
Distance	+2.86%	-1.79%	+10.48%
Cointegration (Kalman Filter)	+4.28%	+0.46%	+0.26%
Cointegration (Rolling OLS)	+3.21%	-3.34%	-0.59%
Reinforcement Learning Agent	+32.69%	+40.83%	+43.78%
S&P 500 Index	+9.23%	+12.65%	−7.60%
PSE Index	+14.80%	+18.74%	−7.76%

Figure 7: Summary of mean returns (Tech)

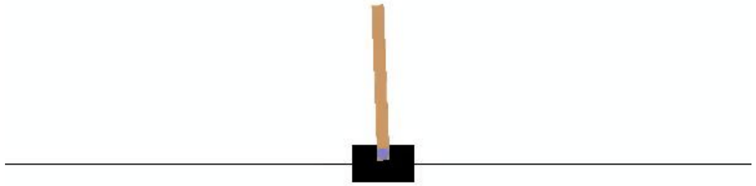
Strategy/Portfolio	Period 1	Period 2	Period 3
Reinforcement Learning Agent	+53.80%	+69.60%	+46.02%
S&P 500 Index	+9.23%	+12.65%	−7.60%
NYE Index	+2.69%	-5.15%	-2.58%

Figure 8: Summary of mean returns (Energy)

Overall, the RL trading agent outperforms both baseline pairs trading strategies and market indices in every comparison period *(Figure 7)*. We have also successfully demonstrated the generalization ability of the agent by training the model on tech dataset and testing on all pairs from NYSE listed energy sector stocks with significant outperformance *(Figure 8)*.

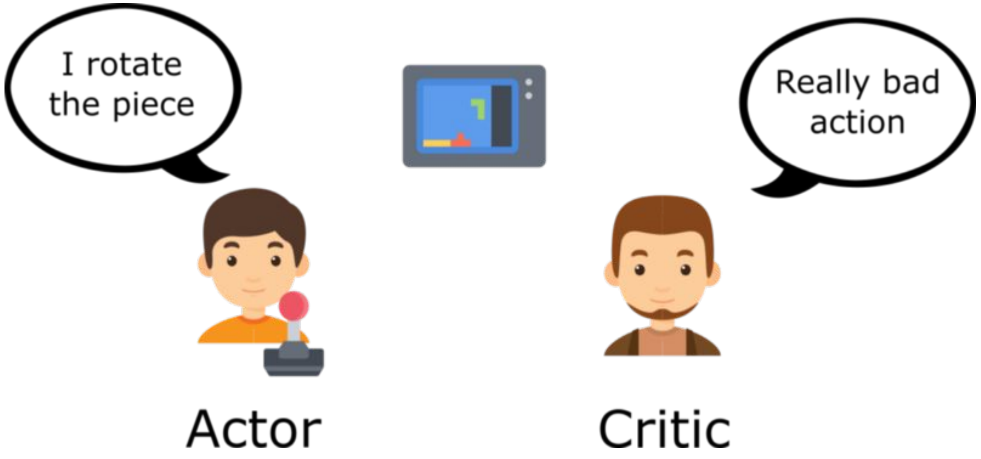
Reinforcement learning

- A machine learning method that involves a model with a certain number of actions that it can take in an environment
- The model learns about its environment through discovery of which actions to take in certain situations
- We will be using the actor-critic (AC) reinforcement learning method



‘Cart-pole problem’ Environment by Gym Open AI
http://gym.openai.com/envs/CartPole-v1/

Actor-Critic Model



An intro to Advantage Actor Critic methods by Simoni
https://www.freecodecamp.org/news/an-intro-to-advantage-actor-critic-methods-lets-play-sonic-the-hedgehog-86d6240171d/

- A ML method with two models
- The actor performs an action
- This action affects the environment
- The critic judges the effect of this action on the environment
- The models help each other

The `gym` environment

`data_source.py`

- Loads stock data in
- Stock data comes from alpha-vantage

`market_metrics.py`

- Converts stock data into observable values
- Calculates the spread between the two stocks

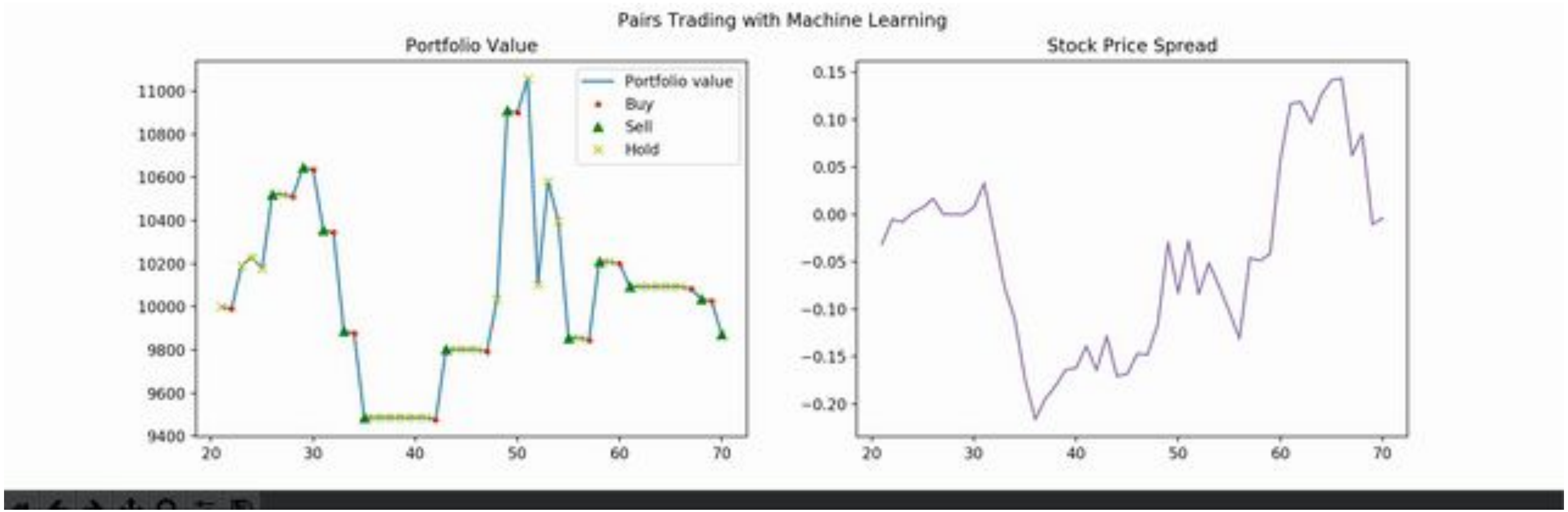
`trading_sim.py`

- Applies the action chosen by the actor to the environment

`pairs_trading_env.py`

- Implements the **`gym`** interface in order to initialize the environment

Implementing `gym` – Random Strategy



Actor-Critic setup for this project

Action Space: {Buy, Sell, Hold}

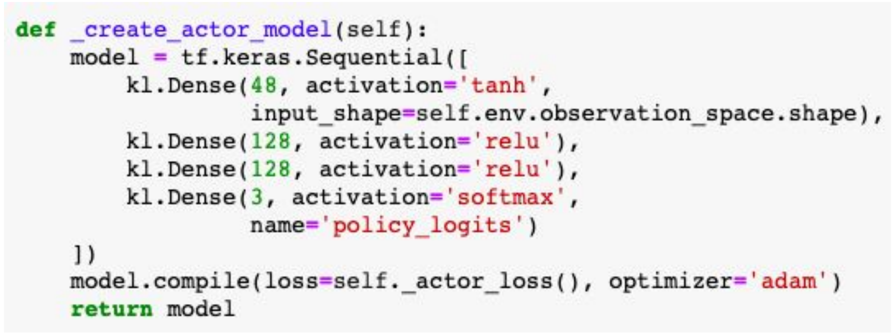
Buy = Enter the spread, ***Sell*** = Leave the spread, ***Hold*** = Do nothing

Actor: Based on the environment, it either buys, sells or holds

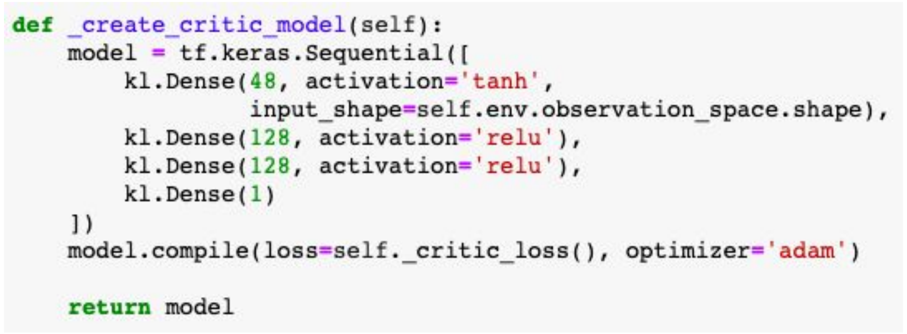
Critic: Based on the environment, it estimates the reward of the current state

Over time, the actor learns the effect of its actions on the environment, and the critic learns to more accurately estimate reward

Keras Actor-Critic Model



```
def _create_actor_model(self):
    model = tf.keras.Sequential([
        kl.Dense(48, activation='tanh',
                 input_shape=self.env.observation_space.shape),
        kl.Dense(128, activation='relu'),
        kl.Dense(128, activation='relu'),
        kl.Dense(3, activation='softmax',
                 name='policy_logits')
    ])
    model.compile(loss=self._actor_loss(), optimizer='adam')
    return model
```



```
def _create_critic_model(self):
    model = tf.keras.Sequential([
        kl.Dense(48, activation='tanh',
                 input_shape=self.env.observation_space.shape),
        kl.Dense(128, activation='relu'),
        kl.Dense(128, activation='relu'),
        kl.Dense(1)
    ])
    model.compile(loss=self._critic_loss(), optimizer='adam')

    return model
```

- Actor uses logits in order to determine an action probability distribution of size 3 (Buy, Sell, Hold probabilities)
- Critic has only one output layer: the estimated reward
- Two different loss functions

Method

- The Actor-Critic model is trained on close to **one year** of data
- The two stocks used are **Apple (AAPL)** and **Microsoft (MSFT)**
- Since these two companies sell similar items in a similar market, they are suitable for pairs trading techniques
- The model then uses the strategy that it learned on **one month** of unseen stock data
- A comparison is made between this strategy and a baseline random strategy

Pairs Trading v1

- The models use an **observation space** in order to make their predictions
- In this case, the observation space is:

  { Percentage change in stock 1, Percentage change in stock 2, Change in spread }

Results for Pairs Trading v1

- A simple strategy is to randomly **Buy**, **Sell** or **Hold** on each day

- Can compare this strategy to the strategy learned by the Actor-Critic model by calculating the final rate of return over the testing time

- Results were essentially equivalent to the random policy: **further optimization is needed.**

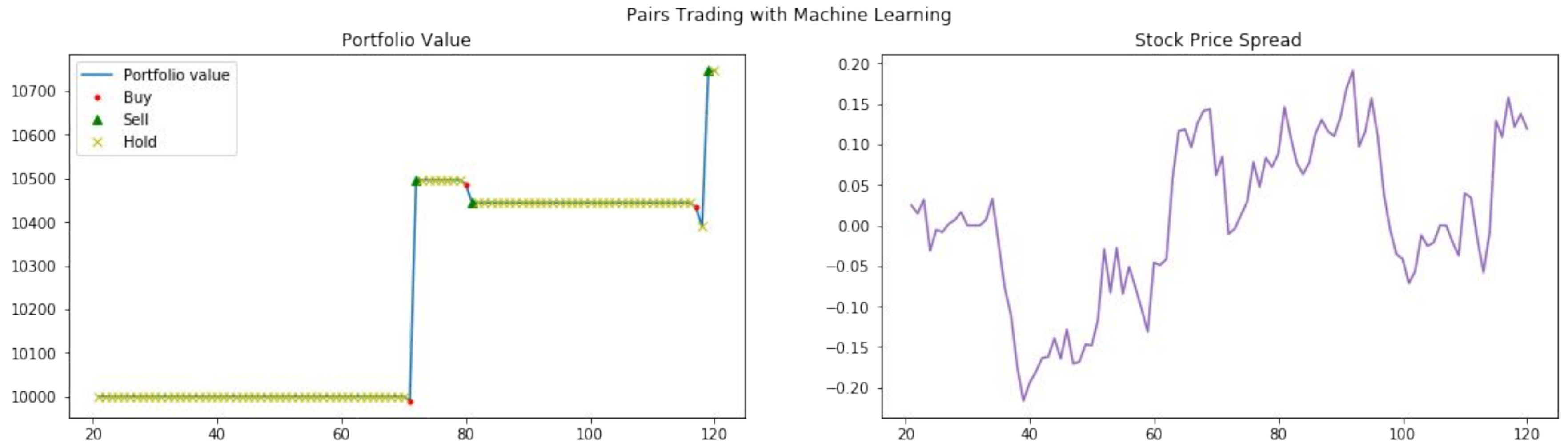
What went wrong

- The model did not have the opportunity to learn if it was in or out of the spread
- In order to mitigate this, the **observation space** was changed:

  { Percentage change in stock 1, Percentage change in stock 2, Change in spread, ***Spread Status*** }

- Additionally, the model can be punished for performing illegal moves in the training phase

Results for Pairs Trading v2



- Model is aware of spread status
- **Risk averse** - holding because need more information
- Needs some more tinkering

Pairs Trading v3

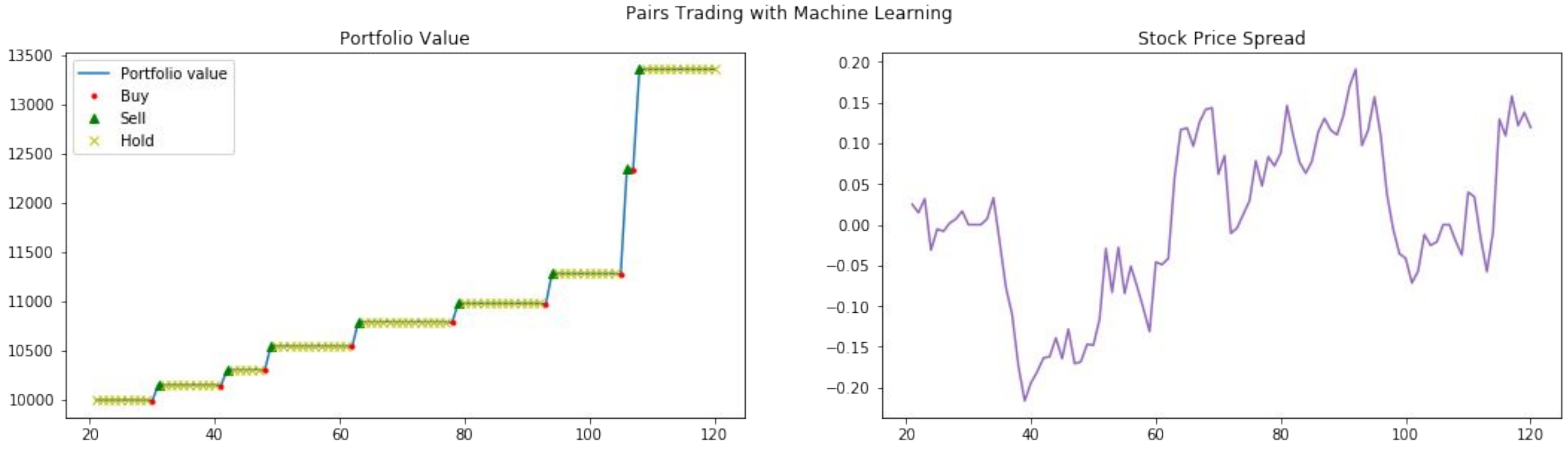
- For the third model, we increase our observation space again:

  { Spread, Spread status, Percentage change **for each day starting from 5 days ago** for stock 1, Percentage change **for each day starting 5 days ago** for stock 2}

- Activation function for the final layer of the critic was changed to **linear**

Results for Pairs Trading v3

- Consistently outperformed the random valid move strategy
- On average, returns were **$150** higher on an unseen data set
- Increasing portfolio value



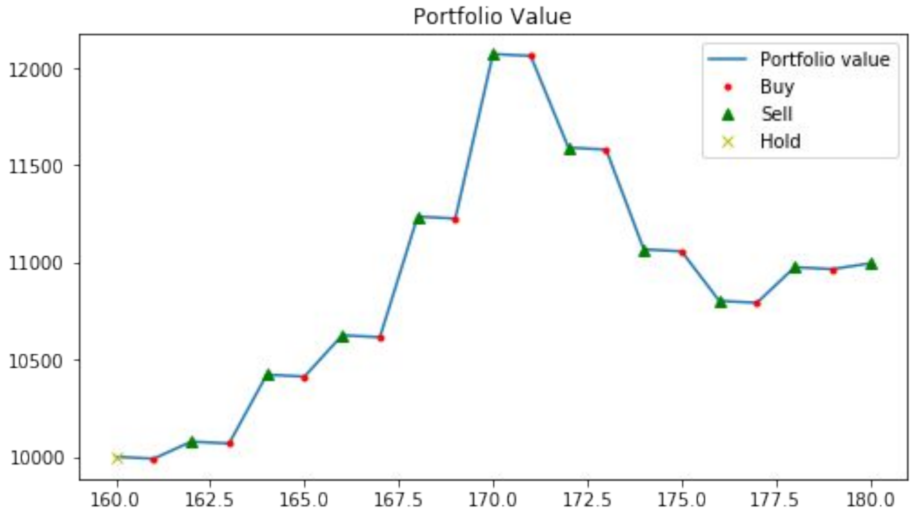
Encourage Holding after purchasing into the spread

- Penalise Holding (various parameters)

- Slowly ramp up penalties for ‘illegal’ moves to encourage buy/sell in early generations

- Bankrupt ‘game’

Too Far



Portfolio Value
Portfolio value
Buy
Sell
Hold
12000
11500
11000
10500
10000
160.0
162.5
165.0
167.5
170.0
172.5
175.0
177.5
180.0

Future Potential

- Expand the action space on the buying and selling of different stocks per amount, instead of the {Buy, Sell, Hold} action space as currently configured
- Use an RNN for the actor, instead of a sequential model, in order to find trends in the data

- Use an RNN to feed ‘predicted’ spread to the actor
  - Can train RNN independently
  - Freedom to train actor without RNN

Thanks for Listening