StockAgent: Application of RL from Lunar Lander to stock price prediction

Caitlin Stanton (stanton1@stanford.edu) and Beite Zhu (jupiterz@stanford.edu)

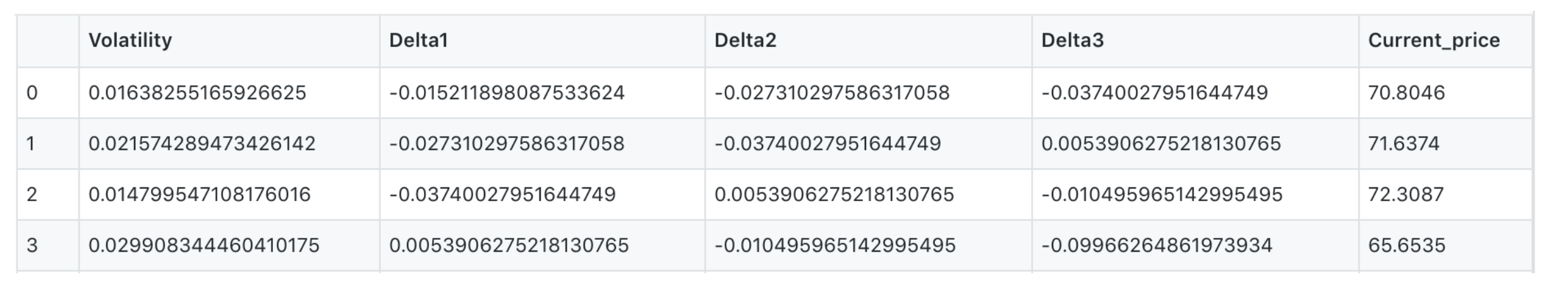
CS229 Machine Learning

Problem Statement

- Reinforcement learning has been used to learn to all sorts of games, from chess to arcade games.
- We first trained an agent to play the arcade game Lunar Lander, using both policy gradient descent and deep-Q learning.
- Then, using very similar techniques to the ones implemented on Lunar Lander, we train an agent to learn how to invest in stocks.

Data

- Used Lunary Lander environment provided by OpenAI gym.
- Data for stock prediction came from IEX's API. Provided 5 years worth of Apple and Google stock data.
- We reformatted our stock data to have one entry for each week (instead of each day), and consolidated the number of features (see section on states below for details).



	Volatility	Delta1	Delta2	Delta3	Current_price
0	0.01638255165926625	-0.015211898087533624	-0.027310297586317058	-0.03740027951644749	70.8046
1	0.021574289473426142	-0.027310297586317058	-0.03740027951644749	0.0053906275218130765	71.6374
2	0.014799547108176016	-0.03740027951644749	0.0053906275218130765	-0.010495965142995495	72.3087
3	0.029908344460410175	0.0053906275218130765	-0.010495965142995495	-0.09966264861973934	65.6535

Figure 1: Formatted stock data, as inputted into our neural network

States, Actions, and Rewards

- Lunar Lander:4
  - **States**: 8-dimensional vector, including information such as the lander's position and orientation in space, and amount of fuel used.
  - **Actions**: Do nothing, fire right engine, fire left engine, fire main engine.
  - **Rewards**: Rewards for landing with feet down, and penalties for wasting time, landing far away from the pad, and wasting fuel.
- Stock Market:
  - **States**: Volatility (standard deviation / mean) from past 3 weeks, weekly change in stock price from past 3 weeks, current price of stock, current cash level, current owned stock value.
  - **Actions**: "Hard" sell, "soft" sell, do nothing, "soft" buy, "hard" buy. For our purposes, we initially set "hard" to mean buying/selling \$100 worth of stock, and "soft" to mean buying/selling \$10 worth of stock.
  - **Rewards**: Change in portfolio for that week.

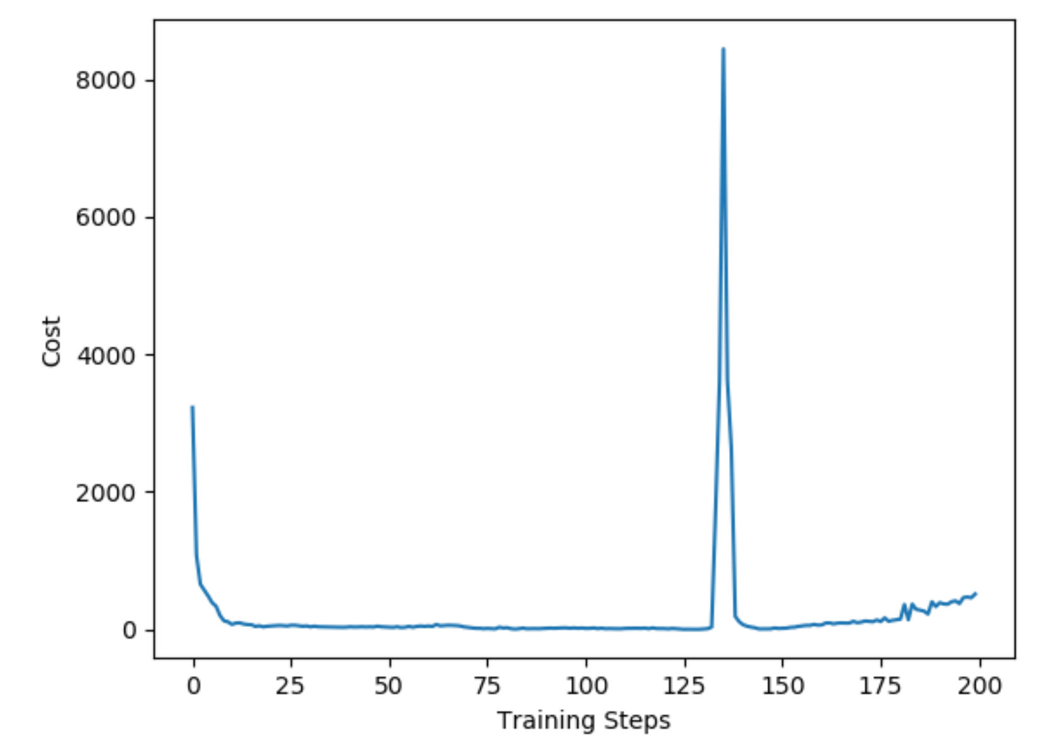


Figure 2: The plot of cost against epochs. The spike is probably due to the exploration in the model.

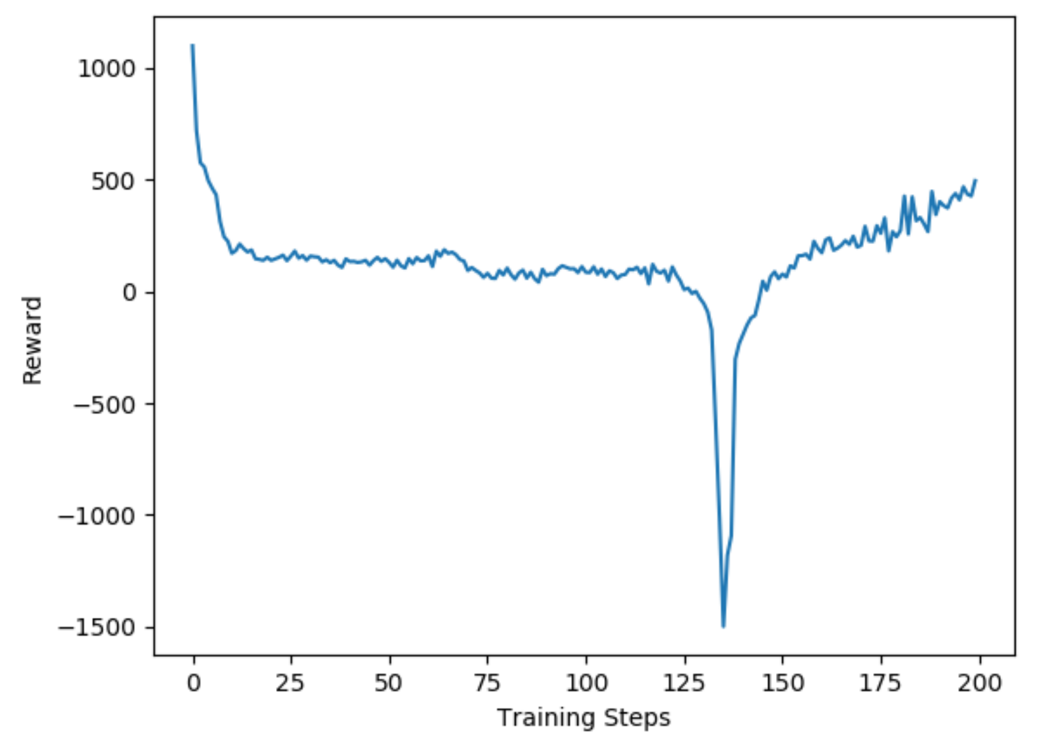


Figure 3: The plot of reward against epochs. The plummet is probably due to the exploration in the model.

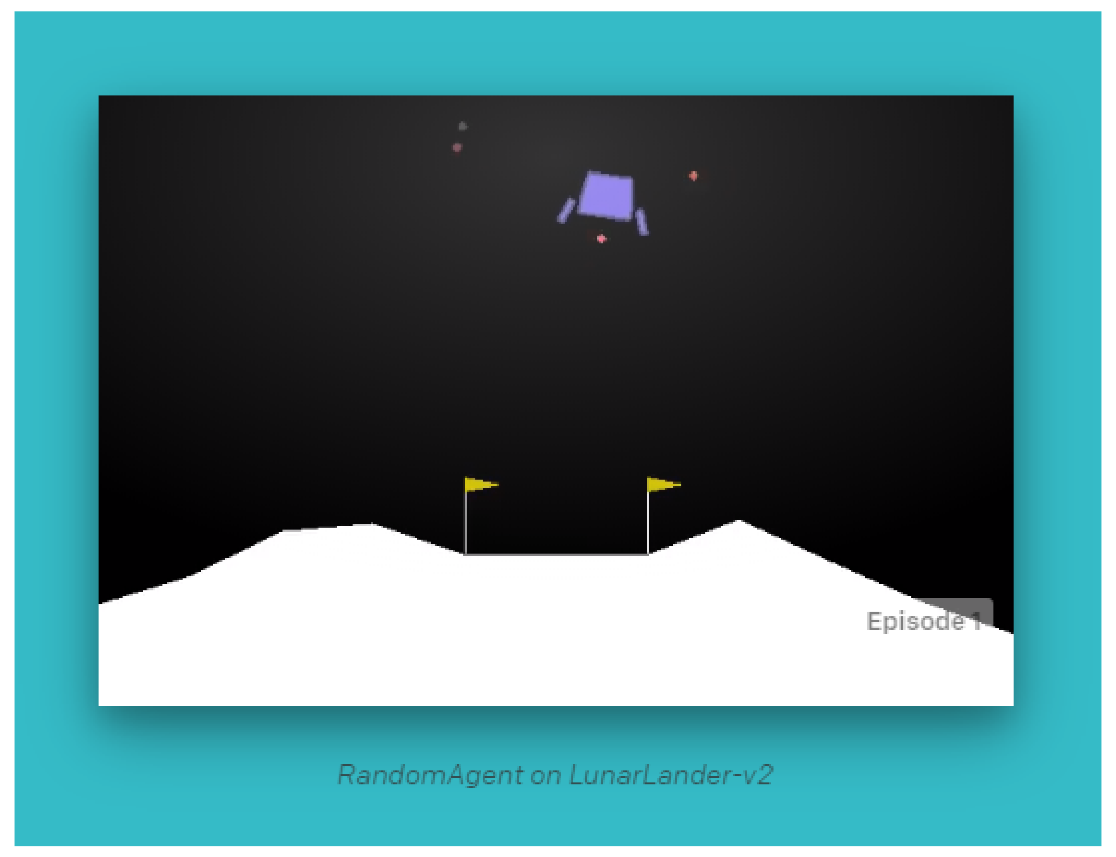


Figure 4: Lunar Lander environment

Model Parameters

- Lunar Lander:
  - Fully connected neural network of shape $(\text{input}, 8, 8, 5)$
  - Exploration rate: starts at 0, then increments by 0.001, with max of 0.9
- Stock Market:
  - Fully connected neural network of shape $(\text{input}, 8, 8, 5)$
  - Default exploration rate: 0.05
  - Train for 1000 epochs, where each epoch plays through 200 weeks of stock data.

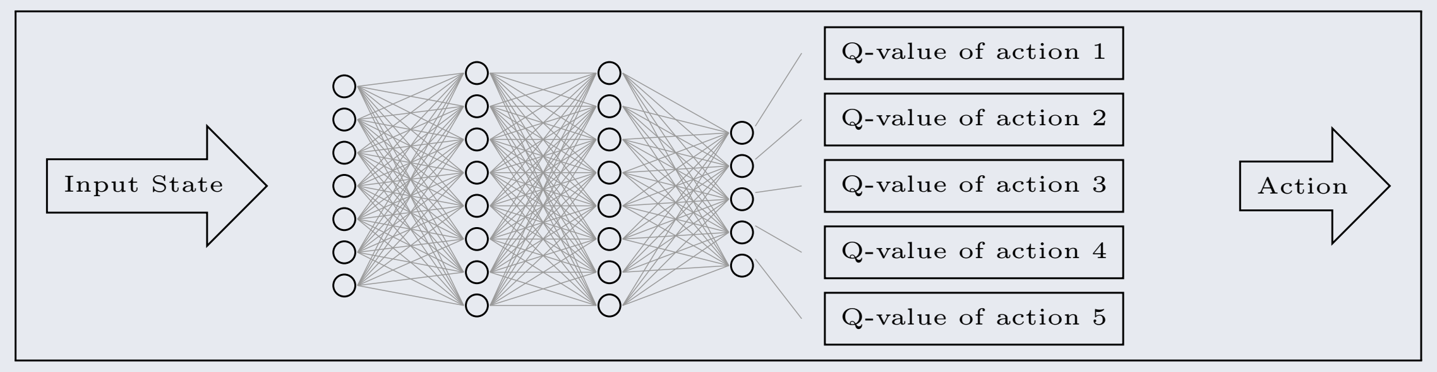


Figure 5: Graph of the StockAgent using deep Q learning.

Results for Stock Prediction

- Model took about 200 epochs to converge.
- Adjusting Soft buy/sell action size, here is a table containing the information of the final portfolio value :

Network	GOOG	AAPL
Soft=1	66.61	235.42
Soft=5	448.15	342.22
Soft=10	50.77	943.07
Soft=20	120.14	19.247
Soft=50	5716.30	8566.56

Table 1: Final portfolio value under different soft buy/sell value.

- Adjusting the exploration rate, ε:

Exploration rate	GOOG	AAPL
$\varepsilon = 0$	249.63	1905.90
$\varepsilon = 0.01$	458.92	1713.68
$\varepsilon = 0.05$	592.98	476.66
$\varepsilon = 0.1$	116.36	264.31
$\varepsilon = 0.2$	-30.76	129.55

Table 2: Final portfolio value under different exploration rates.

Loss Functions

$$\text{Loss} = \|Q - \hat{Q}\|_2$$

where we generate the target $\hat{Q}$ by

$$Q(s,a) = R(s,a) + \gamma \max_{a'} Q(s',a')$$

for s' is the state from s via action a. γ is the discount factor. In lunar lander we sample some size of memory to do replay to generate $\hat{Q}$. In StockAgent, the whole training is recorded and served as memory.

Discussion/Future Directions

- All the parameter choices we tested performed worse than just always choosing "hard" buy each week; with such a policy, we have final portfolio value of about 12,000 for Apple. We are currently unsure of why our model actions skew so heavily in favor of "soft" buy instead of "hard" buy (except possibly that our model is quite risk-averse).
- One thing we would like to try in future training episodes would be to also train and test on stocks that are not very successful (i.e. do not trend upwards in the long run).
- At the moment, each of our training simulations last for 200 weeks, while our test dataset consists of 30 consecutive weeks. In the future we would like to use a longer test set, to mimic the more long-term gains that our model is being taught to achieve.
- If our goal is to precisely model/predict Apple stock, we might add additional features, such when new iPhone or Macbook models are being released. We had a number of spikes in our Apple data that we didn't see in our Google data, and we think that these might correlate with such events.

Acknowledgements

We would like to thank our mentor, Mario Srouji, for his guidance throughout this project.

References

[1] John Schulman Filip Wolski Prafulla Dhariwal Alec Radford Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017.

[2] Gabriel Garza. Deep reinforcement learning-policy gradients-lunar lander! *Medium*.