Policy Gradient Methods

Sources: Stanford CS 231n, Berkeley Deep RL course, David Silver’s RL course
Policy Gradient Methods

• Instead of indirectly representing the policy using Q-values, it can be more efficient to parameterize and learn it directly
• Especially in large or continuous action spaces

Image source: OpenAI Gym
Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:

\[ \pi_\theta(s, a) \approx P(a|s) \]
Stochastic policy representation

• Learn a function giving the probability distribution over actions from the current state:

$$\pi_\theta(s, a) \approx P(a|s)$$

• Why stochastic policies?
  • There are examples even of grid world scenarios where only a stochastic policy can reach optimality.

The agent can’t tell the difference between the gray cells.
Stochastic policy representation

- Learn a function giving the probability distribution over actions from the current state:
  \[ \pi_\theta(s, a) \approx P(a|s) \]

- Why stochastic policies?
  - It’s mathematically convenient!
  - Softmax policy:
    \[ \pi_\theta(s, a) = \frac{\exp(f_\theta(s, a))}{\sum_{a'} \exp(f_\theta(s, a'))} \]
  - Gaussian policy (for continuous action spaces):
    \[ \pi_\theta(s, a) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - f_\theta(s))^2}{2\sigma^2}\right) \]
Expected value of a policy

\[ J(\theta) = \mathbb{E} \left[ \sum_{t \geq 0} \gamma^t r_t \mid \pi_\theta \right] \]

\[ = \mathbb{E}_\tau [r(\tau)] \]

Expectation of return over trajectories \( \tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots) \)

\[ = \int_\tau r(\tau)p(\tau; \theta) d\tau \]

Probability of trajectory \( \tau \) under policy with parameters \( \theta \)
Finding the policy gradient

\[ J(\theta) = \int_\tau r(\tau)p(\tau; \theta)d\tau \]

\[ \nabla_\theta J(\theta) = \int_\tau r(\tau)\nabla_\theta p(\tau; \theta)d\tau \]

\[ = \int_\tau r(\tau)p(\tau; \theta)\frac{\nabla_\theta p(\tau; \theta)}{p(\tau; \theta)}d\tau \]

\[ = \int_\tau r(\tau)p(\tau; \theta)\nabla_\theta \log p(\tau; \theta) d\tau \]

\[ = \mathbb{E}_\tau[r(\tau)\nabla_\theta \log p(\tau; \theta)] \]
Finding the policy gradient

\[ \nabla_\theta J(\theta) = \mathbb{E}_\tau [r(\tau) \nabla_\theta \log p(\tau; \theta)] \]

The score function

Probability of trajectory
\[ \tau = (s_0, a_0, s_1, a_1, \ldots) \]

\[ p(\tau; \theta) = \prod_{t \geq 0} \pi_\theta(s_t, a_t) P(s_{t+1}|s_t, a_t) \]

\[ \log p(\tau; \theta) = \sum_{t \geq 0} \left[ \log \pi_\theta(s_t, a_t) + \log P(s_{t+1}|s_t, a_t) \right] \]

\[ \nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \]

The score function
Score function $\nabla_\theta \log \pi_\theta(s, a)$

- For softmax policy:
  \[
  \pi_\theta(s, a) = \frac{\exp(f_\theta(s, a))}{\sum_{a'} \exp(f_\theta(s, a'))} \\
  \nabla_\theta \log \pi_\theta(s_t, a_t) = \nabla_\theta f_\theta(s, a) - \sum_{a'} \pi_\theta(s, a') \nabla_\theta f_\theta(s, a')
  \]

- For Gaussian policy:
  \[
  \pi_\theta(s, a) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(a - f_\theta(s))^2}{2\sigma^2}\right) \\
  \nabla_\theta \log \pi_\theta(s_t, a_t) = \frac{(a - f_\theta(s))}{\sigma^2} \nabla_\theta f_\theta(s) - \text{const.}
  \]
Finding the policy gradient

\[ \nabla_\theta J(\theta) = \mathbb{E}_\tau [r(\tau) \nabla_\theta \log p(\tau; \theta)] \]

\[ \nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \]

\[ \nabla_\theta J(\theta) = \mathbb{E}_\tau \left[ \left( \sum_{t \geq 0} \gamma^t r_t \right) \left( \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \right) \right] \]

- Return of trajectory \( \tau \)
- Gradient of log-likelihood of actions under current policy

How do we estimate the gradient in practice?
Finding the policy gradient

\[ \nabla_\theta J(\theta) = \mathbb{E}_\tau[ r(\tau) \nabla_\theta \log p(\tau; \theta) ] \]

\[ \nabla_\theta \log p(\tau; \theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \]

\[ \nabla_\theta J(\theta) = \mathbb{E}_\tau \left[ \left( \sum_{t \geq 0} \gamma^t r_t \right) \left( \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(s_t, a_t) \right) \right] \]

- Stochastic approximation: sample \( N \) trajectories \( \tau_1, \ldots, \tau_N \)

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=0}^{T_i} \gamma^t r_{i,t} \right) \left( \sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \right) \]
REINFORCE algorithm

1. Sample $N$ trajectories $\tau_i$ using current policy $\pi_\theta$

2. Estimate the policy gradient:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i) \left( \sum_{t=0}^{T_i} \nabla_\theta \log \pi_\theta(s_{i,t}, a_{i,t}) \right)$$

3. Update parameters by gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

Williams et al. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. Machine Learning, 8(3):229-256, 1992
REINFORCE: Single-step version

1. In state $s$, sample action $a$ using current policy $\pi_\theta$, observe reward $r$

2. Estimate the policy gradient:
$$\nabla_\theta J(\theta) \approx r \nabla_\theta \log \pi_\theta(s, a)$$

3. Update parameters by gradient ascent:
$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

• What effect does this update have?
  • Push up the probability of good actions, push down probability of bad actions
Reducing variance

- Gradient estimate (for a single trajectory):
  \[ \nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) \]

- **General problem**: rewards of sampled trajectories are too noisy and lead to unreliable policy gradients
Reducing variance

- Gradient estimate (for a single trajectory):

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(s_t, a_t) \]

- Observation: it seems bad to weight each action in a trajectory by the return of the entire trajectory. In particular, rewards obtained before an action was taken should not be used to weight that action.

- Instead, for each action, consider only the cumulative future reward:

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(s_t, a_t) \]
Reducing variance

\[ \nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} \left( \sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) \]

Observed cumulative reward after taking action \( a_t \) in state \( s_t \)

- But then, why not use expected cumulative reward?

\[ \nabla_{\theta} J(\theta) \approx \sum_{t \geq 0} Q^{\pi_{\theta}}(s_t, a_t) \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) \]
Actor-Critic algorithm

- Combine policy gradients and Q-learning by simultaneously training an *actor* (the policy) and a *critic* (the Q-function)

Source: D. Silver
Reducing variance

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} Q^{\pi_\theta}(s_t, a_t) \nabla_\theta \log \pi_\theta(s_t, a_t) \]

- Next observation: the raw Q-values are not the most useful. If all Q-values are good, we will try to push up the probabilities of all the actions
- Instead, introduce a baseline function to represent whether the current value is better or worse than what we expect to get

\[ \nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left( Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \right) \nabla_\theta \log \pi_\theta(s_t, a_t) \]

Advantage function
Estimating the advantage function

- Advantage function:

\[
A^{\pi_\theta}(s, a) = Q^{\pi_\theta}(s, a) - V^{\pi_\theta}(s) \\
= \mathbb{E}_{\pi_\theta}[r + \gamma V^{\pi_\theta}(s')|s, a] - V^{\pi_\theta}(s) \\
\approx r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)
\]

(from a single transition)

- Therefore, it is sufficient to learn the value function:

\[
V^{\pi_\theta}(s) \approx V_w(s)
\]
Online actor-critic algorithm

1. Sample action $a$ using current policy, observe reward $r$, successor state $s'$
2. Update $V_w(s)$ towards target $r + \gamma V_w(s')$
3. Estimate $A^{\pi_\theta}(s, a) = r + \gamma V_w(s') - V_w(s)$
4. Estimate $\nabla_\theta J(\theta) = A^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(s, a)$
5. Update policy parameters: $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$

Source: Berkeley RL course
Asynchronous advantage actor-critic (A3C)

\[ V, \pi \]

Agent 1 \rightarrow Experience 1 \rightarrow Local updates
Agent 2 \rightarrow Experience 2 \rightarrow Local updates
\ldots
Agent n \rightarrow Experience n \rightarrow Local updates

Asynchronously update global parameters

## Asynchronous advantage actor-critic (A3C)

<table>
<thead>
<tr>
<th>Method</th>
<th>Training Time</th>
<th>Mean</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN</td>
<td>8 days on GPU</td>
<td>121.9%</td>
<td>47.5%</td>
</tr>
<tr>
<td>Gorila</td>
<td>4 days, 100 machines</td>
<td>215.2%</td>
<td>71.3%</td>
</tr>
<tr>
<td>D-DQN</td>
<td>8 days on GPU</td>
<td>332.9%</td>
<td>110.9%</td>
</tr>
<tr>
<td>Dueling D-DQN</td>
<td>8 days on GPU</td>
<td>343.8%</td>
<td>117.1%</td>
</tr>
<tr>
<td>Prioritized DQN</td>
<td>8 days on GPU</td>
<td>463.6%</td>
<td>127.6%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>1 day on CPU</td>
<td>344.1%</td>
<td>68.2%</td>
</tr>
<tr>
<td>A3C, FF</td>
<td>4 days on CPU</td>
<td>496.8%</td>
<td>116.6%</td>
</tr>
<tr>
<td>A3C, LSTM</td>
<td>4 days on CPU</td>
<td>623.0%</td>
<td>112.6%</td>
</tr>
</tbody>
</table>

Mean and median human-normalized scores over 57 Atari games

Asynchronous advantage actor-critic (A3C)

TORCS car racing simulation video

Asynchronous advantage actor-critic (A3C)

Motor control tasks video

Benchmarks and environments for Deep RL

OpenAI Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

View documentation
View on GitHub

https://gym.openai.com/