10703 Deep Reinforcement Learning and Control

Russ Salakhutdinov

Machine Learning Department
rsalakhu@cs.cmu.edu

Slides borrowed from
Katerina Fragkiadaki

Imitation Learning
Reinforcement Learning: Learning policies guided by sparse rewards, e.g., win the game.

- **Good**: simple, cheap form of supervision
- **Bad**: High sample complexity

Where is it successful so far?

- In simulation, where we can afford a lot of trials, easy to parallelize
- Not in robotic systems:
  - action execution takes long
  - we cannot afford to fail
  - safety concerns

Crusher robot

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010
Reward shaping

Ideally we want **dense in time** rewards to closely guide the agent closely along the way.

Who will supply those shaped rewards?

1. We will manually design them: “*cost function design by hand remains one of the 'black arts' of mobile robotics, and has been applied to untold numbers of robotic systems*”

2. We will learn them from demonstrations: “*rather than having a human expert tune a system to achieve desired behavior, the expert can demonstrate desired behavior and the robot can tune itself to match the demonstration*”

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010
Reward shaping

Ideally we want **dense in time** rewards to closely guide the agent closely along the way.

Who will supply those shaped rewards?

1. **We will manually design them**: “cost function design by hand remains one of the ‘black arts’ of mobile robotics, and has been applied to untold numbers of robotic systems”

2. **We will learn them from demonstrations**: “rather than having a human expert tune a system to achieve desired behavior, the expert can demonstrate desired behavior and the robot can tune itself to match the demonstration”

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010
Learning from demonstrations a.k.a. Imitation Learning:
Supervision through an expert (teacher) that provides a set of demonstration trajectories: sequences of states and actions.

Imitation learning is useful when it is easier for the expert to demonstrate the desired behavior rather than:
   a) coming up with a reward that would generate such behavior,
   b) coding up with the desired policy directly.
The Imitation Learning problem

The agent (learner) needs to come up with a policy whose resulting state, action trajectory distribution matches the expert trajectory distribution.

Does this remind us of something?

GANs! Generative Adversarial Networks (on state-action trajectories)

Generative Adversarial Networks, Goodfellow et al. 2014
The Imitation Learning problem: Challenge

Actions along the trajectories are interdependent, as actions determine state transitions and thus states and actions down the road.

interdependent labels -> structure prediction
The Imitation Learning problem: Challenge

Actions along the trajectories are interdependent, as actions determine state transitions and thus states and actions down the road.

interdependent labels -> structure prediction

Action interdependence in time:

Algorithms developed in Robotics for imitation learning found applications in structured predictions problems, such as, sequence generation/labelling e.g. parsing.
For taking this structure into account, numerous formulations have been developed:

- **Direct**: Supervised learning for policy (mapping states to actions) using the demonstration trajectories as ground-truth (a.k.a. behavior cloning)

- **Indirect**: Learning the latent rewards/goals of the teacher and planning under those rewards to get the policy, a.k.a. Inverse Reinforcement Learning (later in class)

Experts can be:

- Humans
- Optimal or near Optimal Planners/Controllers
Outline

This lecture

• Behavior Cloning: Imitation learning as supervised learning
• Compounding errors
• Demonstration augmentation techniques
• DAGGER
• Structured prediction as Decision Making (learning to search)

Next lecture:

• Inverse reinforcement learning
• Feature matching
• Max margin planning
• Maximum entropy IRL
• Adversarial Imitation learning
Outline

This lecture

• Behavior Cloning: Imitation learning as supervised learning
• Compounding errors
• Demonstration augmentation techniques
• DAGGER
• Structured prediction as Decision Making (learning to search)

Next lecture:

• Inverse reinforcement learning
• Feature matching
• Max margin planning
• Maximum entropy IRL
• Adversarial Imitation learning
Imitation Learning for Driving

Driving policy: a mapping from (history of) observations to steering wheel angles

\[ \pi_\theta (u_t | o_t) \]

\[ o_t \]

\[ u_t \]

End to End Learning for Self-Driving Cars, Bojarski et al. 2016
Imitation Learning for Driving

Driving policy: a mapping from (history of) observations to steering wheel angles

Behavior Cloning=Imitation Learning as Supervised learning

• Assume actions in the expert trajectories are i.i.d.
• Train a classifier or regressor to map observations to actions at each time step of the trajectory.

End to End Learning for Self-Driving Cars, Bojarski et al. 2016
Classifier or regressor?

Because multiple actions \( u \) may be plausible at any given observation \( o \), policy network \( p_{\pi_\theta}(u_t|o_t) \) usually is not a regressor but rather:

- A classifier (e.g., softmax output and cross-entropy loss, after discretizing the action space)

\[
J(\theta) = - \sum_{i=1}^{m} \sum_{k=1}^{K} 1_{y(i) = k} \log[P(y(i) = k|x(i); \theta)]
\]

- A GMM (Gaussian Mixture Model), where means and variances are parametrized at the output of a neural net, (e.g., hand-writing generation Graves 2013)

- A stochastic network (previous lecture)
Independent in time errors

error at time $t$ with probability $\varepsilon$

$\mathbb{E}[	ext{Total errors}] \approx \varepsilon T$
error at time t with probability $\epsilon$

$E[\text{Total errors}] \approx \epsilon(T + (T-1) + (T-2) + \ldots + 1) \propto \epsilon T^2$
CHAPTER 1. INTRODUCTION

Expert trajectory

Learned Policy

No data on how to recover

\[ p_{\pi^*}(o_t) \neq p_{\pi_\theta}(o_t) \]

Figure 1.1: Mismatch between the distribution of training and test inputs in a driving scenario.

Many state-of-the-art software systems we use everyday. Systems based on supervised learning already translate our documents, recommend what we should read ([Yue and Guestrin, 2011]), watch ([Toscher et al., 2009]) or buy, read our handwriting ([Daumé III et al., 2009]) and filter spam from our emails ([Weinberger et al., 2009]), just to name a few. Many subfields of artificial intelligence, such as natural language processing (the understanding of natural language by computers) and computer vision (the understanding of visual input by computers), now deeply integrate machine learning. Despite this widespread proliferation and success of machine learning in various fields and applications, machine learning has had a much more limited success when applied in control applications, e.g. learning to drive from demonstrations by human drivers. One of the main reasons behind this limited success is that control problems exhibit fundamentally different issues that are not typically addressed by standard supervised learning techniques. In particular, much of the theory and algorithms for supervised learning are based on the fundamental assumption that inputs/observations perceived by the predictor to make its predictions are independent and always coming from the same underlying distribution during both training and testing ([Hastie et al., 2001]). This ensures that after seeing enough training examples, we will be able to predict well on new examples (at least in expectation). However, this assumption is clearly violated in control tasks as these are inherently dynamic and sequential: one must perform a sequence of actions over time that have consequences on future inputs or observations of the system, to achieve a goal or successfully perform the task. As predicting actions to execute influence future inputs, this can lead to a large mismatch between the inputs observed under training demonstrations, and those observed during test executions of the learned behavior. This is illustrated schematically in Figure 1.1.

This problem has been observed in previous work. [Pomerleau, 1989], who trained a
SL succeeds when training and test data distributions match. That is a fundamental assumption.
Solutions

Change $p_{\pi^*}(o_t)$ using demonstration augmentation!

Add examples in expert demonstration trajectories to cover the states/observations points where the agent will land when trying out its own policy.
This lecture

- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)

Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning
“In addition, the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made. Partial initial training on a variety of simulated road images should help eliminate these difficulties and facilitate better performance. “ ALVINN: An autonomous Land vehicle in a neural Network, Pomerleau 1989
“DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. …”,

End to End Learning for Self-Driving Cars, Bojarski et al. 2016
A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots Giusti et al.
A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots Giusti et al.
Outline

This lecture

- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
  - Structured prediction as Decision Making (learning to search)
- Imitating MCTS

Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning
Dataset AGGregation: bring learner’s and expert’s trajectory distributions closer by labelling additional data points resulting from applying the current policy.

1. train $\pi_{\theta}(u_t|o_t)$ from human data $D_{\pi^*} = \{o_1, u_1, \ldots, o_N, u_N\}$
2. run $\pi_{\theta}(u_t|o_t)$ to get dataset $D_{\pi} = \{o_1, \ldots, o_M\}$
3. Ask human to label $D_{\pi}$ with actions $u_t$
4. Aggregate: $D_{\pi^*} \leftarrow D_{\pi^*} \cup D_{\pi}$
5. GOTO step 1.

Problems:
- execute an unsafe/partially trained policy
- repeatedly query the expert
DAGGER (in a real platform)

Application on drones: given RGB from the drone camera predict steering angles

Learning monocular reactive UAV control in cluttered natural environments, Ross et al. 2013
Application on drones: given RGB from the drone camera predict steering angle

Caveats:

1. Interaction with the expert is hard: Is hard for the expert to provide the right magnitude for the turn without feedback of his own actions! 
   **Solution**: provide him his visual feedback

Learning monocular reactive uav control in cluttered natural environments, Ross et al. 2013
Caveats:

1. Is hard for the expert to provide the right magnitude for the turn without feedback of his own actions! Solution: provide him his visual feedback

2. The expert’s reaction time to the drone’s behavior is large, this causes imperfect actions to be commanded. Solution: play-back in slow motion offline and record their actions.

3. Executing an imperfect policy causes accidents, crashes into obstacles. Solution: safety measures which make again the data distribution matching imperfect between train and test, but good enough.

Learning monocular reactive uav control in cluttered natural environments, Ross et al. 2013
Outline

This lecture

• Behavior Cloning: Imitation learning as supervised learning
• Compounding errors
• Demonstration augmentation techniques
• DAGGER
• Structured prediction as Decision Making (learning to search)

Next lecture:

• Inverse reinforcement learning
• Feature matching
• Max margin planning
• Maximum entropy IRL
• Adversarial Imitation learning
Structured prediction: a learner makes predictions over a set of interdependent output variables and observes a joint loss.

Example: part of speech tagging

\[
\begin{align*}
  x &= \text{the monster ate the sandwich} \\
  y &= \text{Dt Nn Vb Dt Nn}
\end{align*}
\]

A structured prediction problem consists of:
- an input space \( \mathcal{X} \), an output space \( \mathcal{Y} \)
- a fixed but unknown distribution \( \mathcal{D} \) over \( \mathcal{X} \times \mathcal{Y} \), and
- a non-negative loss function \( f : \mathcal{X} \rightarrow \mathcal{Y} \) which measures the distance between the true \( y^* \) and predicted \( \hat{y} \) outputs.
Structured prediction: a learner makes predictions over a set of interdependent output variables and observes a joint loss.

Example: part of speech tagging

```
x = the monster ate the sandwich
y = Dt   Nn   Vb   Dt   Nn
```

A structured prediction problem consists of:
- an input space $\mathcal{X}$, an output space $\mathcal{Y}$
- a fixed but unknown distribution $D$ over $\mathcal{X} \times \mathcal{Y}$, and
- a non-negative loss function $f : \mathcal{X} \rightarrow \mathcal{Y}$ which measures the distance between the true $y^*$ and predicted $\hat{y}$ outputs.

The goal of structured learning is to use $N$ samples $(x_i, y_i)_{i=1}^{N}$ to learn a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the expected structured loss under $D$. 
Structured prediction

Sequence labelling:
Part of speech tagging

\[ x = \text{the monster ate the sandwich} \]
\[ y = \text{Dt Nn Vb Dt Nn} \]
Structured prediction

Sequence labelling:
- Part of speech tagging
- NER (Name Entity Recognition)

\[ x = \text{Yesterday I traveled to Lille} \]
\[ y = - \quad \text{PER} \quad - \quad - \quad \text{LOC} \]
Structured prediction

Sequence labelling:
- Part of speech tagging
- NER (Name Entity Recognition)
- Attentive Tracking
Structured prediction

Sequence labelling:
Part of speech tagging
NER
Tracking

Sequence generation:
Captioning
Machine translation
Optimizing Graphical Models for Structured prediction

- Encode output labels as a MRF
- Learn parameters of that model to:
  - Maximize $p(\text{true labels} \mid \text{input})$
  - Minimize $\text{loss}(\text{true labels}, \text{predicted labels})$

Let $G = (V, E)$ be a graph such that $Y = (Y_v)_{v \in V}$, so that $Y$ is indexed by the vertices of $G$. Then $(X, Y)$ is a conditional random field when the random variables $Y_v$, conditioned on $X$, obey the Markov property with respect to the graph: $p(Y_v | X, Y_w, w \neq v) = p(Y_v | X, Y_w, w \sim v)$, where $w \sim v$ means that $w$ and $v$ are neighbors in $G$. 
Optimizing Graphical Models for Structured prediction

- Encode output labels as a MRF
- Learn parameters of that model to:
  - Maximize $p(\text{true labels} \mid \text{input})$
  - Minimize loss($\text{true labels}$, predicted labels)

- Assumed Independence Assumptions may not hold
- Computationally intractable with too many “edges” or non-decomposable loss functions (that involve many $y$'s)
Instead: Decomposition of label

**Sequence generation/labelling:**

We can define an ordering and generate labels one at a time, where each generated output **depends on all previous ones:** Sequential data admits the natural sequential ordering.

**Image generation/labelling:**

Here again we can define an ordering:

Pixel Recurrent Neural Networks, van den Oord et al
Structured prediction as sequential decision making

When \( y \) decomposes in an ordered manner, a sequential decision making process emerges.
When $y$ decomposes in an ordered manner, a sequential decision making process emerges.
Structured prediction as sequential decision making

Example: Sequence labelling

- **State**: Captures input sequence $x$ and whatever labels (here part of speech tags) we have produced so far
- **Actions**: Next label to output
- **Policy**: A mapping of the input $x$ and labels generated so far to the next label
- **Reward**: Agreement of the predicted $\hat{y}$ with ground-truth $y^*$: $\ell(e) = \ell(y^*, y_e)$
Example: Image Captioning

- **State**: Captures the image and whatever words we have produced so far
- **Actions**: Next word to output
- **Policy**: A mapping of the state to the next word
- **Reward**: Agreement of the predicted $\hat{y}$ with ground-truth $y^*$: $\ell(e) = \ell(y^*, y_e)$
Structured prediction as sequential decision making

Sequence labelling:
- Parsing
- NER
- Tracking

Sequence generation:
- Captioning
- Machine translation
- Etc..

What function approximation shall we use for our state representations in case of sequence/image labelling/generation?
Recurrent Neural Networks

- RNNs tie the weights at each time step
- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled between inputs and outputs, as well as between output labels
- In practice, limitations from SGD training, capacity, initialization, etc.
Recurrent Neural Network (single hidden layer)

- Given list of **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- At a single time step:

$$ h_t = \sigma \left( W^{(hh)} h_{t-1} + W^{(hx)} x_t \right) $$

$$ \hat{y}_t = \text{softmax} \left( W^{(S)} h_t \right) $$
Recurrent Neural Networks

For sequence labelling problems, actions of the labelling policies are $y_t$, e.g., part of speech tags.

For sequence generation, actions of the labelling policies are $y_t = x_{t+1}$, e.g., word in answer generation $\hat{P}(x_{t+1} = v_j | x_t, \ldots, x_1) = \hat{y}_{t,j}$.
The network is typically trained to maximize the log-likelihood of the output sequences given the input sequences of a training set $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$:

$$\theta^* = \arg\max_{\theta} \log \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} P_\theta(y^{(i)}, x^{(i)})$$

If the likelihood of an example decomposes over individual time steps:

$$\log P_\theta(y|x) = \sum_t \log P_\theta(y_t|h_t)$$

Else loss is computed at the end of the sequence and is back propagated through time.
Recurrent Neural Networks

The network is typically trained to maximize the log-likelihood of the output sequences given the input sequences of a training set $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$:

\[
\theta^* = \arg\max_\theta \log \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} P_\theta(y^{(i)}, x^{(i)})
\]

If the likelihood of an example decomposes over individual time steps:

\[
\log P_\theta(y|x) = \sum_t \log P_\theta(y_t|h_t)
\]

Else loss is computed at the end of the sequence and is back propagated through time.

A learned policy is the inference function of the model:

\[
\hat{\theta}(h_t) = \arg\max_y P(y_t = y|\theta)
\]

The reference policy is the policy that always outputs the true labels:

\[
\theta^*(h_t) = y_t
\]
Recurrent Neural Networks

The regular training procedure of RNNs treat true labels $y_t$ as actions while making forward passes.

Hence, the learning agent follows trajectories generated by the reference policy rather than the learned policy. In other words, it learns:

$$\hat{\theta}^{sup} = \arg \min_{\theta} \mathbb{E}_{h \sim d_{\pi^*}} [l_{\theta}(h)]$$

However, our true goal is to learn a policy that minimizes error under its own induced state distribution:

$$\hat{\theta} = \arg \min_{\theta} \mathbb{E}_{h \sim d_{\theta}} [l_{\theta}(h)]$$
DAGGER for sequence labelling/generation with RNNs

1: function TRAIN($N$, $\alpha$)
2:     Initialize $\alpha = 1$.
3:     Initialize model parameters $\theta$.
4:     for $i = 1..N$ do
5:         Set $\alpha = \alpha \cdot p$.
6:         Randomize a batch of labeled examples.
7:         for each example $(x, y)$ in the batch do
8:             Initialize $h_0 = \Phi(X)$.
9:             Initialize $D = \{(h_0, y_0)\}$.
10:            for $t = 1..|Y|$ do
11:                Uniformly randomize a floating-number $\beta \in [0, 1)$.
12:                if $\alpha < \beta$ then
13:                    Use true label $\tilde{y}_{t-1} = y_{t-1}$
14:                else
15:                    Use predicted label: $\tilde{y}_{t-1} = \text{arg max}_y P(y \mid h_{t-1}; \theta)$.
16:                end if
17:                Compute the next state: $h_t = f_{\theta}(h_{t-1}, \tilde{y}_{t-1})$.
18:                Add example: $D = D \cup \{(h_t, y_t)\}$.
19:            end for
20:     end for
21:     Online update $\theta$ by $D$ (mini-batch back-propagation).
22: end function

Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks, Bengio(Samy) et al.
Imitation Learning with Recurrent Neural Networks, Nyuyen 2016
Data augmentation(4): Mocap generation

Graves et al.
Demonstration Augmentation: Temporal subsampling

- Two tasks considered: pick and place, move to desired pose
- **Input** $x$: the poses of all the objects in the seen (rotations, translations) and the pose of the end effector
- **Output** $y$: the desired next pose of the end effector

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016
Demonstration Augmentation: Temporal subsampling

- **Supervision**: expert trajectories in the simulator
- **Data augmentation**: consider multiple trajectories by subsampling in time the expert ones, and by translating in space the end effector

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016
RNNs for Imitation(1)

- Multimodality of actions -> GMM loss!

- Predict mixture weights over a Gaussian Mixture Model at the output (alphas) and mean and variances for the mixture components.

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016
• Multimodality of actions $\rightarrow$ GMM loss!

• Predict mixture weights over a Gaussian Mixture Model at the output (alphas) and mean and variances for the mixture components.
Recurrent Neural Networks for Imitation(1)

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016
Learning Manipulation Trajectories Using Recurrent Neural Networks
Learning to imitate Search

Task: playing Atari games

1. DQN : model free, knows nothing about the game dynamics

2. MCTS: performs better than DQN but:
   a. takes too long per step to choose the action (too many trees to search)
   b. assume access to the game simulator to `look ahead”

Idea: instead of learning from trial and error learn to imitate MCTS

Let MCTS run for long enough to provide the ground-truth actions

Dealing with compounding errors: MCTS uses the current learnt policy to unfold the tree
Table 1: Performance (game scores) of the four real-time game playing agents, where UCR is short for UCT-to-Regression, UCC is short for UCT-to-Classification, and UCC-I is short for UCT-to-Classification-Interleaved.

<table>
<thead>
<tr>
<th>Agent</th>
<th>B.Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S.Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>4092</td>
<td>168</td>
<td>470</td>
<td>20</td>
<td>1952</td>
<td>1705</td>
<td>581</td>
</tr>
<tr>
<td>-best</td>
<td>5184</td>
<td>225</td>
<td>661</td>
<td>21</td>
<td>4500</td>
<td>1740</td>
<td>1075</td>
</tr>
<tr>
<td>UCC</td>
<td>5342 (20)</td>
<td>175 (5.63)</td>
<td>558 (14)</td>
<td>19 (0.3)</td>
<td>11574 (44)</td>
<td>2273 (23)</td>
<td>672 (5.3)</td>
</tr>
<tr>
<td>-best</td>
<td>10514</td>
<td>351</td>
<td>942</td>
<td>21</td>
<td>29725</td>
<td>5100</td>
<td>1200</td>
</tr>
<tr>
<td>-greedy</td>
<td>5676</td>
<td>269</td>
<td>692</td>
<td>21</td>
<td>19890</td>
<td>2760</td>
<td>680</td>
</tr>
<tr>
<td>UCC-I</td>
<td>5388 (4.6)</td>
<td>215 (6.69)</td>
<td>601 (11)</td>
<td>19 (0.14)</td>
<td>13189 (35.3)</td>
<td>2701 (6.09)</td>
<td>670 (4.24)</td>
</tr>
<tr>
<td>-best</td>
<td>10732</td>
<td>413</td>
<td>1026</td>
<td>21</td>
<td>29900</td>
<td>6100</td>
<td>910</td>
</tr>
<tr>
<td>-greedy</td>
<td>5702</td>
<td>380</td>
<td>741</td>
<td>21</td>
<td>20025</td>
<td>2995</td>
<td>692</td>
</tr>
<tr>
<td>UCR</td>
<td>2405 (12)</td>
<td>143 (6.7)</td>
<td>566 (10.2)</td>
<td>19 (0.3)</td>
<td>12755 (40.7)</td>
<td>1024 (13.8)</td>
<td>441 (8.1)</td>
</tr>
</tbody>
</table>

Table 2: Performance (game scores) of the off-line UCT game playing agent.

<table>
<thead>
<tr>
<th>Agent</th>
<th>B.Rider</th>
<th>Breakout</th>
<th>Enduro</th>
<th>Pong</th>
<th>Q*bert</th>
<th>Seaquest</th>
<th>S.Invaders</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCT</td>
<td>7233</td>
<td>406</td>
<td>788</td>
<td>21</td>
<td>18850</td>
<td>3257</td>
<td>2354</td>
</tr>
</tbody>
</table>

but... 800 games * 1000 actions/game * 10000 rollouts/action * 300 steps/rollback = 2.4e12 steps