Project Report: Reinforcement Learning - Learn to Cut

Name: Khanh-Tung Nguyen-Ba
UNI: kn2465

1. Project Goal

The goal of the project is to apply reinforcement learning to solve Integer Programming problems more efficiently than commonly used heuristic methods. We reformulated the IP problems using deep RL to adaptively select candidate cuts to the cutting plane method, a very popular subroutine used by all modern IP solvers. The core of the work is the implementation of Tang et al. (2019) [1] with some enhancement from Mania at al. (2018) [2] using Ray for distributed computing [3].

2. Algorithm Design

2.1 Formulate IP as a RL problem

Below we quickly summarize the idea of [1]. The MDP for IP problems is formulated as:

- State Space S: The representation of each state is $s_t = \{C^{(t)}, c, x^*_{LP}(t), D^{(t)}\}$ where $C^{(t)} = \{a_i^T x \le b_i\}_{i=1}^{N_t}$ is the feasible region including the original constraints and the cuts added so far, $D^{(t)}$ is the set of candidate Gomory's cuts, $x^*_{LP}(t)$ is the LP optimal solution.
- Action Space A: At each time step t, this is $D^{(t)}$. Each action is a possible Gomory's cutting plane, represented as $e_i^T x \le d_i$
- Reward: $r_t = c^T x^*_{LP}(t+1) - c^T x^*_{LP}(t)$, the gap between objective values of consecutive LP solutions
- Transition: At time step t, a new action of adding a cutting plane $e_i^T x \le d_i$ is taken, then $C^{(t+1)} = C^{(t)} \cup \{e_i^T x \le d_i\}$, we have a new state $s_{t+1} = \{C^{(t+1)}, c, x^*_{LP}(t+1), D^{(t+1)}\}$

2.2 Policy Network

We considered two different architectures:

- Attention: The attention embedding is a 2-layer neural network with tanh activation. As detailed in [1], the rationale for using a simple attention architecture is to stay agnostic to the order of the action space.
- LSTM + Attention: Adding a level of LSTM embedding allows the network to generalize to different input sizes.

2.3 Training

Here we borrowed the ideas from Augmented Random Search [2] to improve the training of the Evolutionary Strategies algorithm. Specifically:

- Scale update steps by the standard deviation of the rewards collected at each iterations (V1)
- Normalize states to ensure equal weight on the different components of the states(V2)
- Drop perturbation directions that yield the least improvement of the rewards (V1-t, V2-t)

3. Pseudo-code

The pseudo code for the policy rollout per [1]:

```
1. Input: policy network parameter $\theta$, IP instance parameterized by
   $c, A, b$, number of iterations $T$.
2. Initialize iteration counter $t = 0$
3. Initialize minimization LP with constraints $C^{(0)} = \{Ax \leq b\}$ and
   cost vector $c$. Solve to obtain $x^*_{LP}(0)$. Generate set of candidate cuts
   $D^{(0)}$
4. while $x^*_{LP}(t)$ not all integer-valued and $t \leq T$ do
     a. Construct state $s_t = \{C^{(t)}, c, x^*_{LP}(t), D^{(t)}\}$
     b. Sample action using the distribution over candidate cuts
        given by policy $\pi_\theta$, as $a_t \ \pi_\theta(.|s_t)$. Here action $a_t$ corresponds
        to a cut $e_i^T x \leq d_i$
     c. Append the cut to the constraint set $C^{(t+1)} = C^{(t)} \cup \{e_i^T x \leq d_i\}$
        . Solve for $x^*_{LP}(t+1)$. Generate $D^{(t+1)}$. Compute reward $r_t$
     d. $t \leftarrow t + 1$
5. end while
```

The pseudo code for the Augmented Random Search per [2]:

```
1. Hyperparameters: step-size $\alpha$, number of directions sampled per
   iterations $N$, standard deviation of the exploration noise $v$,
   number of top-performing directions to use $b$ ($b < N$ is allowed
   only for V1-t and V2-t)
2. while ending condition not satisfied do
     a. Sample $\delta_1, \delta_2, ..., \delta_N \in R^{p \times n}$ with i.i.d standard normal entries
```

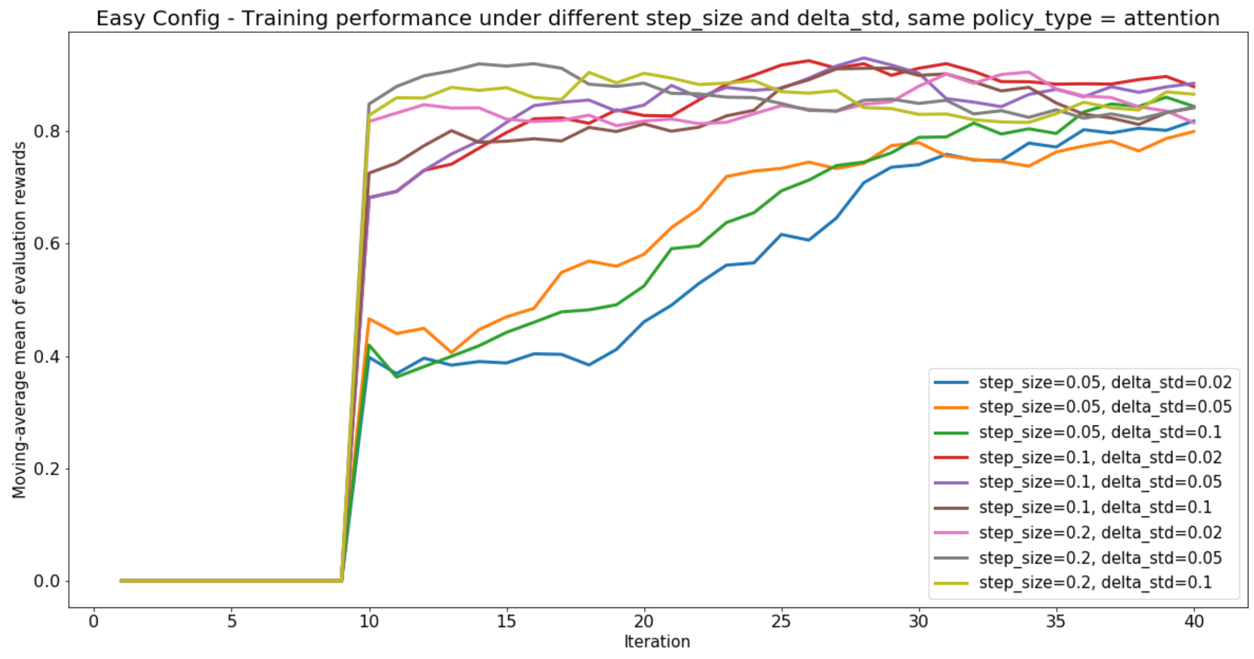
```
        b. Collect $2N$ rollouts of horizon $T$ and their corresponding
           rewards using the $2N$ policies
           **V1:**
           $\pi_{j,(k),+}(x) = (M_j + v\theta_k)(x)$
           $\pi_{j,(k),-}(x) = (M_j - v\theta_k)(x)$

           **V2:**
           $\pi_{j,(k),+}(x) = (M_j + v\theta_k)(\text{diag}(\Sigma_j)^{-0.5}(x - \mu_j))$
           $\pi_{j,(k),-}(x) = (M_j - v\theta_k)(\text{diag}(\Sigma_j)^{-0.5}(x - \mu_j))$

        c. Sort the directions $\delta_k$ by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by
           $\delta_{(k)}$ the k-th largest direction.
        d. Make the update steps:
           $\theta_{j+1} = \theta_j + \frac{\alpha}{\beta\sigma_R}\sum_{k=1}^{b}[r(\pi_{j,(k),+}) - r(\pi_{j,(k),-})]\delta_{(k)}$,
           where $\sigma_R$ is the standard deviation of the $2b$ rewards used
           in the update step.
        e. **V2:** Set $\mu_{j+1}, \Sigma_{j+1}$ to be the mean and covariance of the
           $2NH(j+1)$ states encountered from the start of the
           training.
        f. $j = j + 1$
    3. end while
```

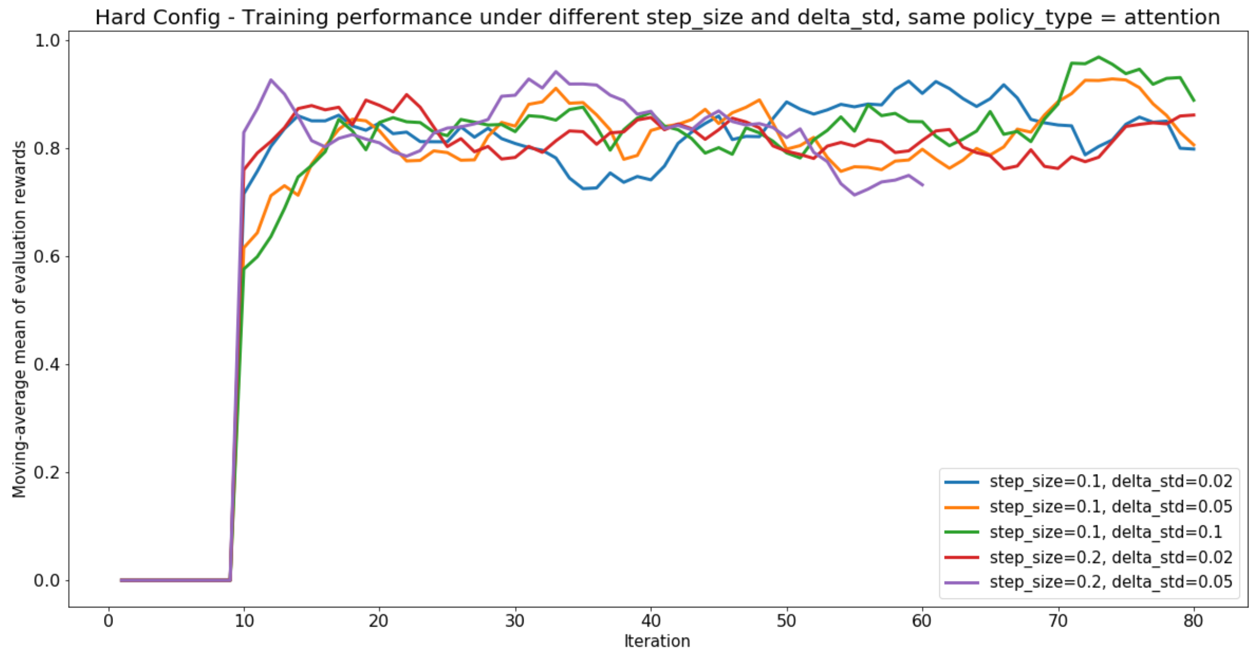
4. Results

Below we show the 10-iteration moving average mean of the evaluation rewards from training on the easy and hard config using:

- V2 training method
- Attention (no LSTM) policy network
- For each iteration, 10 roll-outs for each gradient update, 5 roll-outs for evaluation

To be clear about what we are showing, at each iteration we use rewards from 5 roll-outs for evaluation and capture the mean, this is a more accurate reflection of the algorithm performance than result from 1 roll-out, which depends on the randomness of the instances and takes more iterations to show convergence.





Additional results can be found in the Appendix, including:

- Visualize moving-average standard deviation of rewards
- Comparison between Attention and Attention + LSTM training performances
- Comparison between V1, V1-t, V2, V2-t training performances

5. Reference

[1] Tang, Y., Agrawal, S., & Faenza, Y. (2019). Reinforcement learning for integer programming:Learning to cut. *ArXiv*.

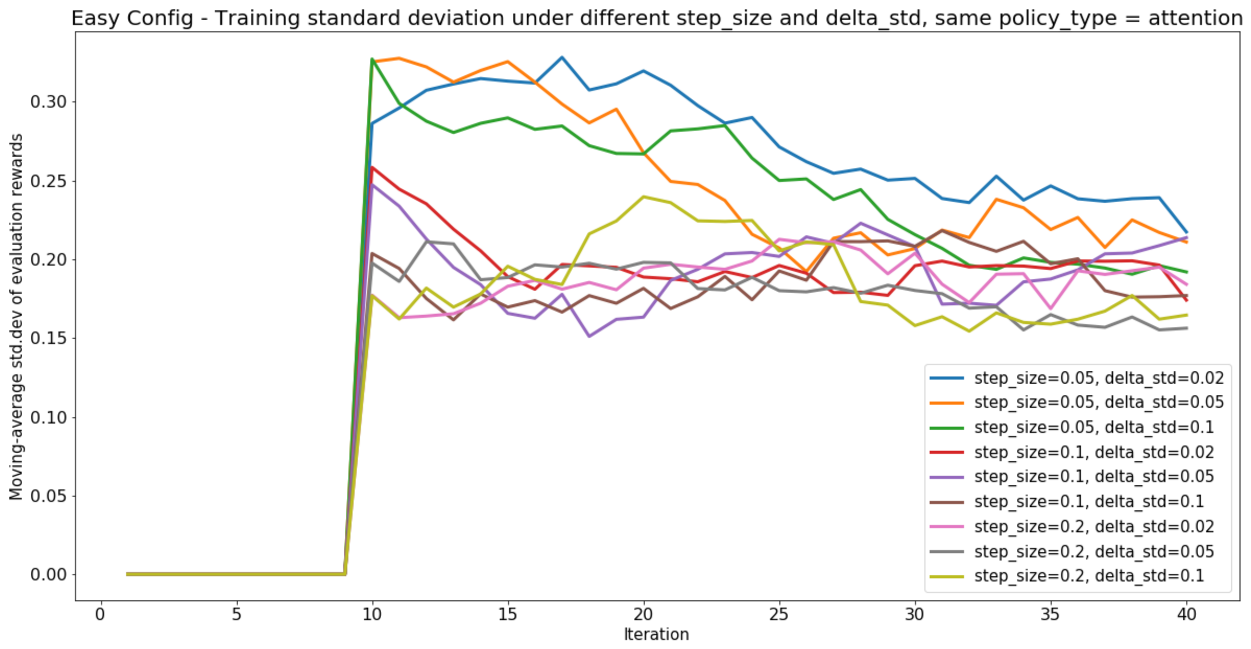
[2] Mania, H., Guy, A., & Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *ArXiv*, 1–22.

[3] Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., … Stoica, I. (2007). Ray: A distributed framework for emerging AI applications. *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018*, 561–577.

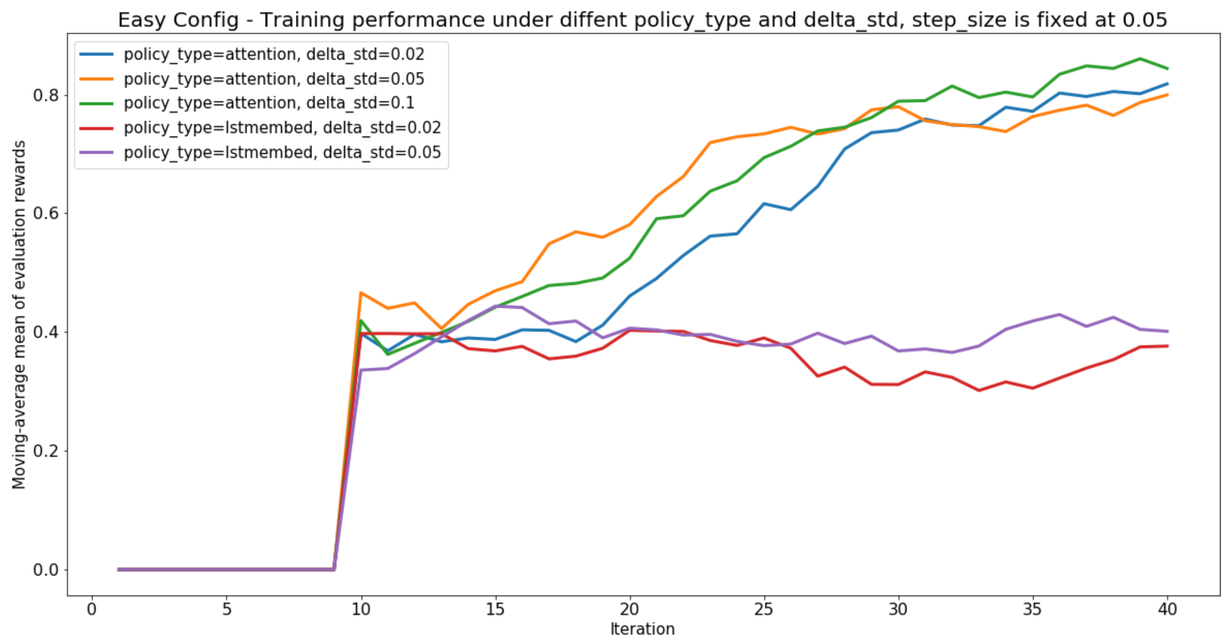
6. Appendix

6.1 Visualize moving-average standard deviation of evaluation rewards

This shows how the standard deviation of the evaluation rewards (over 5 roll-outs) has a decreasing trend as training progresses. This is in line with the effect of scaling the update steps by standard deviation in [2].



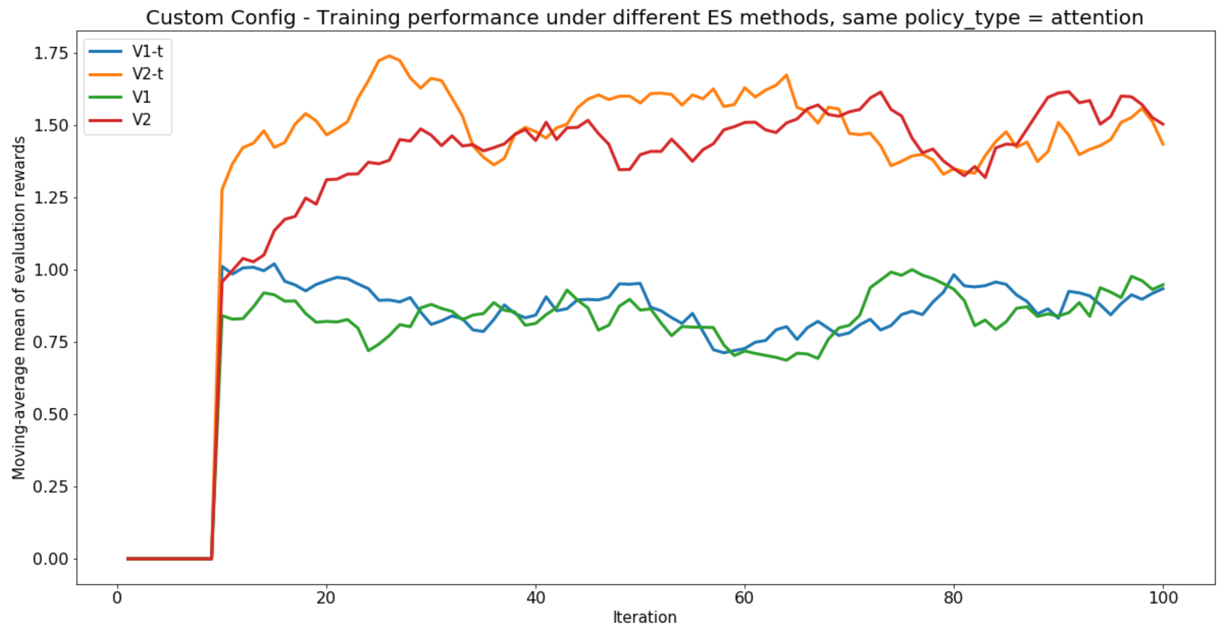
6.2 Comparison between Attention and Attention + LSTM training performances



One possible reason for the worse performance of the LSTM+Attention policy network is that it requires more training due to the larger number of parameters to tune.

6.3 Comparison between V1, V2, V1-t and V2-t

Due to computational constraints, the results below are from training on a custom config using 15x15 instances, maximum rollouts T = 12, b = 6, policy_type = attention, step_size = 0.1, delta_std = 0.02. We can see two things: V2 obviously outperforms V1, and dropping perturbation directions does not really make much difference as training progresses.


Custom Config - Training performance under different ES methods, same policy_type = attention
V1-t
V2-t
V1
V2
Moving-average mean of evaluation rewards
Iteration