Deep Reinforcement Learning and Control

Policy gradients

CMU 10-403

Katerina Fragkiadaki
Used Materials

- **Disclaimer**: Much of the material and slides for this lecture were borrowed from Russ Salakhutdinov, Rich Sutton’s class and David Silver’s class on Reinforcement Learning.
Revision
Deep Q-Networks (DQNs)

- Represent action-state value function by Q-network with weights $w$

$$Q(s, a, w) \approx Q^*(s, a)$$
Cost function

- Minimize mean-squared error between the true action-value function \( q_{\pi}(S,A) \) and the approximate Q function:

\[
J(w) = \mathbb{E}_\pi \left[ (q_{\pi}(S,A) - Q(S,A,w))^2 \right]
\]

- We do not know the groundtruth value

- Minimize MSE loss by stochastic gradient descent

\[
\mathcal{L} = \left( r + \gamma \max_{a'} Q(s,a',w) - Q(s,a,w) \right)^2
\]

wrong!
Cost function

- Minimize mean-squared error between the true action-value function $q_\pi(S, A)$ and the approximate Q function:

$$J(w) = \mathbb{E}_\pi \left[ (q_\pi(S, A) - Q(S, A, w))^2 \right]$$

- We do not know the groundtruth value

- Minimize MSE loss by stochastic gradient descent

$$\mathcal{L} = \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2$$
Q-Learning: Off-Policy TD Control

- One-step Q-learning:

\[
Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[ R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right]
\]

Initialize \( Q(s, a), \forall s \in S, a \in A(s) \), arbitrarily, and \( Q(\text{terminal-state}, \cdot) = 0 \)

Repeat (for each episode):
  - Initialize \( S \)
  - Repeat (for each step of episode):
    - Choose \( A \) from \( S \) using policy derived from \( Q \) (e.g., \( \varepsilon \)-greedy)
    - Take action \( A \), observe \( R, S' \)
    - \( Q(S, A) \leftarrow Q(S, A) + \alpha \left[ R + \gamma \max_a Q(S', a) - Q(S, A) \right] \)
    - \( S \leftarrow S' \)
  - until \( S \) is terminal
Stability of training problems for DQN

- Minimize MSE loss by stochastic gradient descent
  \[ \mathcal{L} = \left( r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w) \right)^2 \]
- Converges to \( Q^* \) using table lookup representation
- But diverges using neural networks due to:
  1. Correlations between samples
  2. Non-stationary targets
- Solutions:
  1. Experience buffer
  2. Targets stay fixed for many iterations
Learning a DQN supervised from a planner

- Minimize MSE loss by stochastic gradient descent

\[ L = \left( Q_{MCTS}(s, a) - Q(s, a, w) \right)^2 \]

- Boils down to a supervised learning problem

- I use MCTS to play 800 games, I gather the Q estimates of states and actions in the MCTS trees and train a regressor.

- Any problems?

- Any solutions?

- DAGGER!
Learning a DQN supervised from a planner

- Minimize MSE loss by stochastic gradient descent
  \[ \mathcal{L} = \left( Q_{MCTS}(s, a) - Q(s, a, w) \right)^2 \]

- Boils down to a supervised learning problem

- I use MCTS to play 800 games, I gather the Q estimates of states and actions in the MCTS trees and train a regressor. Then use it to find a policy

- Any problems?

- Any solutions?

- DAGGER!

- Also: training a classifier directly worked best!
Policy-Based Reinforcement Learning

- So far we approximated the value or action-value function using parameters $\theta$ (e.g. neural networks)

$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

- A policy was generated directly from the value function e.g. using $\epsilon$-greedy

- In this lecture we will directly parameterize the policy

$$\pi_\theta(s, a) = \mathbb{P}[a \mid s, \theta]$$

- We will not use any models, and we will learn from experience, not imitation
Policy-Based Reinforcement Learning

- So far we approximated the value or action-value function using parameters $\theta$ (e.g. neural networks)

$$V_\theta(s) \approx V^\pi(s)$$
$$Q_\theta(s, a) \approx Q^\pi(s, a)$$

- A policy was generated directly from the value function e.g. using $\epsilon$-greedy

- In this lecture we will directly parameterize the policy
  
  $$\pi(A_t|S_t, \theta)$$

- We will focus again on model-free reinforcement learning

Sometimes I will also use the notation:

$$\pi_\theta(s, a) = \mathbb{P}[a | s, \theta]$$
Value-Based and Policy-Based RL

- **Value Based**
  - Learned Value Function
  - Implicit policy (e.g. $\epsilon$-greedy)

- **Policy Based**
  - No Value Function
  - Learned Policy

- **Actor-Critic**
  - Learned Value Function
  - Learned Policy
Advantages of Policy-Based RL

- Advantages
  - Effective in high-dimensional or continuous action spaces
  - Can learn stochastic policies
  - We will look into the benefits of stochastic policies in a future lecture
Policy function approximators

With continuous policy parameterization the action probabilities change smoothly as a function of the learned parameter, whereas in epsilon-greedy selection the action probabilities may change dramatically for an arbitrarily small change in the estimated action values, if that change results in a different action having the maximal value.
Policy function approximators

**Deterministic continuous policy**

\[ a = \pi_\theta(s) \]

**Stochastic continuous policy**

\[ a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s)) \]

**Discrete actions**

- go left
- go right

Output is a distribution over a discrete set of actions
Policy Objective Functions

- **Goal**: given policy $\pi_{\theta}(s, a)$ with parameters $\theta$, find best $\theta$
- But how do we measure the quality of a policy $\pi_{\theta}$?
- In episodic environments we can use the start value
  $$J_1(\theta) = V^{\pi_{\theta}}(s_1) = \mathbb{E}_{\pi_{\theta}}[v_1]$$
- In continuing environments we can use the average value
  $$J_{avV}(\theta) = \sum_s d^{\pi_{\theta}}(s) V^{\pi_{\theta}}(s)$$
- Or the average reward per time-step
  $$J_{avR}(\theta) = \sum_s d^{\pi_{\theta}}(s) \sum_a \pi_{\theta}(s, a) R_s^a$$

where $d^{\pi_{\theta}}(s)$ is stationary distribution of Markov chain for $\pi_{\theta}$
Policy Objective Functions

- **Goal**: given policy \( \pi_\theta(s,a) \) with parameters \( \theta \), find best \( \theta \)

- But how do we measure the quality of a policy \( \pi_\theta \)?

- In continuing environments we can use the average value

\[
J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s)V^{\pi_\theta}(s)
\]

- In the episodic case, \( d^{\pi_\theta}(s) \) is defined to be
  - the expected number of time steps \( t \) on which \( S_t = s \)
  - in a randomly generated episode starting in \( s_0 \) and
  - following \( \pi \) and the dynamics of the MDP.

Remember: Episode of experience under policy \( \pi \):
\[
S_1, A_1, R_2, ..., S_k \sim \pi
\]
Policy Optimization

- Policy based reinforcement learning is an optimization problem
  - Find $\theta$ that maximizes $J(\theta)$
- Some approaches do not use gradient
  - Hill climbing
  - Genetic algorithms
- Greater efficiency often possible using gradient
- We focus on gradient descent, many extensions possible
- And on methods that exploit sequential structure
Policy Gradient

- Let $J(\theta)$ be any policy objective function.

- Policy gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy, w.r.t. parameters $\theta$.

  $\Delta \theta = \alpha \nabla_{\theta} J(\theta)$

  $\alpha$ is a step-size parameter (learning rate)

  is the policy gradient

  $\nabla_{\theta} J(\theta) = \left( \frac{\partial J(\theta)}{\partial \theta_1} \right) \ldots \left( \frac{\partial J(\theta)}{\partial \theta_n} \right)$
Computing Gradients By Finite Differences

- To evaluate policy gradient of $\pi_\theta(s, a)$
- For each dimension $k$ in $[1, n]$:
  - Estimate $k^{th}$ partial derivative of objective function w.r.t. $\theta$
  - By perturbing $\theta$ by small amount $\epsilon$ in $k^{th}$ dimension

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

where $u_k$ is a unit vector with 1 in $k^{th}$ component, 0 elsewhere

- Uses $n$ evaluations to compute policy gradient in $n$ dimensions
- Simple, noisy, inefficient - but sometimes effective
- Works for arbitrary policies, even if policy is not differentiable
Learning an AIBO running policy

- Goal: learn a fast AIBO walk (useful for Robocup)
- AIBO walk policy is controlled by 12 numbers (elliptical loci)
- Adapt these parameters by finite difference policy gradient
- Evaluate performance of policy by field traversal time
Learning an AIBO running policy

Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion, Kohl and Stone, 2004

\[
\pi \leftarrow \text{InitialPolicy}
\]

\[
\text{while } \neg \text{done do}
\]
\[
\{R_1, R_2, \ldots, R_t\} = t \text{ random perturbations of } \pi
\]
\[
\text{evaluate}( \{R_1, R_2, \ldots, R_t\})
\]
\[
\text{for } n = 1 \text{ to } N \text{ do}
\]
\[
\text{Avg}_{+,n} \leftarrow \text{average score for all } R_i \text{ that have a positive perturbation in dimension } n
\]
\[
\text{Avg}_{+,0,n} \leftarrow \text{average score for all } R_i \text{ that have a zero perturbation in dimension } n
\]
\[
\text{Avg}_{-,n} \leftarrow \text{average score for all } R_i \text{ that have a negative perturbation in dimension } n
\]
\[
\text{if } \text{Avg}_{+,0,n} > \text{Avg}_{+,n} \text{ and } \text{Avg}_{+,0,n} > \text{Avg}_{-,n} \text{ then}
\]
\[
A_n \leftarrow 0
\]
\[
\text{else}
\]
\[
A_n \leftarrow \text{Avg}_{+,n} - \text{Avg}_{-,n}
\]
\[
\text{end if}
\]
\[
\text{end for}
\]
\[
A \leftarrow \frac{A}{|A|} \ast \eta
\]
\[
\pi \leftarrow \pi + A
\]
\[
\text{end while}
\]
Learning an AIBO running policy

Initial

Training

Final
Policy Gradient: Score Function

- We now compute the policy gradient analytically.

- Assume
  - policy $\pi_\theta$ is differentiable whenever it is non-zero
  - we know the gradient $\nabla_\theta \pi_\theta(s, a)$

- Likelihood ratios exploit the following identity

\[
\nabla_\theta \pi_\theta(s, a) = \pi_\theta(s, a) \frac{\nabla_\theta \pi_\theta(s, a)}{\pi_\theta(s, a)}
\]

\[
= \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a)
\]

- The score function is $\nabla_\theta \log \pi_\theta(s, a)$
Softmax Policy: Discrete Actions

- We will use a softmax policy as a running example.
- Weight actions using linear combination of features $\phi(s, a)^\top \theta$.
- Probability of action is proportional to exponentiated weight:
  \[
  \pi_\theta(s, a) \propto e^{\phi(s, a)^\top \theta}
  \]

Nonlinear extension: replace $\phi(s, a)$ with a deep neural network with trainable weights $w$.

Think a neural network with a softmax output probabilities.
Softmax Policy: Discrete Actions

- We will use a softmax policy as a running example
- Weight actions using linear combination of features $\phi(s, a) \top \theta$
- Probability of action is proportional to exponentiated weight
  \[ \pi_\theta(s, a) \propto e^{\phi(s,a) \top \theta} \]
  
  Nonlinear extension: replace $\phi(s, a)$ with a deep neural network with trainable weights $w$
  Think a neural network with a softmax output probabilities
- The score function is
  \[ \nabla_\theta \log \pi_\theta(s, a) = \phi(s, a) - E_{\pi_\theta} [\phi(s, \cdot)] \]
Gaussian Policy: Continuous Actions

- In **continuous action spaces**, a Gaussian policy is natural.
- Mean is a linear combination of state features:
  \[ \mu(s) = \phi(s)^\top \theta \]
  **Nonlinear extension:** replace \( \phi(s) \) with a deep neural network with trainable weights \( w \).
- Variance may be fixed \( \sigma^2 \), or can also **parameterized**.
- Policy is Gaussian:
  \[ a \sim \mathcal{N}(\mu(s), \sigma^2) \]
- The **score function** is:
  \[ \nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s))\phi(s)}{\sigma^2} \]
One-step MDP

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = \mathcal{R}_{s,a}$

First, let’s look at the objective:

$$J(\theta) = \mathbb{E}_{\pi_\theta} [r]$$

$$= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) \mathcal{R}_{s,a}$$

**Intuition:** Under MDP:

$$\mathbb{E}_{\pi_\theta} [r] = \sum_{s \in S} \sum_{a \in A} P_\theta(s, a) \mathcal{R}_{s,a} = \sum_{s \in S} \sum_{a \in A} P(s) \pi_\theta(a|s) \mathcal{R}_{s,a}$$

$$= \sum_{s \in S} P(s) \sum_{a \in A} \pi_\theta(a|s) \mathcal{R}_{s,a}$$
One-step MDP

- Consider a simple class of one-step MDPs
  - Starting in state $s \sim d(s)$
  - Terminating after one time-step with reward $r = R_{s,a}$
- Use likelihood ratios to compute the policy gradient

$$J(\theta) = \mathbb{E}_{\pi_\theta} [r]$$
$$= \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) R_{s,a}$$

$$\nabla_\theta J(\theta) = \sum_{s \in S} d(s) \sum_{a \in A} \pi_\theta(s, a) \nabla_\theta \log \pi_\theta(s, a) R_{s,a}$$
$$= \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a)r]$$