IEOR 4575 Project Report

Erica Wei (cw3137)[1]

[1]Department of Computer Science, Columbia University, NYC, USA.

4/22/2021

1 Introduction

This project report includes work for Reinforcement Learning in cutting plane problem, including algorithm design, pseudo-code, experiments and results. The project is basically following the paper, Reinforcement Learning for Integer Programming: Learning to Cut [1] and experiments with different parameters and policy network. I used Policy Gradient Methods (PG) in this project for doing reinforcement learning (RL).

2 Description of the algorithm design

There are several components in this project. The mechanism flow is: constraints [A, b] and cuts [E, d] are two inputs fed into Policy network to get a probability distribution on current available cuts and sample a cut as action based on this probability distribution. Record each step action, state and corresponding reward until finishing this trajectory. I will explain those parts: pre-processing, policy gradient, network, Evaluation Step.

2.1 Pre-processing

For each state, we have A, b as current constraints, and E, d as available cuts to add. The first step is to normalize A, E and normalize b d. After that, we concatenate constraints [A, b] and cuts [E, d] so that they can feed into polciy actor.

2.2 Policy Gradient

Policy Gradient Methods (PG) are frequently used algorithms in reinforcement learning (RL) and being proved to be effective in many problems. For PG, we train a policy to act based on observations. The training in PG makes actions with high rewards and also update policy after every episode.
After each trajectory, the discounted reward being calculated and used for policy gradient step to update parameters in the policy network.
Figure 1 shows the work flow of our policy actor.

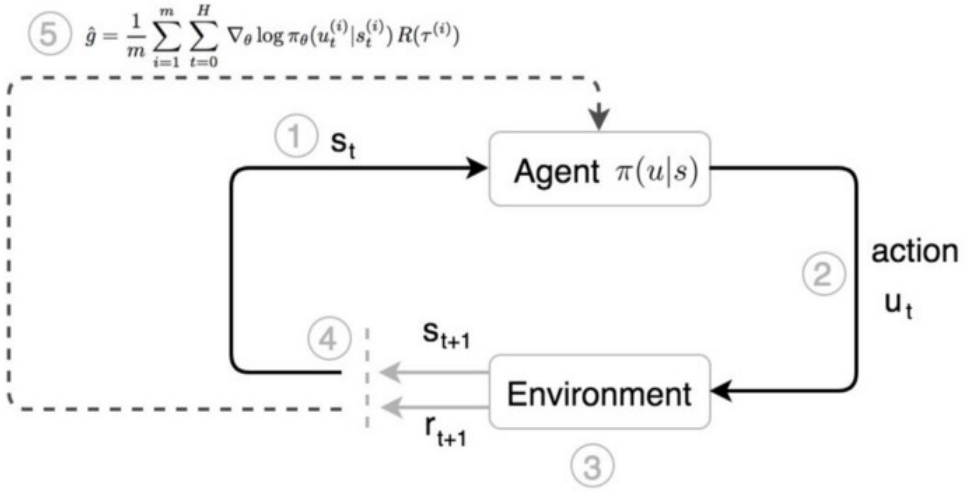

Figure 1: Policy Gradient Descent [2]

2.3 LSTM and Attention

The policy network includes LSTM network, dense linear network, and then attention network. Note: I used the policy template directly from lab4.

After getting inputs, we first use the LSTM network to do an embedding. This is just simple LSTM with one layer and single direction. The reason we choose the simple LSTM is the problem is not considered complicated in terms of dimension and not like NLP task, the direction doesn't matter in this problem. Although I tried bi-directional as well but it doesn't improve a lot on performance. So I keep it simple as possible. During the experiment, the hidden dimension in LSTM, called H1, is adjusted and tuned for this project. The output are two embedded vectors: $h_1 = LSTM([A, b])$ and $h_2 = LSTM([E, d])$.

After LSTM, we feed the two embedded vectors into dense linear network (2 layers with tanh activation) and then do an attention: $\hat{h_1} = F(h_1)$ and $\hat{h_2} = F([h_2)$, the attention is calculated as $score = \frac{1}{N}\sum \hat{h_2}^T \cdot \hat{h_1}$

Then the probability distribution will just be taking softmax of the scores.

2.4 Exploration

The exploration is tricky here, if we used small exploration rate at firs iteration then the policy couldn't learn very well. So I used 0.5 as beginning exploration rate and exploration rate will decay every iteration. Another point I found during the experiments is that if we take argmax action from probability distribution, the model is easy to overfit in on instance and never getting improved. So instead of choosing action with max probability when we don't explore, I did sampling action based on probability distribution.

2.5 ES

For Evolution Strategy(ES), I just followed what paper did: After getting dicounted reward

$$J_{t+1} = \gamma^t \cdot r_t$$

We add random factor here

$$E_{t+1} = J_{t+1} \cdot \frac{\epsilon}{\sigma}$$

where ϵ is a random number generated from Gaussian distribution $N(0, 1)$.

3 Pseudo-code

Here's the pseudo-code for the model I used in my project.

Algorithm 1: Project model algorithm

Result: Trained Policy
initialize Policy model, initialize parameters;
for *every iteration e* **do**
  Rollouts
  initialize state s_0;
  while *d is not True* **do**
    pre-processing A, b, E, d from current state s_t;
    compute probability distribution for action a_t based on current policy;
    if *explore* **then**
      randomly choose action a_t from available cuts;
    else
      sample action a_t based on probability distribution;
    end
    Record sate s_t, action a_t, reward r_t;
  end
  Training policy
  Compute discounted rewards $J_{t+1} = r_t \gamma^t$;
  Compute Evolutionary Strategies(ES) $E_t = J_t \frac{\epsilon}{\sigma}$ by discounted rewards;
  Compute loss $= -\frac{1}{T} \sum_i^T E_t \cdot \log(\pi(a_i|s_i))$ for Policy network;
  Update parameters in Policy network;
end

Table 1: Experiments

Model Number	LSTM hidden dim	Dense hidden dim	σ	exploration
1	16	32	5	No
2	16	32	5	Yes
3	10	64	0.2	No
4	10	64	0.2	Yes

4 Experiments Setup

The different models and parameters I tried in this experiment are shown in below table 1, I only showed several settings I think meaningful here. For exploration, it doesn't improve too much with it or not. But overall performance(training reward moving average) getting more stable.

5 Results

Since there are hundreds of runs on wandb with different parameters tuning, it's impossible to show all of results here. So I only choose 4 models in the table 1 to compare results:

For testing case, we saved best model during the training process and use saved policy to act in testing environment. This could be overfitting in training environment but this is best we can do for now as we don't have validation environment.

6 Conclusion

Overall the models have good performance in training process. Although it could be overfitting in some case since I didn't use validation here, but it still got good performance in test configure. Model 1 and 2 with simpler hidden dimension works better in easy configure and Model 3 and 4 from paper works better for hard-configure. This project shows that the model from paper works well in cutting plane problem and also models get more stable after adding exploration step.

Acknowledgments

Thanks for the help from TA Yunhao Tang, Abhi Gupta and professor Shipra Agrawal in this project. The paper I mainly used in the project is [1]

References

[1] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," *CoRR*, vol. abs/1906.04859, 2019. arXiv: `1906.04859`. [Online]. Available: `http://arxiv.org/abs/1906.04859`.

[2] J. Hui, "Policy gradient picture credited,"

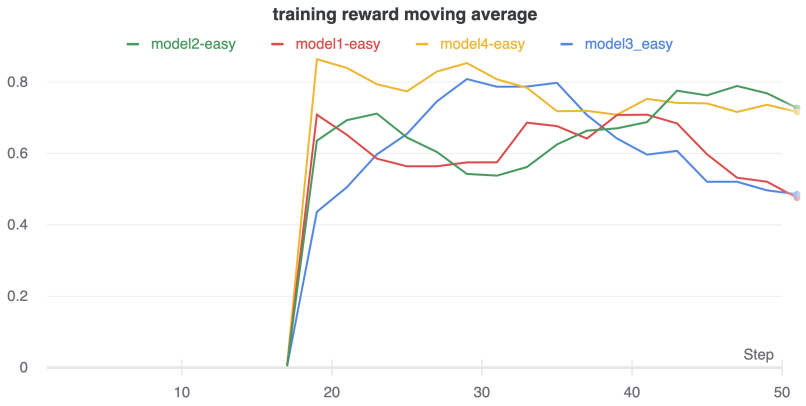

Figure 2: Models compared in easy-configure with moving average

Figure 3: Models compared in easy-configure with rewards

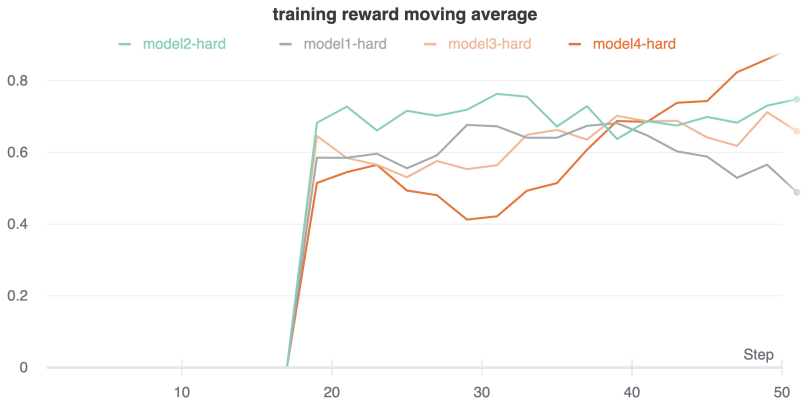

Figure 4: Models compared in hard-configure with moving average

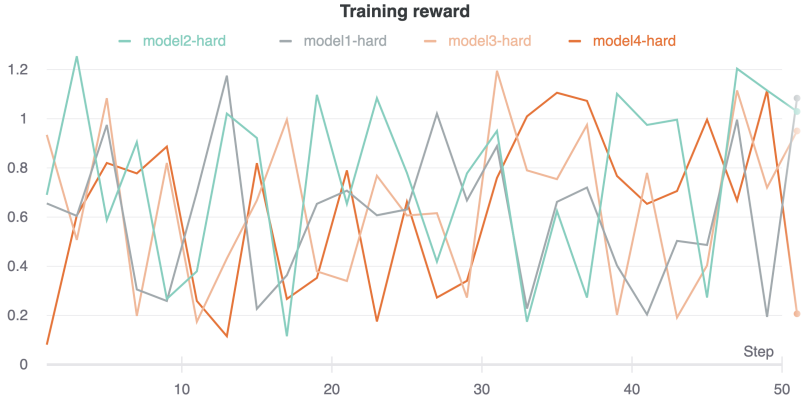

Figure 5: Models compared in hard-configure with reward

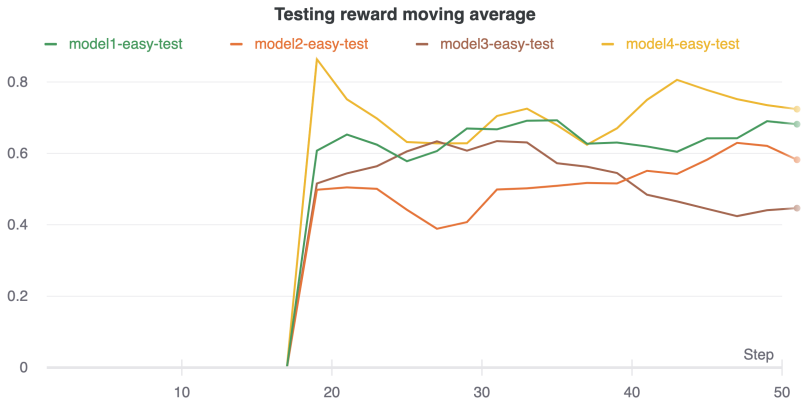

Figure 6: testing moving average by models in easy configure

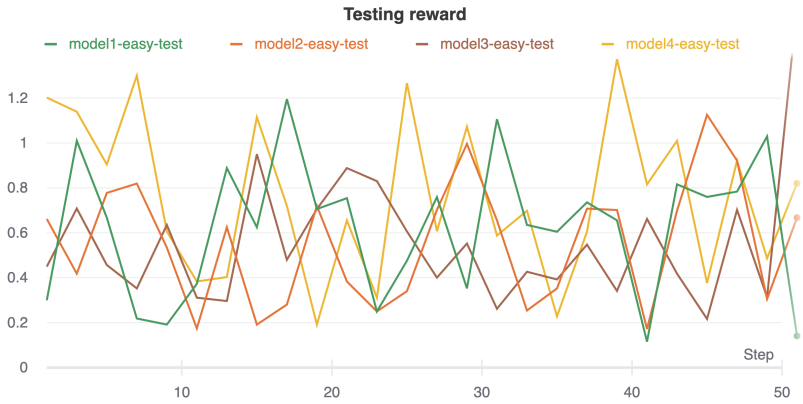

Figure 7: testing reward by models in easy configure

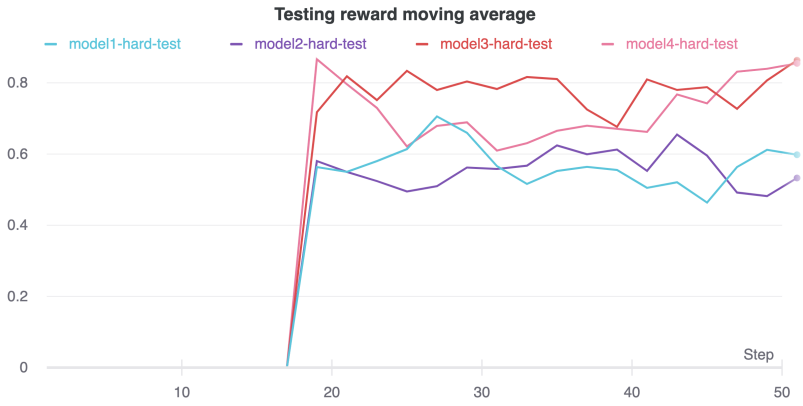

Figure 8: testing moving average by models in hard configure

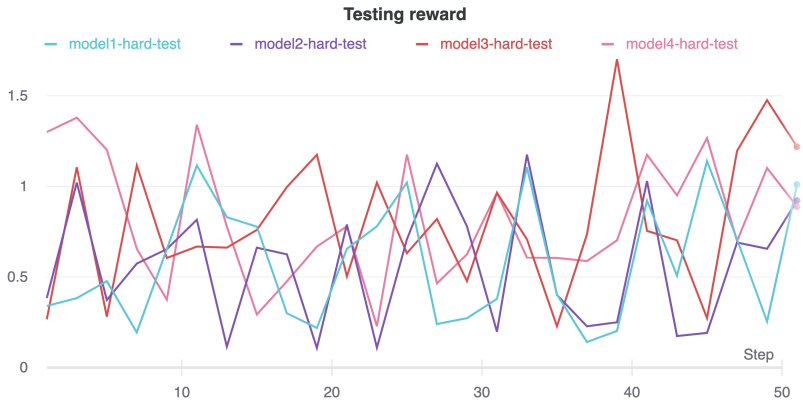

Figure 9: testing reward by models in hard configure