Work Trial Results

Emi Soroka

December 6, 2024

1 Running the code

I use marimo Python notebooks, which are `.py` files that can be run standalone or as notebooks.

To set up marimo:

```
pip install marimo
marimo edit task1.py
```

Marimo provides advantages over Jupyter because it uses a computational graph to track dependencies and automatically re-run dependent cells after changes are made, preventing out-of-order execution and unreproducible notebook results. It's also easier to version control.

2 Task 1

Construct and code the linear OW model and nonlinear AFS model, and visualize the distribution of price impact based on the given data.

Results

(Figure 1.) I used the provided data with the following preprocessing: data was binned into 10 second intervals using Pandas to aggregate it, and the signed volume was used for $\dot{Q}$. This followed an example in the Handbook of Price Impact Modeling. I did not remove the after hours trades from the data.

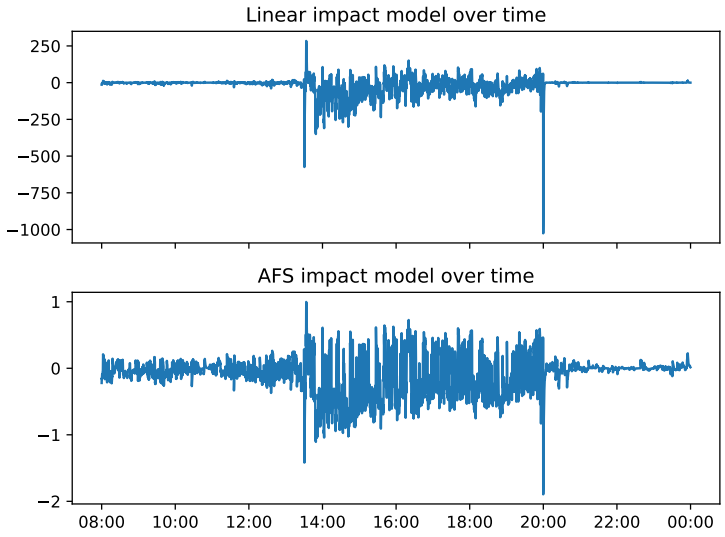

Figure 1: Price impact computed from `merged_data.csv`.

The AFS model yields a smaller price impact for the same β and λ parameters, and smooths out the impact of large spikes in signed volume. This makes sense since AFS is a concave model, thus it penalizes larger trades less per share than mid-sized trades.

3 Task 2

Implement and code the optimal strategy with Linear Impact and visualize the Sharpe Ratio plots in Section 6.2.

Results

The results look similar but are off by a scaling factor (Figure 2). This is probably caused by confusion on how the paper defines daily risk, which seems to scale with the number of shares, vs PnL.

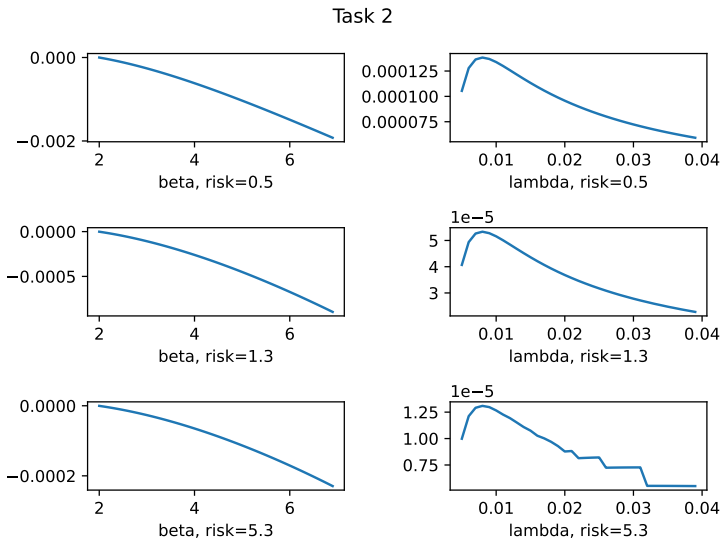

Figure 2: Plots generated by optimal strategy. Plots follow the expected shape, however they are off by a scaling factor. The expected behavior of a decrease as the risk increases is still observed.

4 Task 3

Implement and code the Deep Learning Algorithm in for discrete setting in Appendix C.2 and visualize the training loss for different network structures in Appendix C.2.

Results

I implemented the deep learning algorithm for the linear model (NetSimple) in Equation C.18 and the linear model (NetLinear). I didn't implement the NetPower modification due to time constraints. In my implementation, NetSimple performed better than NetLinear (Figure 3).

Because the paper implements a custom environment, defining the signal f_n (Figure **??**) for the neural network as well as the relationships between f_n, Q_n and J_n^θ, I implemented the policy gradient algorithm in my code including rollouts of the policy θ on batches of signals.

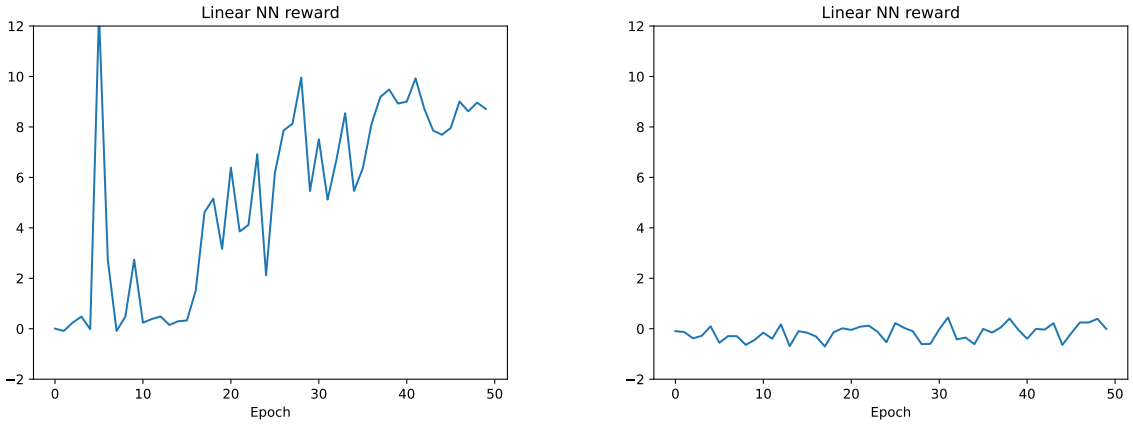

Figure 3: Reward function for NetSimple model (left) and single-layer model (right).

Unfortunately my code is slow; it should be parallelized but I don't have time to do it. I also don't have GPUs available on my computer, and observed that about 50% of the computational time was due to backpropagation so I concluded the benefit of optimizing the code would be limited. Due to this, I used a smaller batch size but the same number of 50 epochs.

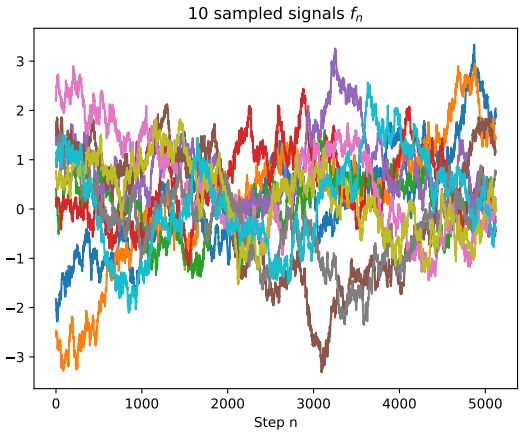

Figure 4: Visualization of random samples $\{f_n\}$.