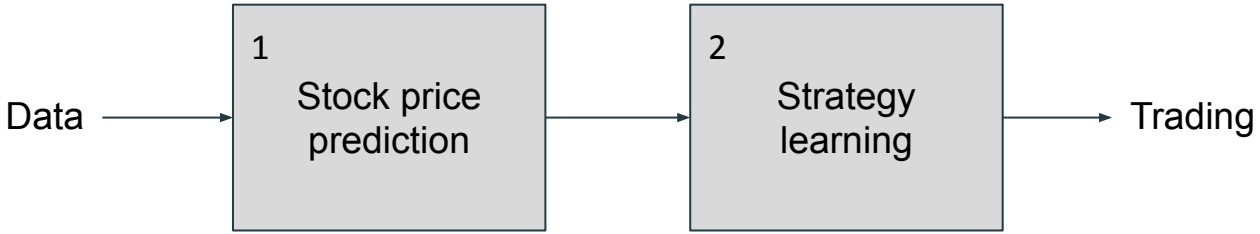
Project Update
2022/6/16

Jeff Schneider
Shaofeng Qin
Bo Wu

Two step algorithm



1. Supervised Learning to predict the next day's closing price
2. Reinforcement Learning to use the predictions to optimally trade

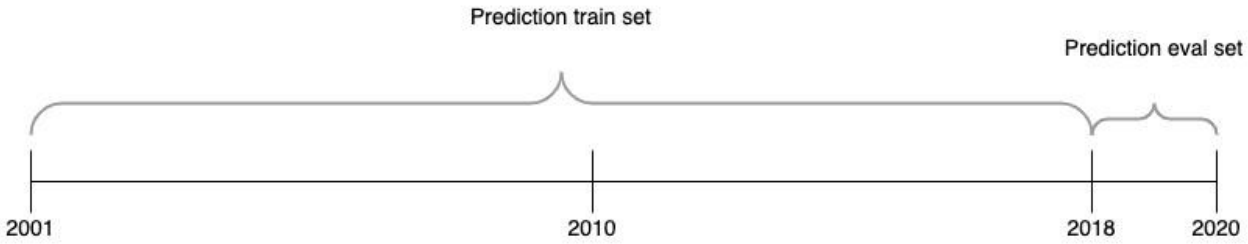
Stock price prediction

1. Data Scheme
2. Models’ Architectures
3. Performance

Data Scheme

Input(X): using look_backward = 30 days' features: daily highest, lowest, closing price, volume, volatility, moving averages, Dow Jones Industrial Averages,.. Etc

Output(y): next day's predicted price (closing price)



Model Architectures

(1) Baseline model

1 Linear layer(MLP) (35(INPUT FEATURES), 256)

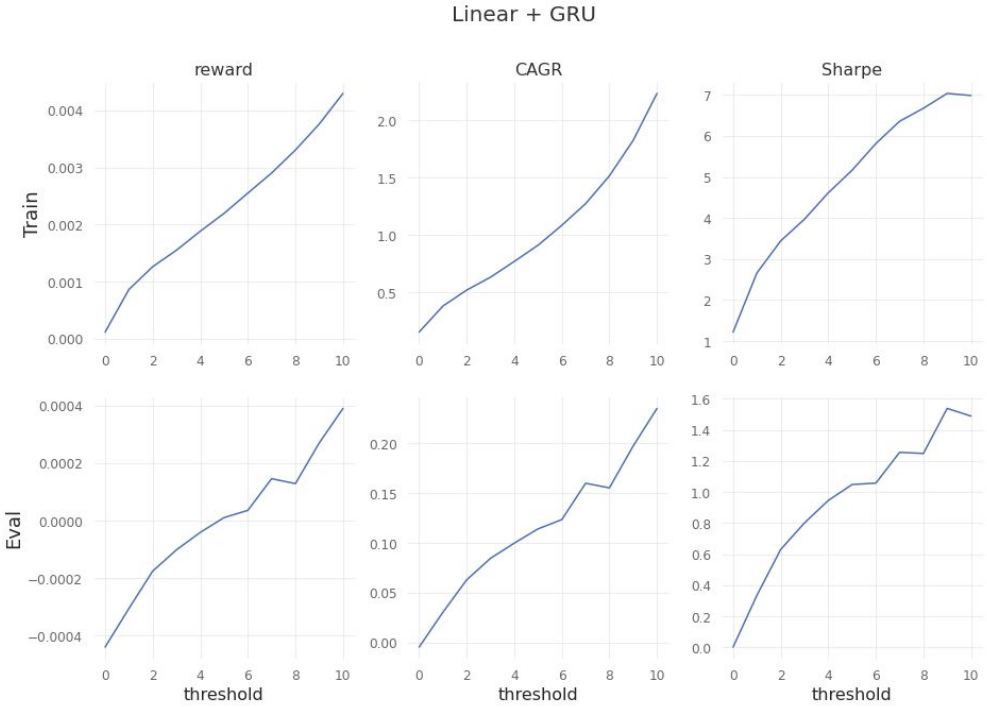
+

1 layer of single directional GRU (256, 512)

+

1 linear output layer(map to 1-D price) (512, 1)

Performance (threshold strategy for 20 stocks portfolio)



Episode: threshold—> how many models need to agree to say that the stock will rise, buy the stock

n can range from 0 models (always buy and hold) to all models (only buy if 100% sure it will rise)

Evaluation metric: rewards, CAGR(Compound Annual Growth Rate), Sharpe Ratio

Model Architectures

(2) Deeper Baseline model

2 Linear layers(MLP) (35, 128), (128,256)

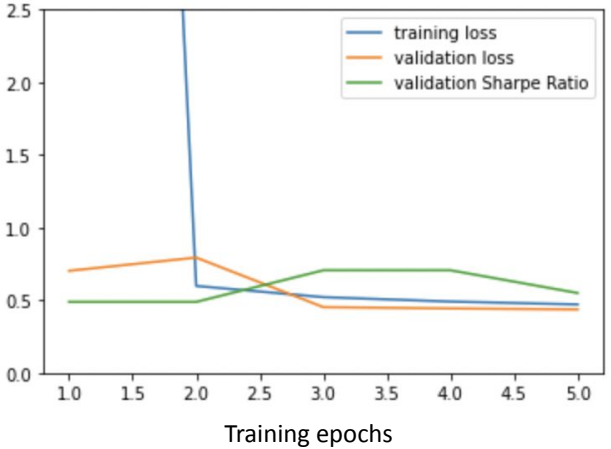
+

2 layers of bidirectional GRU (256, 512)

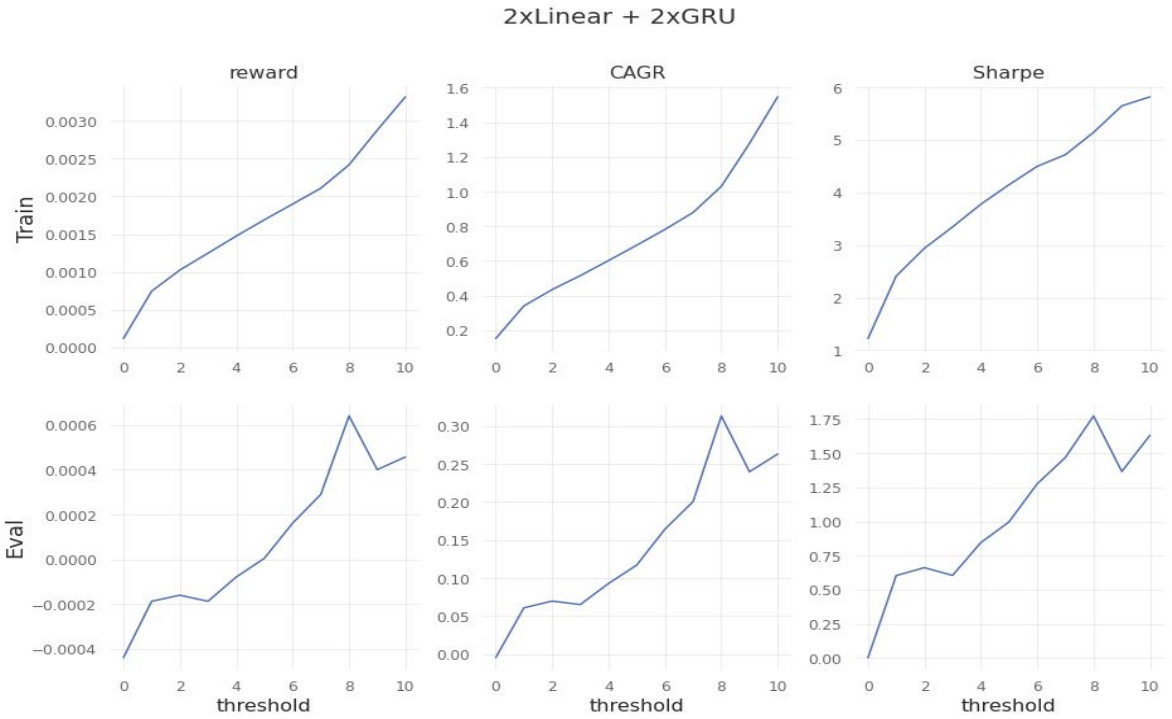
+

2 MLP output layerS(map to 1-D price) (1024, 512), (512,1)

Training-validation curve for a single model for stock “AMM”



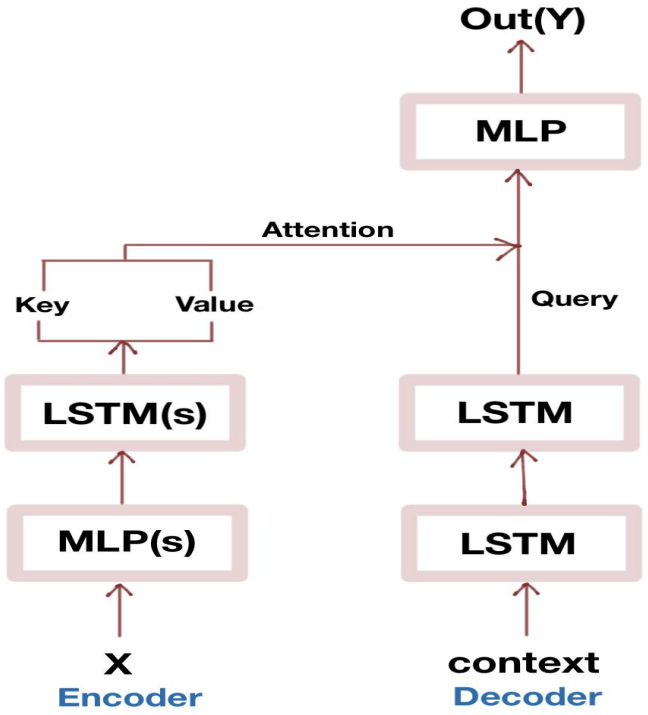
Performance



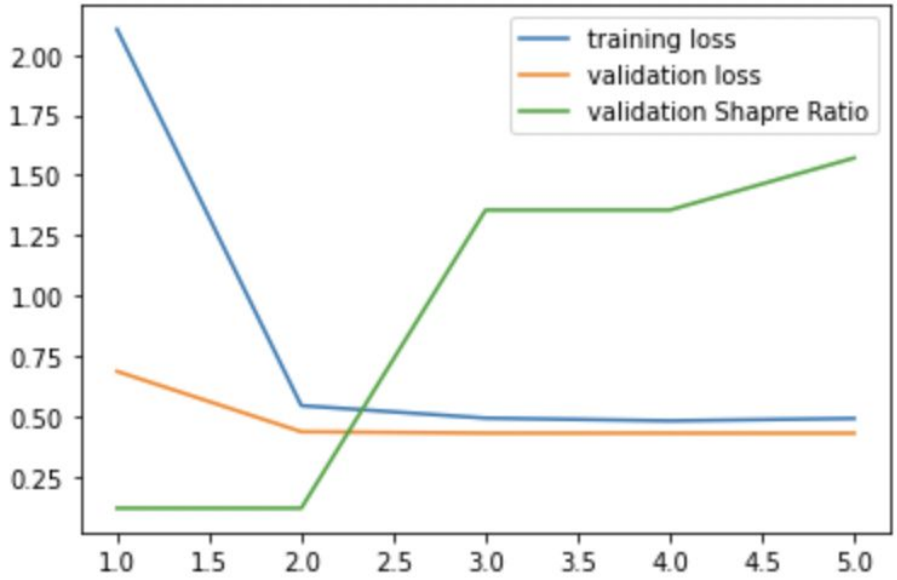
Model Architectures

(3) Attention Model (encoder + decoder)

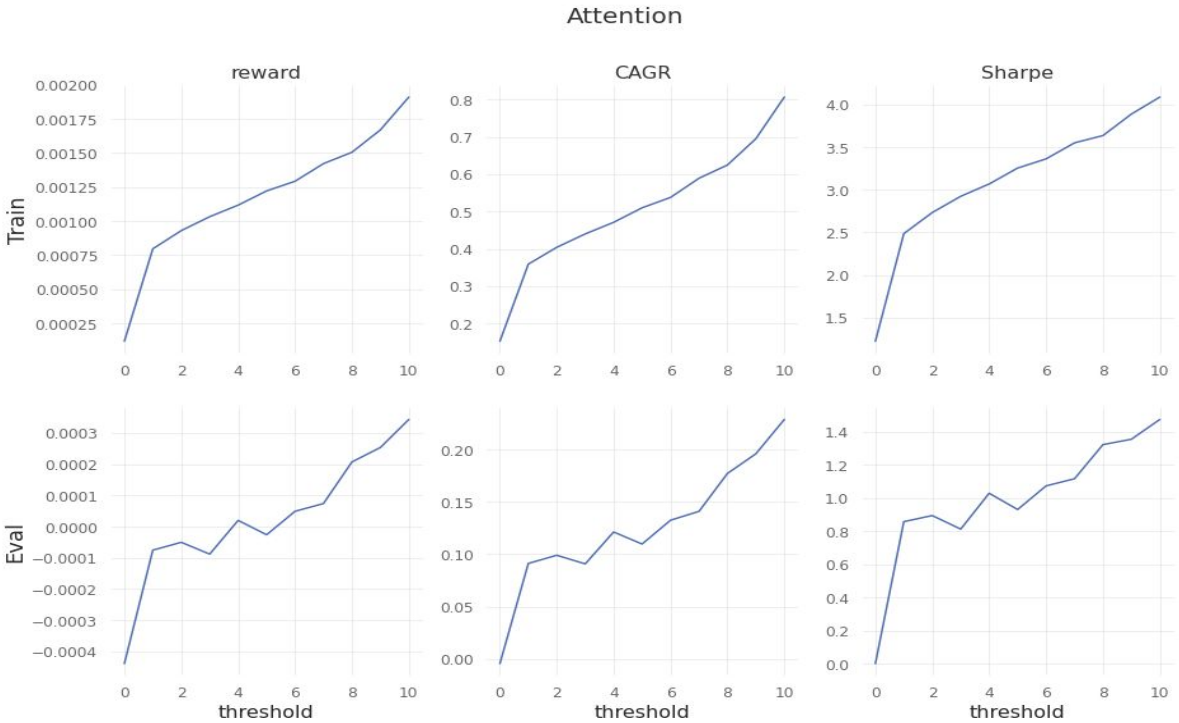
```
DailyModel(
  (encoder): Encoder(
    (linear_in): Sequential(
      (0): Linear(in_features=35, out_features=128, bias=True)
      (1): Dropout(p=0.2, inplace=False)
      (2): LeakyReLU(negative_slope=0.01)
      (3): Linear(in_features=128, out_features=256, bias=True)
      (4): Dropout(p=0.2, inplace=False)
      (5): LeakyReLU(negative_slope=0.01)
    )
    (lstm): LSTM(256, 512, num_layers=2, batch_first=True, bidirectional=True)
    (pBLSTMs): Sequential(
      (0): pBLSTM(
        (blstm): LSTM(1024, 256, num_layers=2, batch_first=True, dropout=0.5, bidirectional=True)
      )
    )
    (key_network): Linear(in_features=1024, out_features=128, bias=True)
    (value_network): Linear(in_features=1024, out_features=128, bias=True)
  )
  (decoder): Decoder(
    (lstm1): LSTMCell(129, 512)
    (lstm2): LSTMCell(512, 128)
    (attention): Attention()
    (linear_out): Linear(in_features=256, out_features=1, bias=True)
  )
)
```



Training-validation curve for a single model for stock “AMM”:



Performance



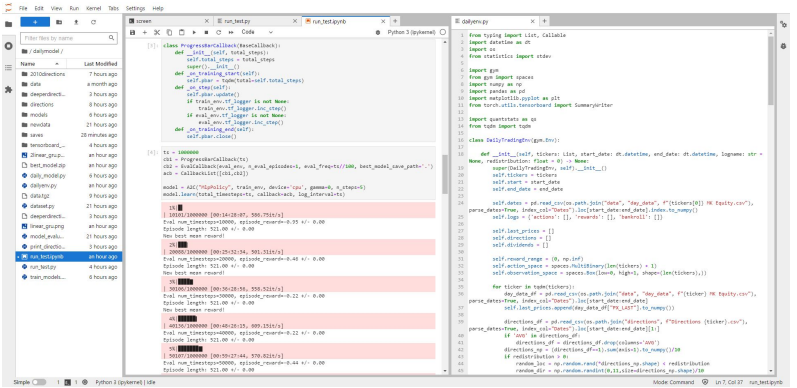
Part 2: RL strategizing

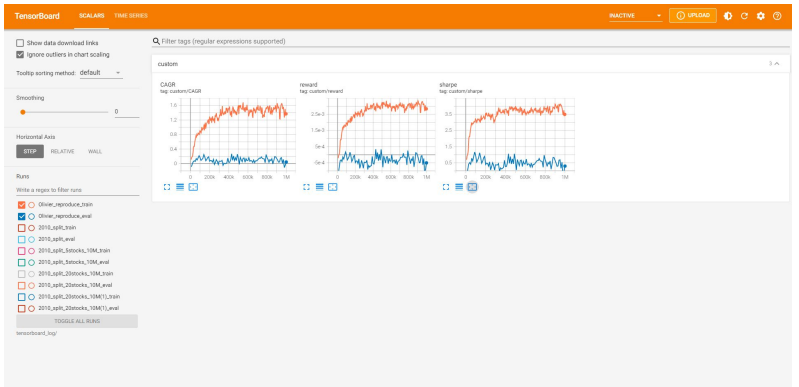
Testbench (dailyenv.py + run_test.ipynb):

Able to easily experiment with different
- Predictions
- Stocks
- Baseline strategies
- RL algorithms
- Hyperparameters
-

Able to produce
- Training monitor graph
- Reward graph comparison
- Quantstats report
-

Limitations:
- Currently assumes no transaction cost





Baseline (non-RL) strategies

Validation (2018-01-01 to 2020-01-01) results on large portfolio

Strategy	Sharpe	Max drawdown	Volatility (ann.)	CAGR %
Buy & hold	0.03	-11.04%	9.81%	-0.16%
Momentum	0.41	-10.49%	10.58%	3.95%
Mean reversion	-0.02	-22.43%	15.42%	-1.59%
RSI 30/70	0.79	-7.95%	9.44%	7.53%
Just listening to average model	0.93	-10.35%	10.84%	10.39%
Invest proportional to # of model that say up	1.35	-10.03%	10.72%	15.53%
Just listening to consensus model	1.49	-16.12%	14.43%	23.68%
Need 8 models that say up	1.28	-8.70%	11.72%	16.02%

Baseline (non-RL) strategies

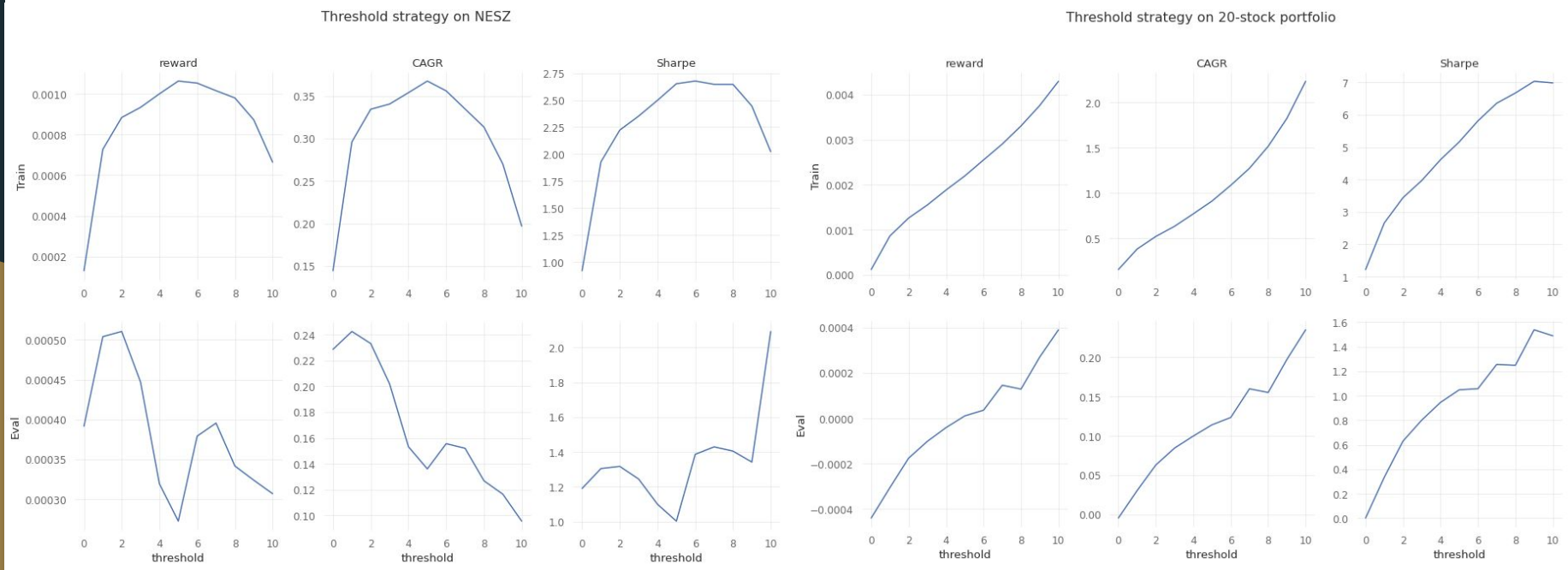
Based on the previous results, we decided to use a “threshold” baseline strategy.

- If n models say that the stock will rise, buy the stock
- n can range from 0 models (always buy and hold) to all models (only buy if 100% sure it will rise)
- Higher n = safer buys? Lower n = riskier buys but with potentially more return?

In general when there are more stocks, it is better to buy only when fully confident

When there are less stocks, it could help to gamble some more

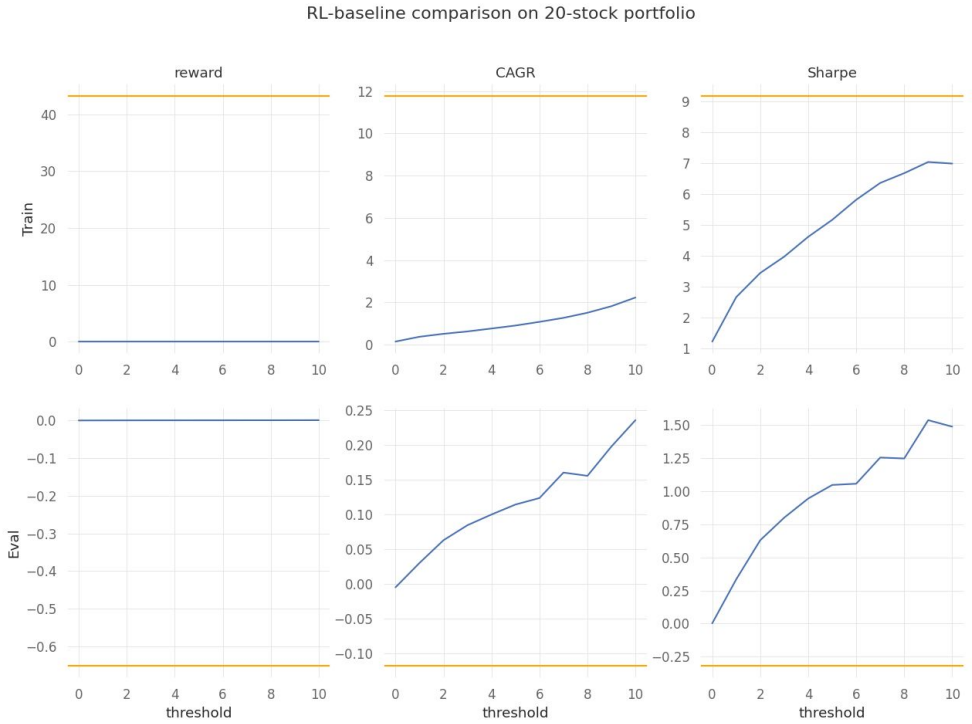
Threshold strategy



Basic RL

Tested one basic algorithm: A2C (Advantage Actor Critic) to find a strategy to trade based on predictions of whether the stock will go up or down

- 20-stock portfolio
- state space is fraction of models predicting buy for each of the 20 stocks
- action is 20 binary decisions whether to buy each stock
- train for 5M steps,
- default hyperparameter settings from RL package “stable-baselines3”

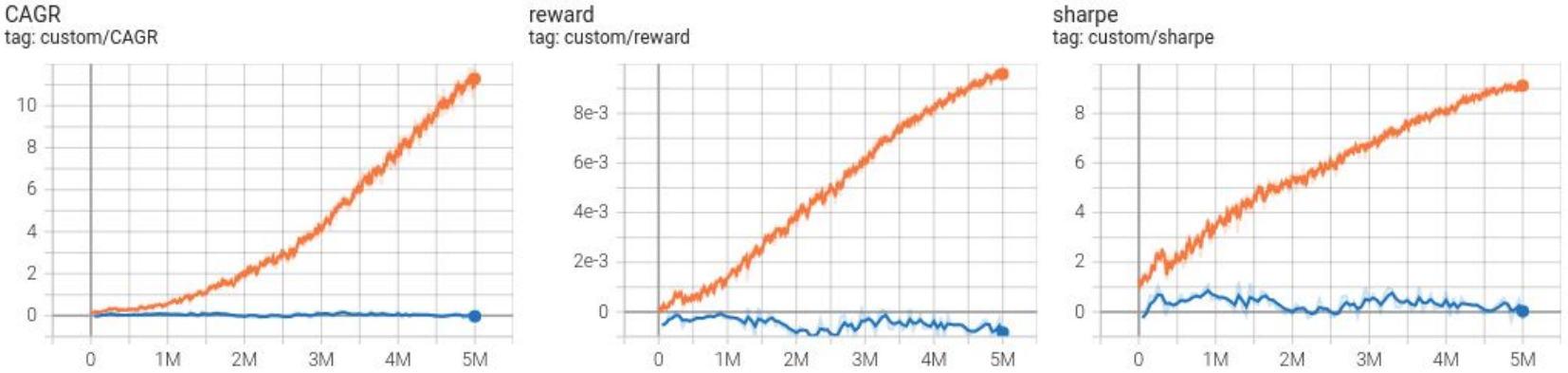


Train-eval discrepancy (overfitting?)

RL Algorithm more than overfits training set (seen data), but bombs evaluation set (unseen data)

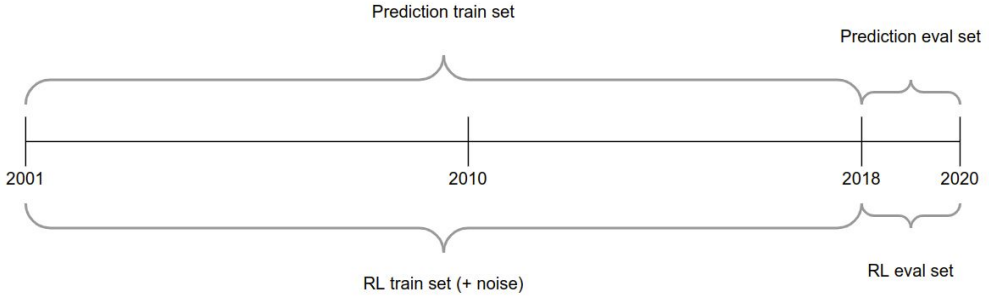
2 main suspected reasons:

- RL learned to trade on accurate price predictions (training set) but not noisy predictions (eval set)
- RL is simply overfitting

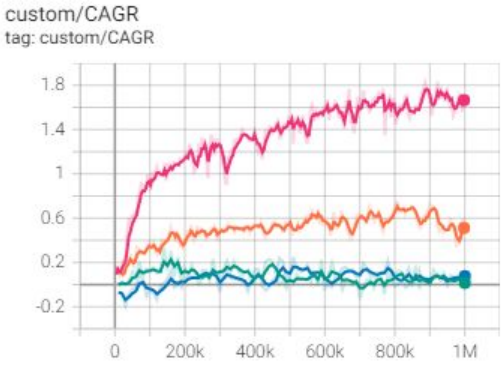


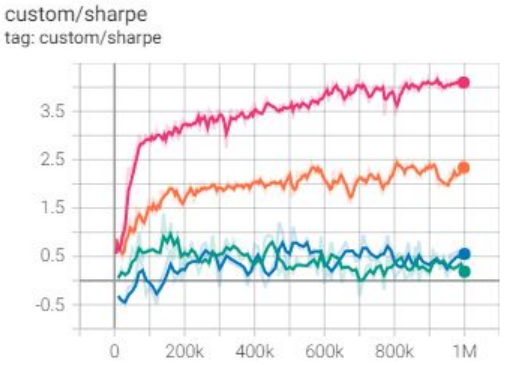
Attempts to fix

1. Add noise into training data so RL learns from inaccurate predictions



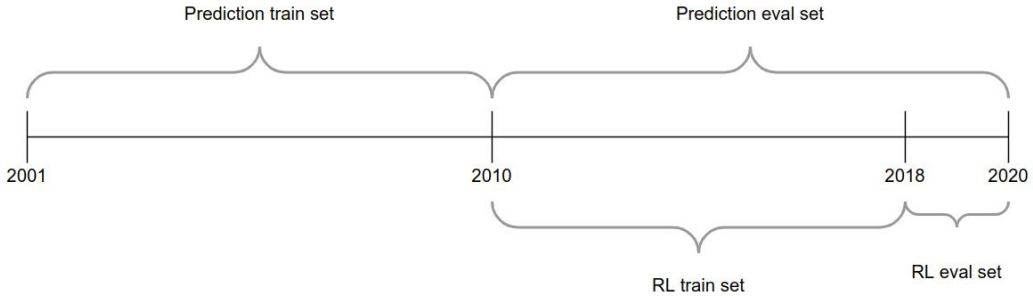
- Original train
- New train
- Original eval
- New eval



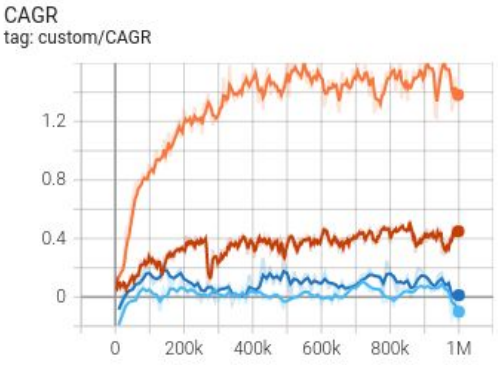


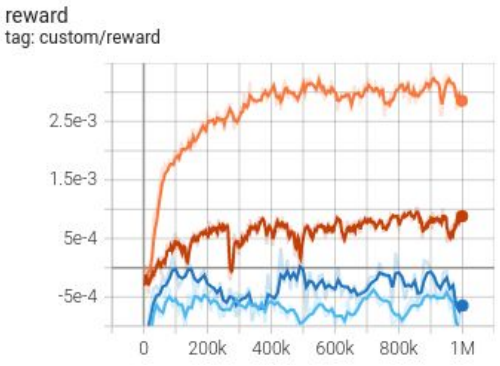
Attempts to fix

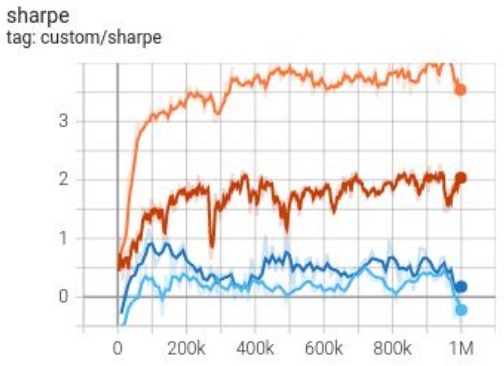
2. Train RL on data unseen to the prediction model



- Original train
- New train
- New eval
- Original eval







Conclusion

- It is possible (and easy) to profitably trade under no transaction costs
- RL is clearly capable of representing successful strategies, but struggle to learn them under noisy environments
    - Currently baseline strategies still outperform RL on evaluation set
- More work is needed to make this simple scenario work before adding in transaction costs

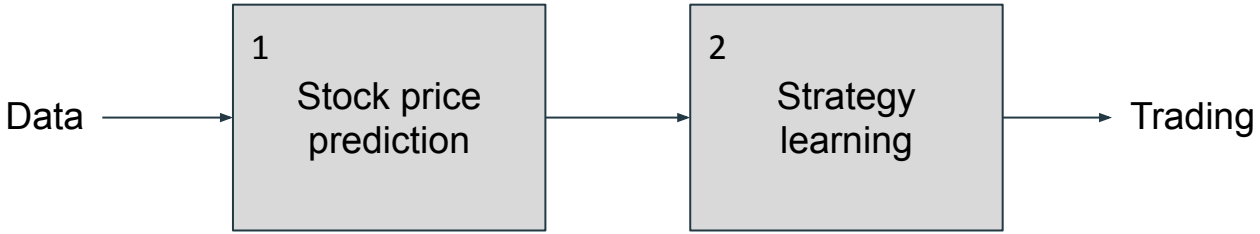
Project Update
2022/6/16

Jeff Schneider
Shaofeng Qin
Bo Wu

Project Update (Summer Conclusion) 2022/8/18

Jeff Schneider
Shaofeng Qin
Bo Wu

Two step algorithm



1. Supervised Learning to predict the next day's closing price
2. Reinforcement Learning to use the predictions to optimally trade

Second Phase

Target:

For our prediction model, we sticked with our attention model, and optimized over architectural parameters, including prediction range, LSTM layers, and non-architectural parameters such as learning rate, training epochs, gamma(learning rate decay rate)

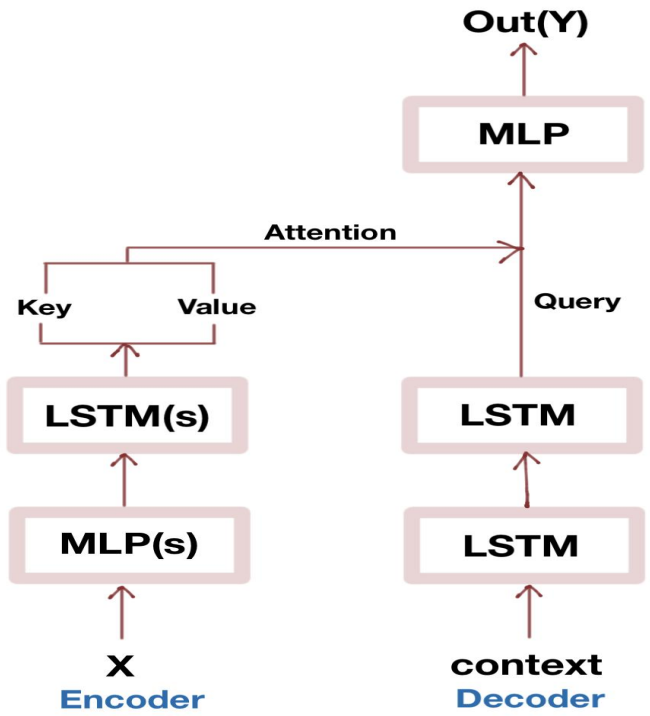
Evaluation metrics: using baseline threshold strategy(thresholds on 0 to 10 models' agreements) —> Theoretically, the more accurate(lower MSE) of our prediction model is, higher consensus and higher CAGR and sharpe ratio will be.

Reference

Model architecture is modification from:

William Chan, Navdeep Jaitly, Quoc V. Le, Oriol Vinyals. “Listen, Attend, Spell”. 2015.

https://arxiv.org/abs/1508.01211#:~:text=Download%3A-,PDF,-Other%



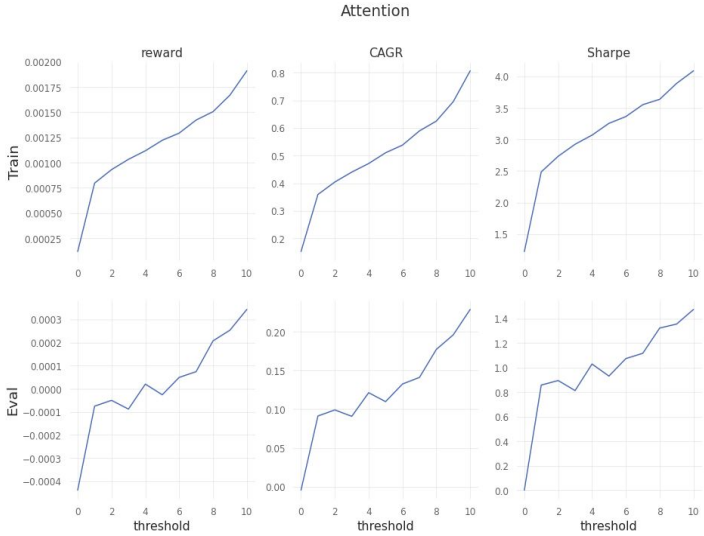
Architecture Tuning (prediction range)

At i th day, we are using past 30 days as our X(input) and make prediction on stock price at [i - prediction range + 1, i+1] day, and using only i+1 th day as “real prediction” to generate directions (stock price goes up: + 1, goes down: -1) .

We do this in order to implement a “self learning mechanism” of attention model, which tries to capture recent days’ features in stronger manner such that strengthen their correlation with current today’s price.

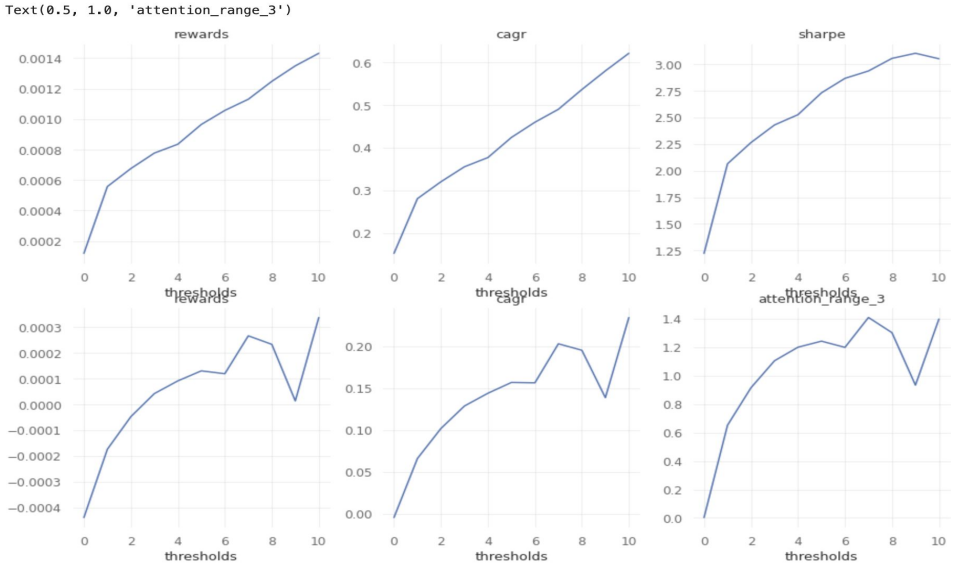
Architecture Tuning(prediction range)

Default = 1, Other default parameters: lr = 0.001, gamma = 0.3, epochs = 5, lstm layers in encoder = 2



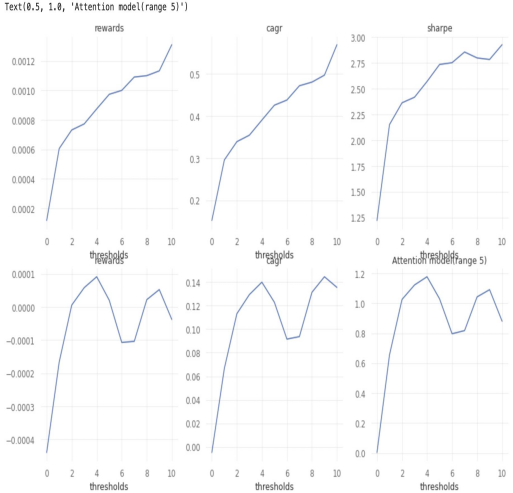
Architecture Tuning(prediction range)

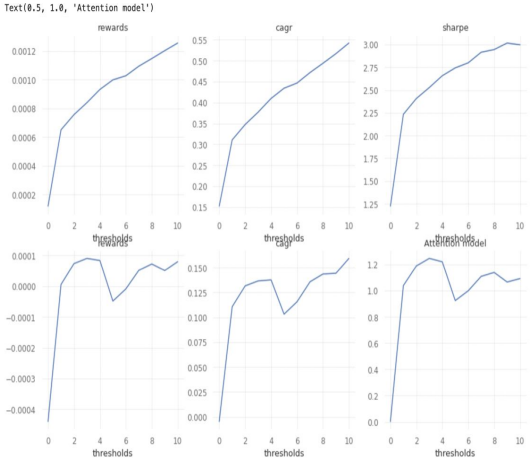
Prediction range = 3

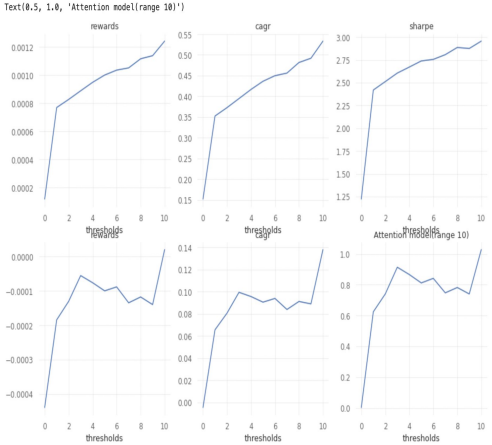


Architecture Tuning(prediction range)

Prediction range = 5, 7, 10

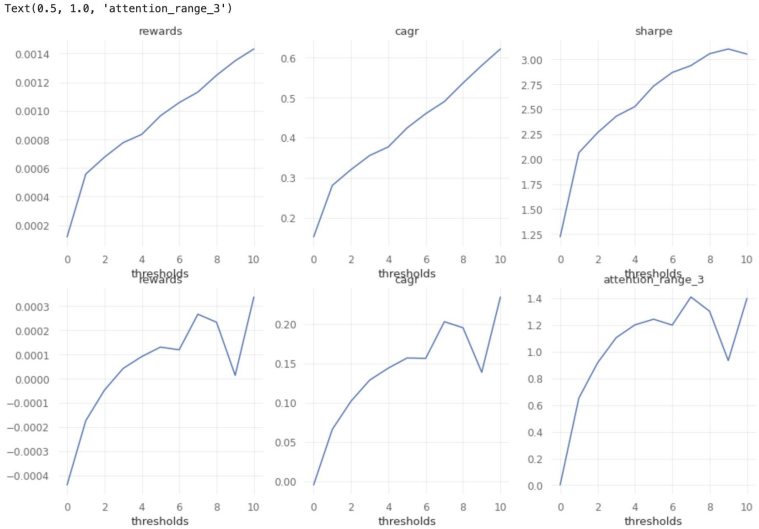




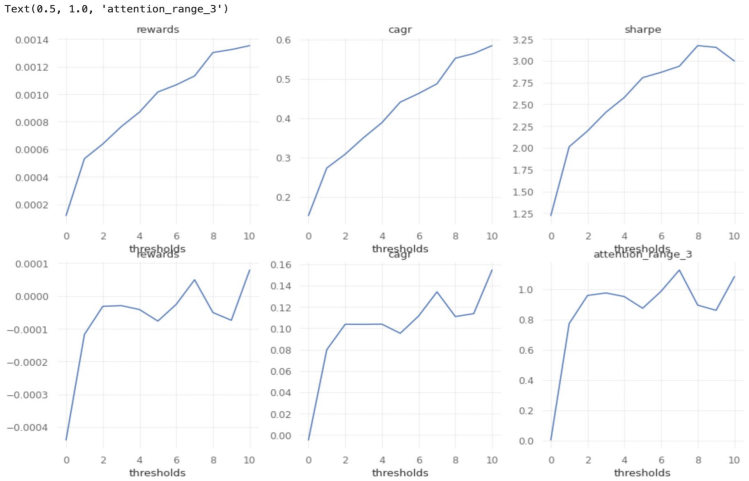


Architecture Tuning(LSTM layers)

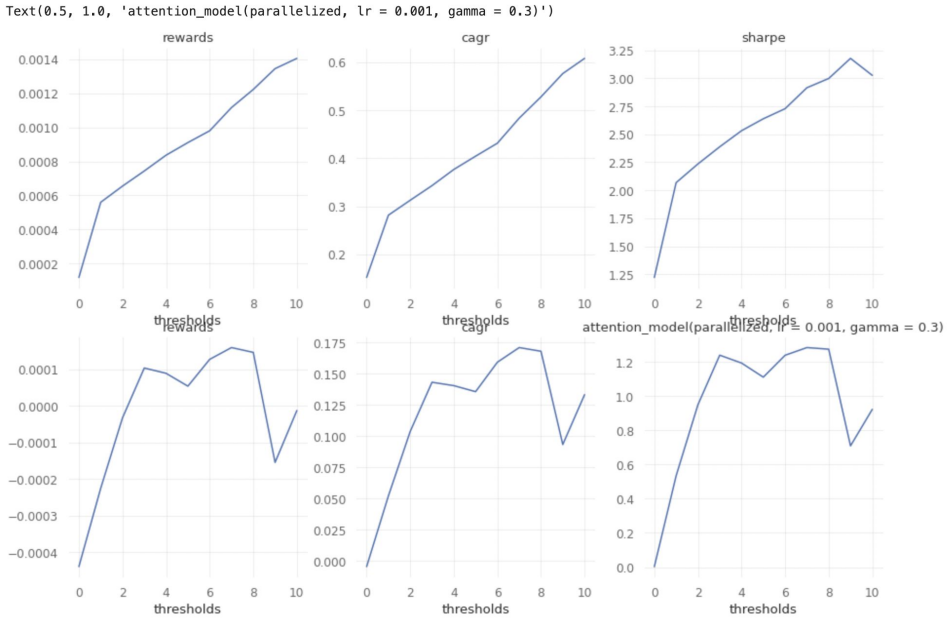
Default: 2



# of lstm layers = 1



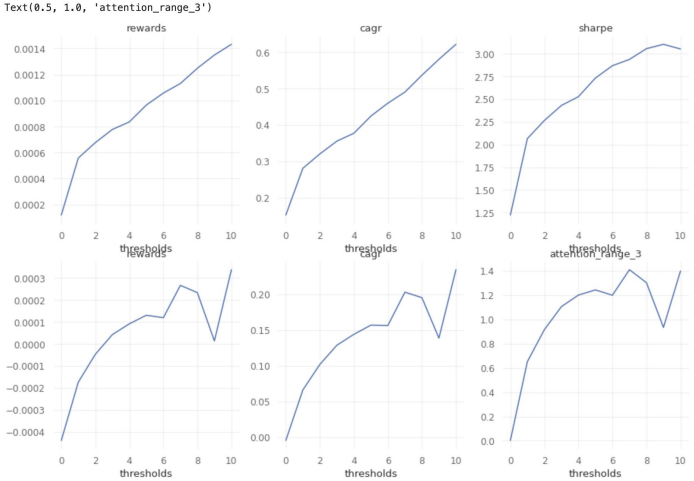
Architecture Tuning(LSTM layers)



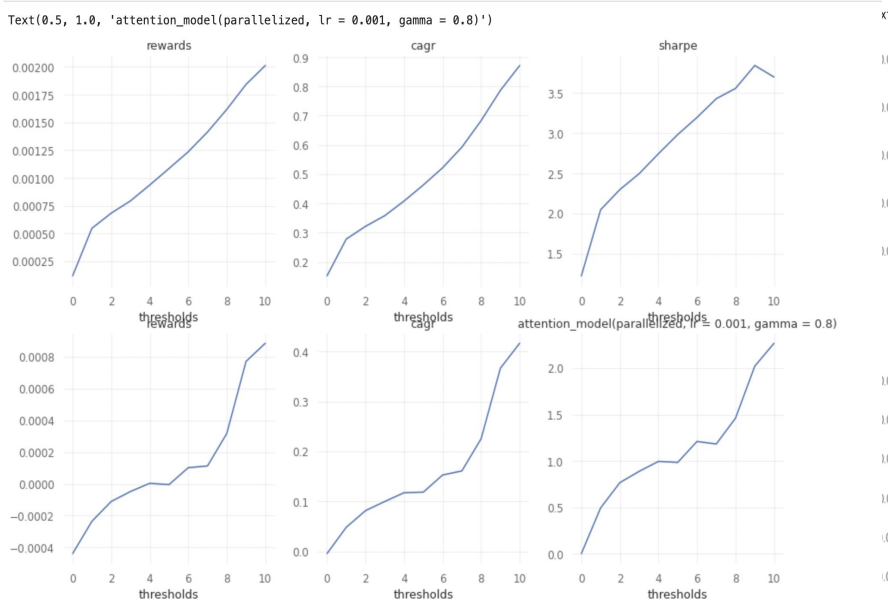
# of lstm layers = 4
With other default parameters

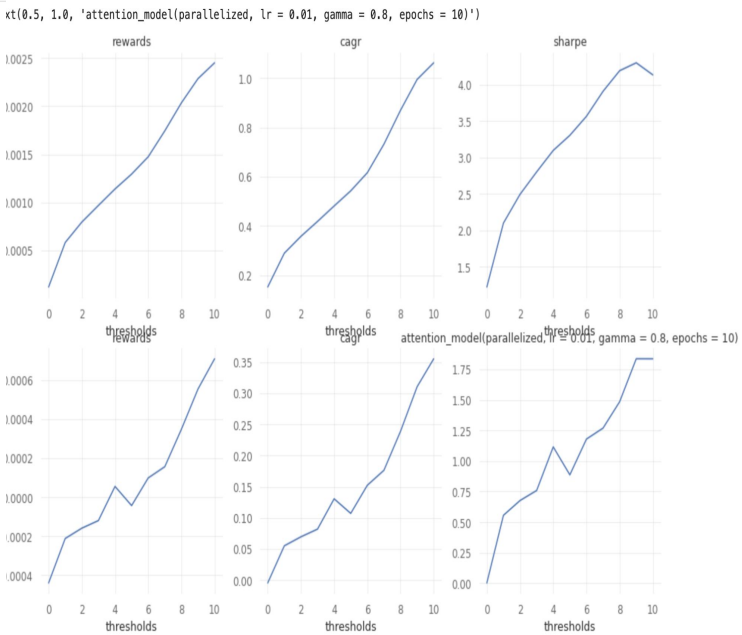
Architecture Tuning(lr & gamma)

Default: lr = 0.001, gamma = 0.3, epochs = 5, range = 3

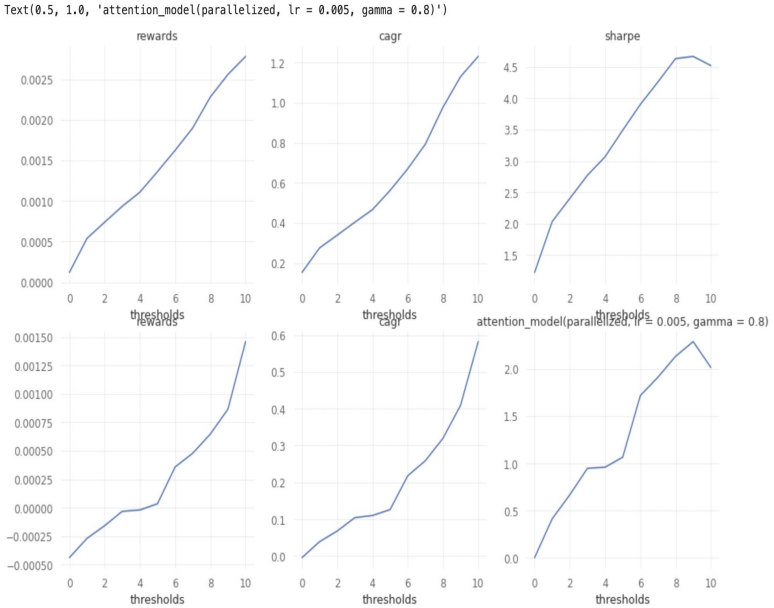


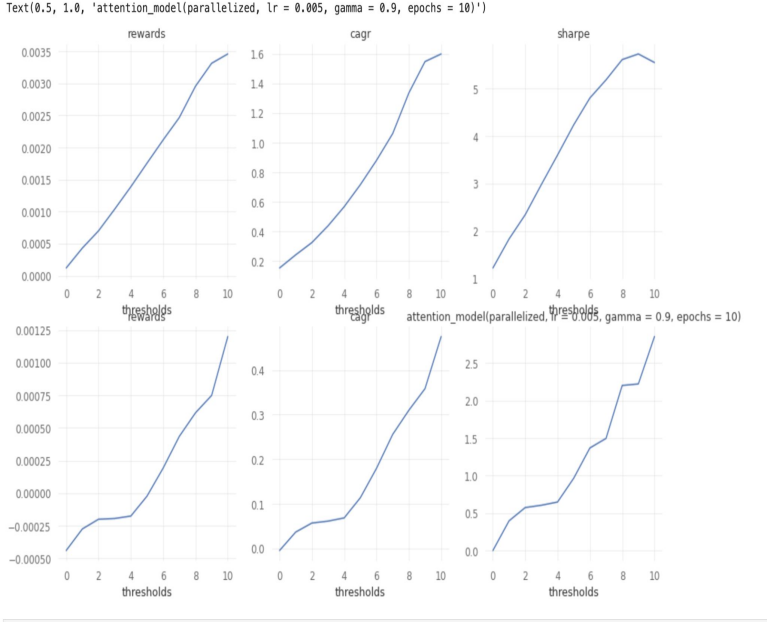
Architecture Tuning(lr & gamma)



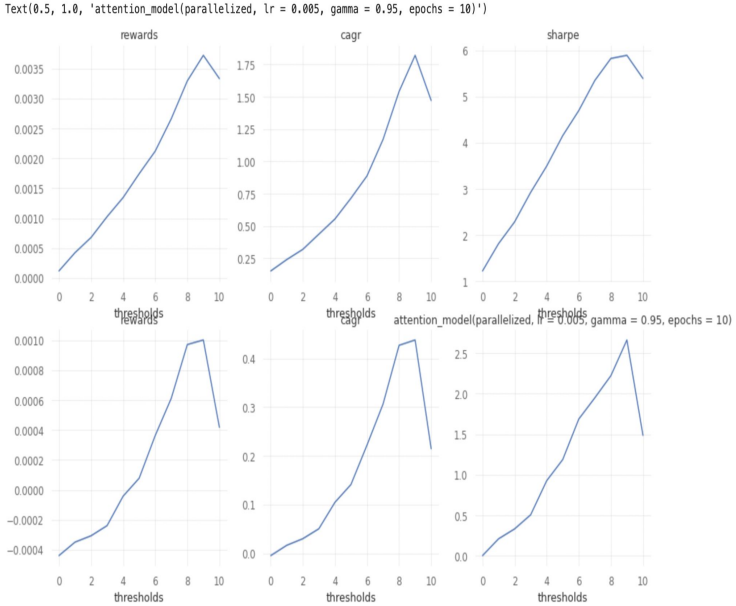


Architecture Tuning(lr & gamma)

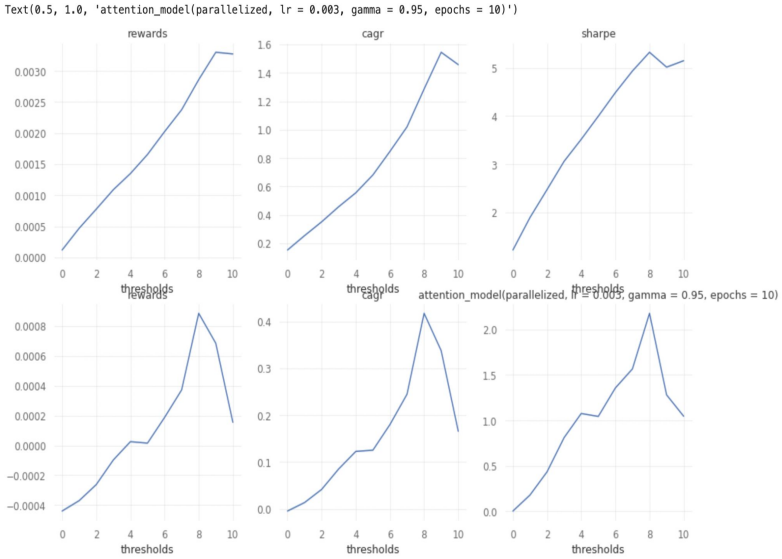




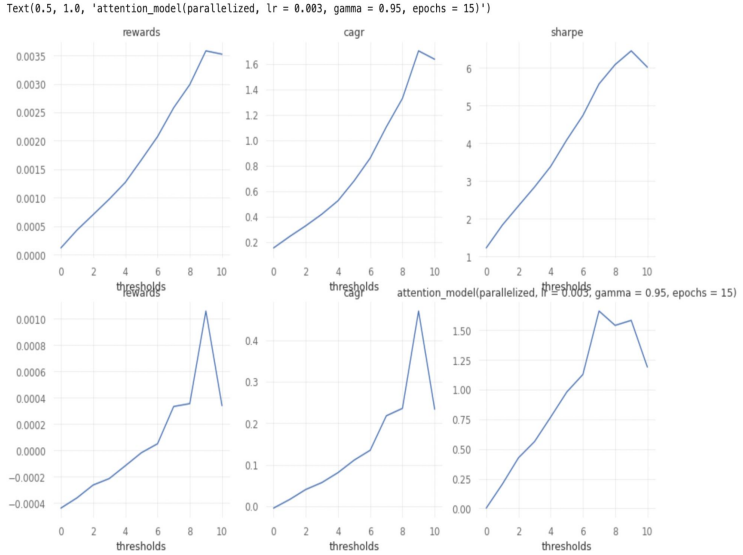
Architecture Tuning(lr & gamma)



Architecture Tuning(lr & gamma)



OVERFITTING

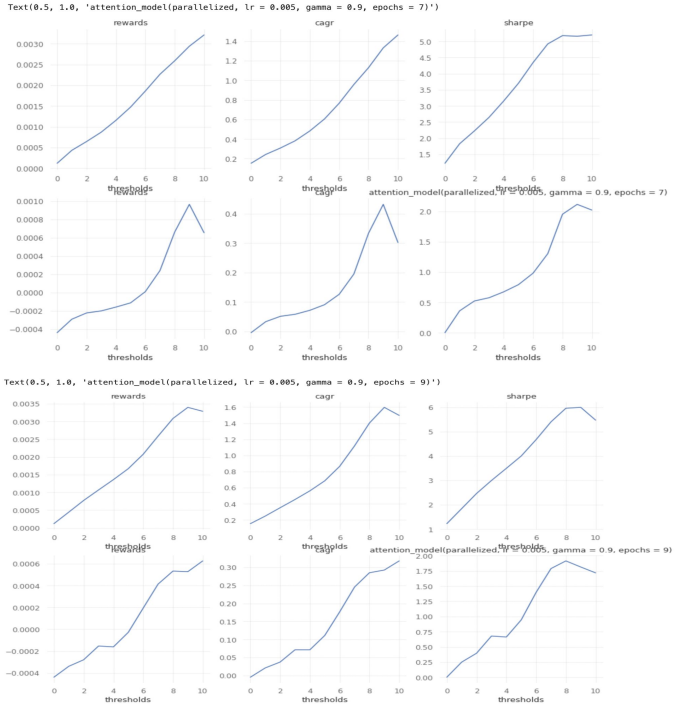


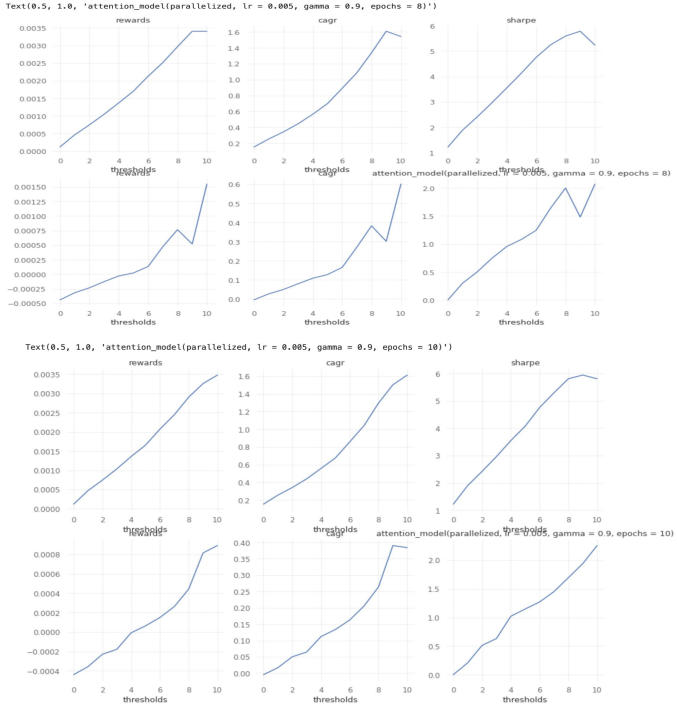
Architecture Tuning(lr & gamma)

Conclusion so far:

Best combination so far is: lstm = 2, epochs = 10, gamma = 0.9, lr = 0.005

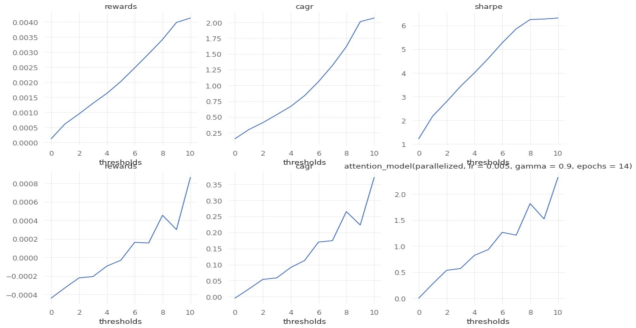
Architecture Tuning(epochs)





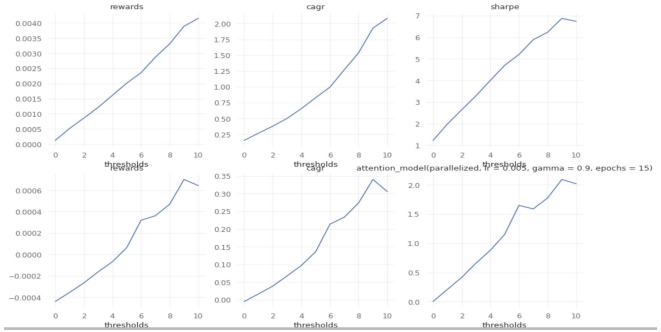
Architecture Tuning(epochs)

Text(0.5, 1.0, 'attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 14)')



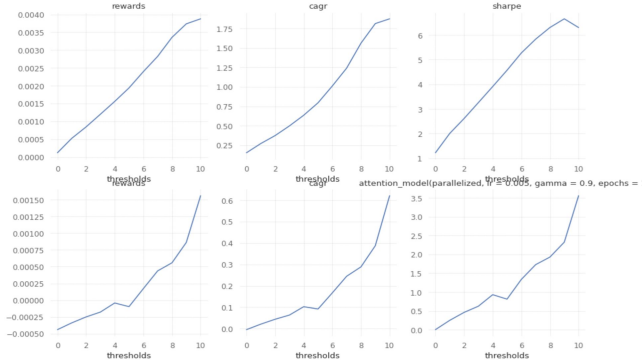
attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 14)

Text(0.5, 1.0, 'attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 15)')



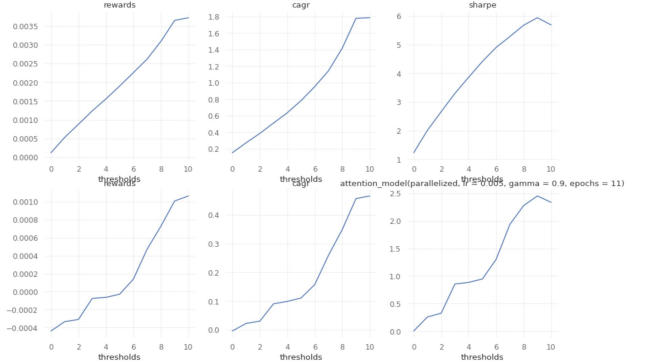
attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 15)

Text(0.5, 1.0, 'attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 13)')



attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 13)

Text(0.5, 1.0, 'attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 11)')



attention_model(parallelized, lr = 0.005, gamma = 0.9, epochs = 11)

Final Conclusion (prediction model)

Default: lstm = 2, lr = 0.001, epochs = 5, gamma = 0.3, epochs = 5→ Validation dataset sharpe ratio: about 1.4

Best combination so far is: lstm = 2, epochs = 13, gamma = 0.9, lr = 0.005 → Validation dataset sharpe ratio: about 3.5

Other:

lstm = 2, epochs = 14, gamma = 0.9, lr = 0.005 → Validation dataset sharpe ratio: about 2.3

lstm = 2, epochs = 11, gamma = 0.9, lr = 0.005 → Validation dataset sharpe ratio: about 2.4

Future Works

Future Works:

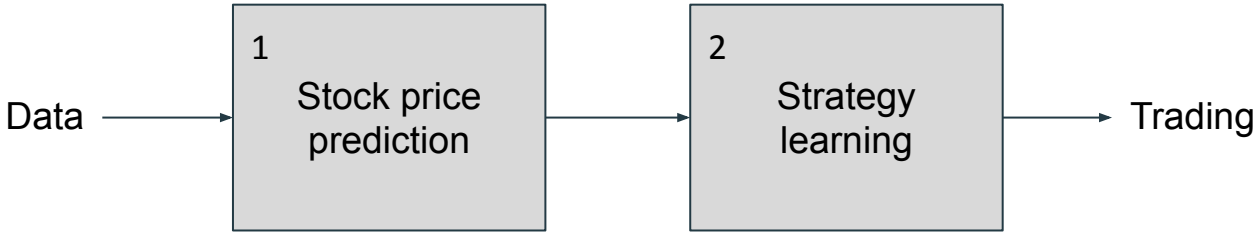
(1) increase wider prediction range on “future days”: now = 1 day —> extend to at most days

(2) ensemble models to gain an averaging performance → decrease randomness of any single model

How to run the code

1. Download the code
   a. https://github.com/buhiroshi0205/algotrade_research (either clone the repository using git, or download a zip archive from the green “code” tab on the website)
2. Environment
   a. Install a cuda-compatible pytorch distribution (https://pytorch.org/get-started/locally/)
   b. `pip install -r requirements.txt` installs the other necessary libraries.
3. Data folder setup
   a. Keep the folder structure the same, or make sure are located at ../code/
4. Run!
   a. Running the script `main.py` (by executing `python3 main.py` in the terminal) trains models in parallel and automatically generate csv files under the directions folder
   b. In `main.py`:
      i. Modify and add to the hyperparameters through the params dict that is passed to train_with_config
      ii. Variable “process” is how many gpu nodes you want to use, and please make sure that it is corresponding to the number of devices in setting: os.environ["CUDA_VISIBLE_DEVICES"]

Two step algorithm



1. Supervised Learning to predict the next day's closing price
2. Reinforcement Learning to use the predictions to optimally trade

Main Takeaways

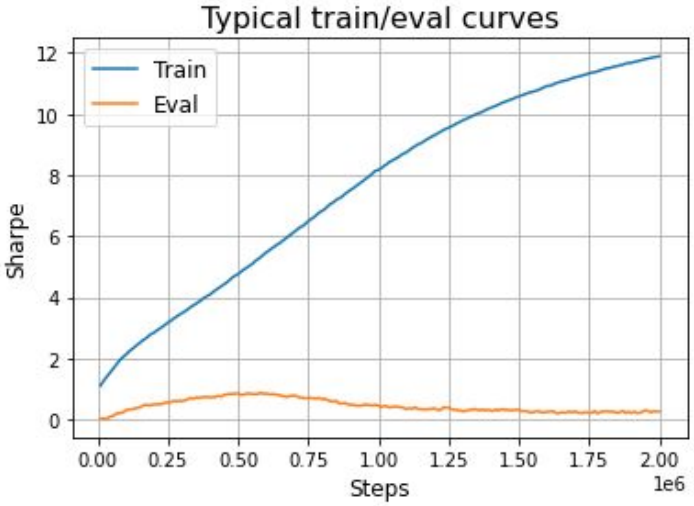
Discovered Problems:

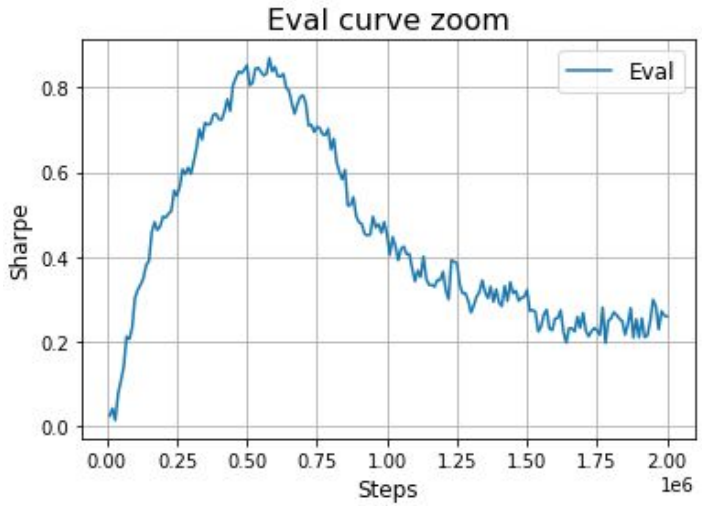
1. There is a significant overfitting issue that can’t be resolved through basic methods
2. The extreme variance throws off most off-the-shelf techniques and algorithms

Current Solutions:

1. Split dataset between prediction and RL parts so that their training sets don’t overlap
2. Aggressive early stopping
3. Use multiple portfolios of small number of stocks
4. Massive parallelization to take average of multiple runs
5. Automatic hyperparameter tuning using even more parallelization

Problem 1: Overfitting





* 20 stocks, PPO, default hparams + 0.003 ent_coef, 2M steps, average of 60 runs

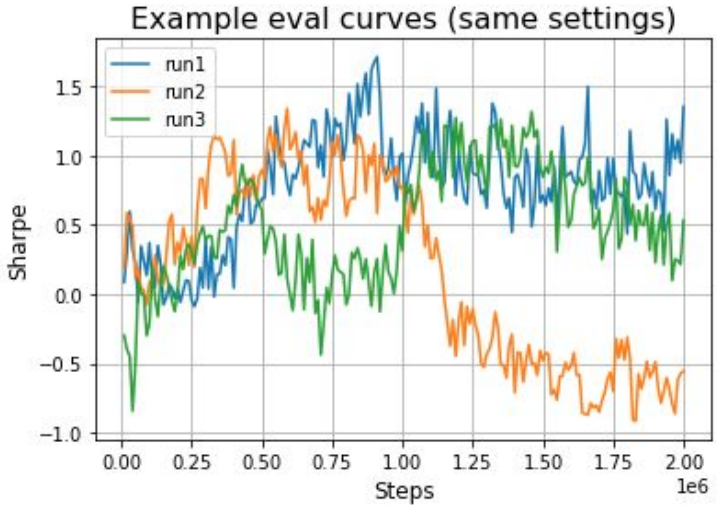
Problem 1: Overfitting

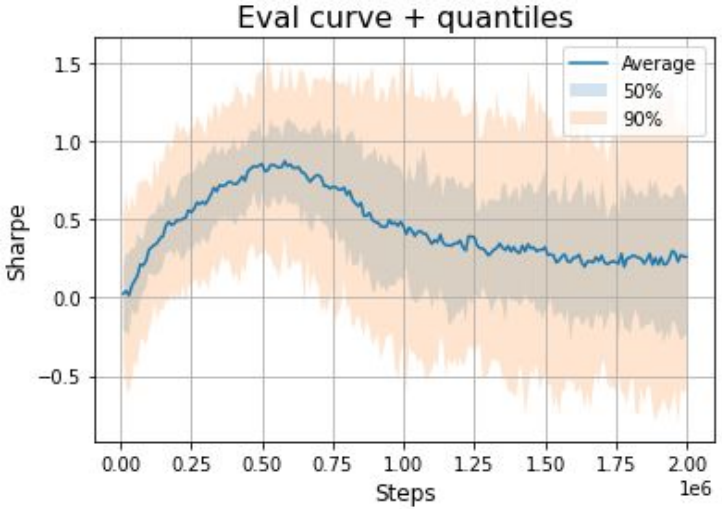
2 suspected root causes:

1. Lack of data
    a. 5625 days of stock market = 5625 datapoints. Even MNIST has 60000 datapoints, for comparison.
    b. RL environments typically have infinite datapoints since the environments can keep generating new data in their astronomical state spaces
2. Noisiness of data
    a. Financial data has very low signal-to-noise ratio, meaning that we need even more data than normal to discern a trend

Renders RL on such data incredibly difficult.

Problem 2: Variance





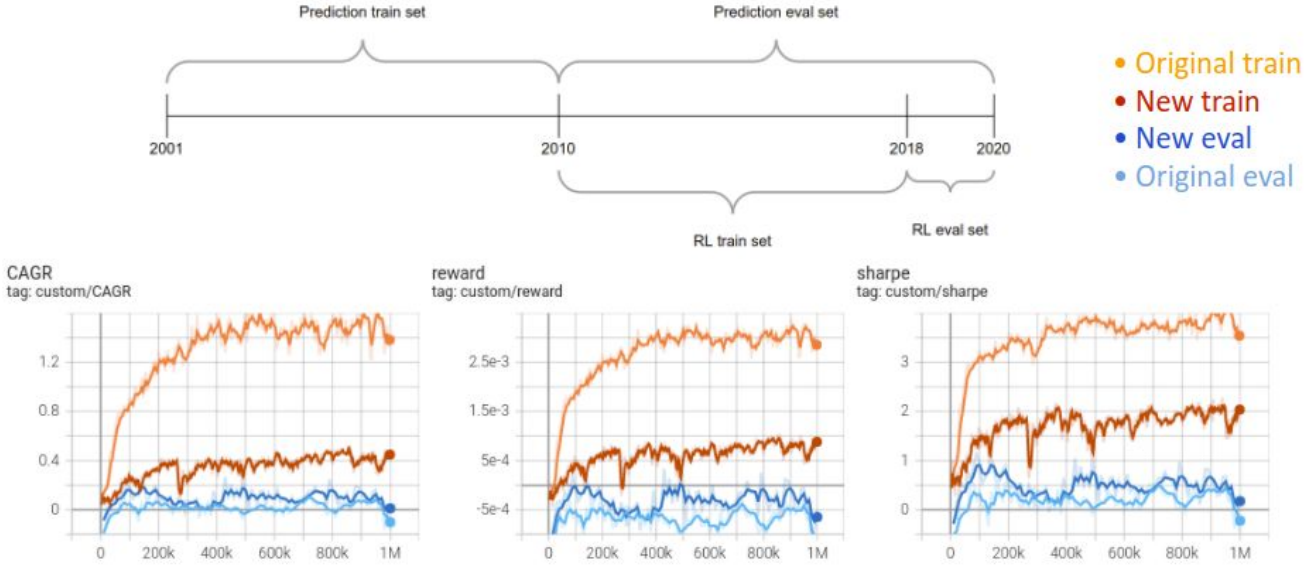
Problem 2: Variance

1. Variance within a single run
    a. Lots of stochasticity within a single training session: the eval curve jumps up and down sporadically and a stable learning behavior cannot be observed
    b. This is using the PPO algorithm, which is supposedly one of the more stable algorithms
    c. A2C experimentally has much worse stochasticity than PPO
2. Run-to-run variance
    a. Same settings, different seed results in completely different looking curves
    b. At the same point in time, two runs can have a sharpe ratio difference of 2
    c. Even when we constrain to middle 50% of the values, it could differ by about ~0.8

Makes interpretation of the runs very difficult

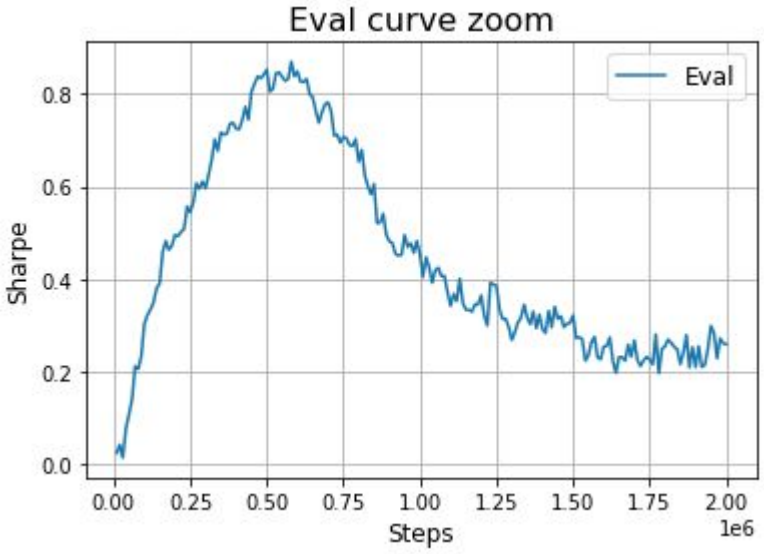
Solution 1: Dataset split

A better split of the dataset leads to lower training curve and higher eval curve, indicating a decrease in overfitting



Solution 2: Early stopping

We take the model where the eval Sharpe is the highest, instead of at the end of the training. This leads to a much better result (0.8 vs 0.3) since we ignore the curve after serious overfitting kicks in.

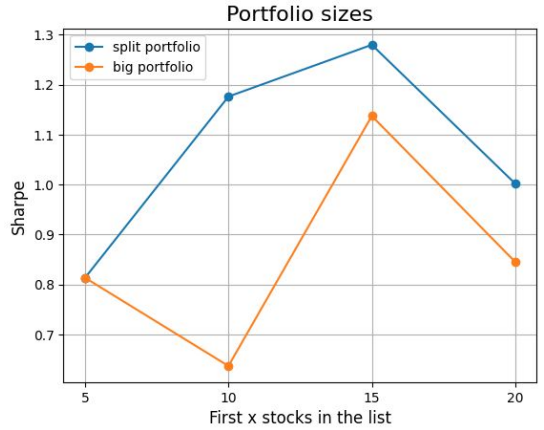


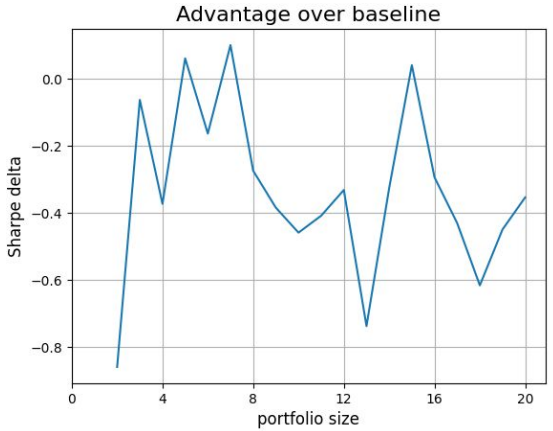
Solution 3: Split portfolios

Most of the times, multiple small portfolios outperform one large portfolio

- E.g. using the same stocks
  - 2 portfolios of 5 outperform 1 portfolio of 10
  - 3 portfolios of 5 outperform 1 portfolio of 15
  - 4 portfolios of 5 outperform 1 portfolio of 20

A small portfolio size of around 5 ~ 7 seems to be a good range to outperform the baseline



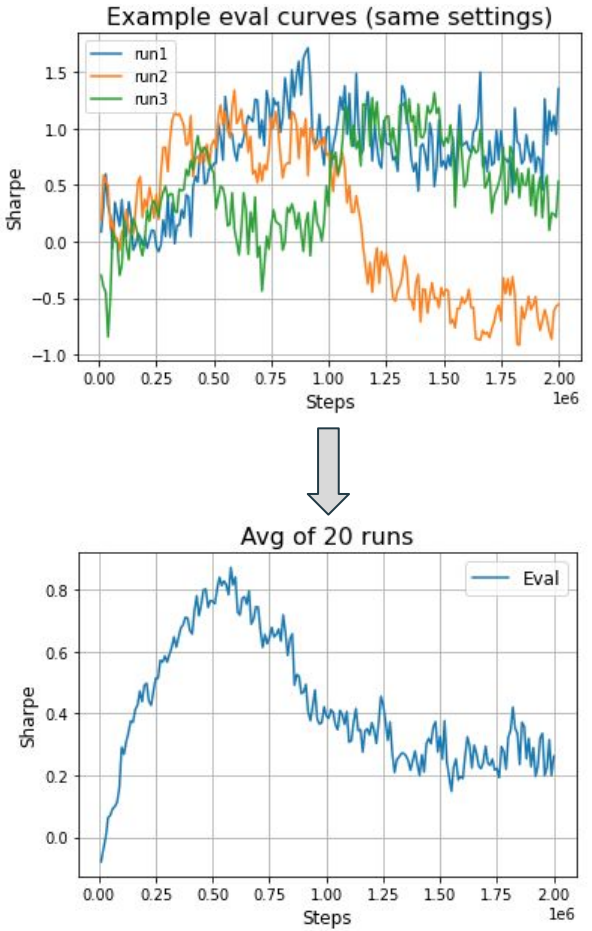


Solution 4: Parallel runs

Averaging the curves of multiple runs shows the general trend of a particular experiment and makes it look like a regular training curve that's easily analyzed.

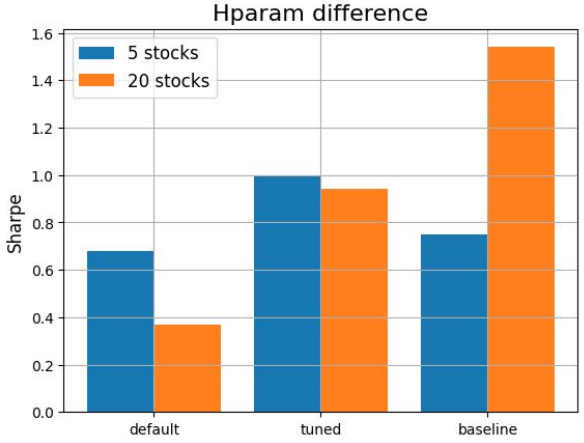
Cons:

- Repeating runs require lots of computing power which is completely infeasible if not done in parallel
- This masks the true stochasticity of a single model, so actually training and using one of these models won't give such good results



Solution 5: AutoML

- Hyperparameters seem to have significant effect on the final result, so tuning is required.
- Manual tuning: 2 weeks => auto tuning: 1 day
- Using the Optuna library for distributed auto hyperparameter tuning across multiple machines
  - Bayesian Optimization
- Cons
  - Current automl algorithms aren't too developed and might yield worse results than manual tuning
  - Automl might record inaccurate data due to high run-to-run variance and tune wrongly



Future work

- More fundamental ways to alleviate the overfitting
  - Stop distinguishing between different stocks and use arbitrary combinations of stocks in arbitrary orders to train the same model? This could increase data size.
  - Use more data than just the closing price of each day
- Better ways to handle the variance
  - How to explicitly quantify the variance?
  - Have prediction ensemble/model output explicit price prediction and confidence interval?
- More realistic problem setup
  - Trading costs
  - Current portfolio allocation in the state space
  - Maybe volume limits?

Running the code

1. Download the code
   a. https://github.com/buhiroshi0205/algotrade_research (either clone the repository using git, or download a zip archive from the green “code” tab on the website)
2. Environment
   a. `pip install -r requirements.txt` installs the necessary libraries.
   b. cpu mode is sufficient for pytorch, but a cuda one works too
3. Data folder setup
   a. Keep the folder structure the same, or make sure directions are located at ../data/directions/{directions_name}
4. Run!
   a. Running the script `train_rl.py` (by executing `python3 train_rl.py` in the terminal) trains models in parallel and writes the result to tensorboard with logdir ../tensorboard_logs/
   b. In `train_rl.py`:
      i. In the main function choose to run one experiment, or run an automl tuning experiment
      ii. Modify and add to the hyperparameters through the params dict that is passed to run()
      iii. n_trials is the number of parallel runs to take average of. For smooth operation, make sure this number is much less than the number of cpu cores available on your machine