Introduction to Reinforcement Learning and Policy-Gradients with Tensor-Flow

Frederik Ebert (UC Berkeley)
Stanford CS 20, 03-07-2018

Slides adapted from (Berkeley CS 294: Deep Reinforcement Learning by Sergey Levine)
Why Reinforcement Learning?
Today’s Lecture

1. Definition of reinforcement learning problem
2. Brief overview of RL algorithm types
3. Introduction to policy gradient algorithms
4. Implementation of policy gradient algorithms in TF

• Goals:
  • Understand definitions & notation
  • Get an overview of different reinforcement learning algorithms
  • Understand how the policy gradient RL-algorithm can be implemented in TF
Definitions
Terminology & notation

- $s_t$ – state
- $o_t$ – observation
- $a_t$ – action

$\pi_\theta(a_t|o_t)$ – policy
$\pi_\theta(a_t|s_t)$ – policy (fully observed)

Markov property independent of $s_{t-1}$
Reward functions

which action is better or worse?

$r(s, a)$: reward function

tells us which states and actions are better

$s, a, r(s, a)$, and $p(s' | s, a)$ define Markov decision process
Definitions

partially observed Markov decision process \( M = \{ \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r \} \)

\( \mathcal{S} \) – state space  
states \( s \in \mathcal{S} \) (discrete or continuous)

\( \mathcal{A} \) – action space  
actions \( a \in \mathcal{A} \) (discrete or continuous)

\( \mathcal{O} \) – observation space  
observations \( o \in \mathcal{O} \) (discrete or continuous)

\( \mathcal{T} \) – transition operator (like before)

\( \mathcal{E} \) – emission probability \( p(o_t | s_t) \)

\( r \) – reward function  
\( r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \)
Expectations and stochastic systems

\[
\theta^* = \arg \max_\theta E_{(s,a) \sim p_\theta(s,a)}[r(s, a)]
\]

infinite horizon case

\[
\theta^* = \arg \max_\theta \sum_{t=1}^T E_{(s_t,a_t) \sim p_\theta(s_t,a_t)}[r(s_t, a_t)]
\]

finite horizon case

In RL, we almost always care about *expectations*

\[r(s,a) - not \text{ smooth}\]

\[\psi - \text{probability of falling}\]

\[E_{(s,a) \sim p_\psi(s,a)}[r(s, a)] - \text{smooth in } \psi!\]
Algorithms
Types of RL algorithms

\[ \theta^* = \arg \max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

- Policy gradients: directly differentiate the above objective
- Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)
- Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy
- Model-based RL: estimate the transition model, and then...
  - Use it for planning (no explicit policy)
  - Use it to improve a policy
  - Something else
Direct policy gradients

- Generate samples (i.e. run the policy)
- Estimate the return
  \[ R_\tau = \sum_t r(s_t, a_t) \]
- Evaluate returns
- Improve the policy
  \[ \theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(s_t, a_t)] \]
Value function based algorithms

Examples:
• Value-Iteration
• Q-Learning
• DQN

1. **generate samples** (i.e. run the policy)
2. **improve the policy**
   
   set $\pi(s) = \text{arg max}_a Q(s, a)$

3. **fit a model/estimate the return**
   
   fit $V(s)$ or $Q(s, a)$
Actor-critic: value functions + policy gradients

1. **generate samples (i.e. run the policy)**
2. **fit a model/estimate the return**
   - fit $V(s)$ or $Q(s, a)$
   - evaluate returns using $V$ or $Q$
3. **improve the policy**
   - $\theta \leftarrow \theta + \alpha \nabla_{\theta} E[\sum_t r(s_t, a_t)]$
Model-based RL algorithms

- generate samples (i.e. run the policy)
- fit a model
- improve the policy
- a few options
Comparison: sample efficiency

- Sample efficiency = how many samples do we need to get a good policy?
- Most important question: is the algorithm off policy?
  - Off policy: able to improve the policy without generating new samples from that policy
  - On policy: each time the policy is changed, even a little bit, we need to generate new samples

\[
\theta \leftarrow \theta + \alpha \nabla_\theta E[\sum_t r(s_t, a_t)]
\]

just one gradient step
Comparison: sample efficiency

Why would we use a less efficient algorithm?

Wall clock time is not the same as efficiency!
Comparison: stability and ease of use

• Value function fitting
  • At best, minimizes error of fit (“Bellman error”)
    • Not the same as expected reward
  • At worst, doesn’t optimize anything
    • Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case

• Model-based RL
  • Model minimizes error of fit
    • This will converge
  • No guarantee that better model = better policy

• Policy gradient
  • The only one that actually performs gradient descent (ascent) on the true objective
Example: Robotic Manipulation with value function based algorithm

For detail see the Normalized Advantage Function (NAF) algorithm
Introduction to Policy Gradients
Evaluating the objective

$$\theta^* = \arg \max_{\theta} E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right]$$

$$J(\theta)$$

$$J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t})$$

sum over samples from $\pi_\theta$
Direct policy differentiation

\[ \theta^* = \arg \max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \]

\[ J(\theta) = \sum_{t=1}^{T} r(s_t, a_t) \]

\[ J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[r(\tau)] = \int_{0}^{\tau} \pi_\theta(\tau)r(\tau)d\tau \]

\[ \nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau)r(\tau)d\tau = \int \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)r(\tau)d\tau = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau)r(\tau)] \]

a convenient identity

\[ \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau) = \pi_\theta(\tau) \frac{\nabla_\theta \pi_\theta(\tau)}{\pi_\theta(\tau)} = \nabla_\theta \pi_\theta(\tau) \]
Evaluating the policy gradient

recall: \( J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[ \sum_t r(s_t, a_t) \right] \approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t}) \)

\[
\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left( \sum_{t=1}^T r(s_t, a_t) \right) \right]
\]

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^N \left( \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right) \left( \sum_{t=1}^T r(s_{i,t}, a_{i,t}) \right)
\]

\[
\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)
\]

REINFORCE algorithm:

1. sample \( \{\tau^i\} \) from \( \pi_\theta(a_t | s_t) \) (run the policy)
2. \( \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a^i_t | s^i_t) \right) \left( \sum_t r(s^i_t, a^i_t) \right) \)
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)
Example: Gaussian policies

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \]

example: \[ \pi_\theta(a_t|s_t) = \mathcal{N}(f_{\text{neural network}}(s_t); \Sigma) \]

\[ \log \pi_\theta(a_t|s_t) = -\frac{1}{2} \| f(s_t) - a_t \|_\Sigma^2 + \text{const} \]

\[ \nabla_\theta \log \pi_\theta(a_t|s_t) = -\frac{1}{2} \Sigma^{-1} (f(s_t) - a_t) \frac{df}{d\theta} \]

REINFORCE algorithm:

1. sample \{\tau^i\} from \pi_\theta(a_t|s_t) (run it on the robot)
2. \[ \nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a^i_t|s^i_t) \right) \left( \sum_t r(s^i_t, a^i_t) \right) \]
3. \[ \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \]
What did we just do?

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t}) \right) \left( \sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right)
\]

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t} \nabla_\theta \log \pi_\theta(\tau_i) r(\tau_i) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} | s_{i,t})
\]

maximum likelihood: \[
\nabla_\theta J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau_i)
\]

good stuff is made more likely
bad stuff is made less likely
simply formalizes the notion of “trial and error”!

REINFORCE algorithm:

1. sample \( \{\tau^i\} \) from \( \pi_\theta(a_t | s_t) \) (run it on the robot)
2. \[
\nabla_\theta J(\theta) \approx \sum_i \left( \sum_t \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right) \left( \sum_t r(s_t^i, a_t^i) \right)
\]
3. \( \theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \)
Reducing variance

\[ \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left( \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) \right) \left( \sum_{i=1}^{T} r(s_{i,t}, a_{i,t}) \right) \]

What you do now does **not** affect the reward of the past!

\[ \nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}(a_{i,t}|s_{i,t}) \left( \sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'}) \right) \]

“reward to go”

\[ \hat{Q}_{i,t} \]


**Baselines**

\[
\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau)[h(\tau_i) - b]
\]

\[
b = \frac{1}{N} \sum_{i=1}^{N} r(\tau)
\]

but... are we *allowed* to do that??

\[
E[\nabla_\theta \log \pi_\theta(\tau)b] = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau)b \, d\tau = b \int \nabla_\theta \pi_\theta(\tau) b \, d\tau = b \int \nabla_\theta 1 = 0
\]

subtracting a baseline is *unbiased* in expectation!

average reward is *not* the best baseline, but it’s pretty good!

\[
\pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) = \nabla_\theta \pi_\theta(\tau)
\]
Implementation of Policy Gradients
Policy gradient with automatic differentiation

\[ \nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \hat{Q}_{i,t} \]

pretty inefficient to compute these explicitly!

How can we compute policy gradients with automatic differentiation?

We need a graph such that its gradient is the policy gradient!

maximum likelihood: \[ \nabla_\theta J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \quad J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(a_{i,t}|s_{i,t}) \]

Just implement “pseudo-loss” as a weighted maximum likelihood:

\[ \tilde{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(a_{i,t}|s_{i,t}) \hat{Q}_{i,t} \]

\[ \text{cross entropy (discrete) or squared error (Gaussian)} \]
Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# rew_to_go - (N*T) x 1 tensor of estimated reward to go
# Build the graph:
logits = policy.predictions(states)  # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, rew_to_go)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

\[
\hat{J}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(a_{i,t}|s_{i,t}) Q_{i,t} \quad \text{Reward to go}
\]
Policy gradient with automatic differentiation

Pseudocode example (with discrete actions):

Policy gradient:

# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# rew_to_go - (N*T) x 1 tensor of estimated reward to go
# Build the graph:
mean = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = gaussian_log_prob(sy_ac_na, mean, sy_logstd)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, rew_to_go)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)

log πθ(a_t|s_t) = −1/2∥f(s_t) − a_t∥2 + const

\dot{J}(θ) ≈ \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log πθ(a_{i,t}|s_{i,t})Q_{i,t}

Reward to go
Policy gradient in practice

• Remember that the gradient has high variance
  • This isn’t the same as supervised learning!
  • Gradients will be really noisy!

• Consider using much larger batches

• Tweaking learning rates is very hard
  • Adaptive step size rules like ADAM can be OK-ish
  • There exist algorithms that adjust the gradient stepsize to obtain more stability, such as Trust-Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO)
Suggested Project

• Implement policy gradient as in homework 2 of CS 294: DeepRL, Fall 2017
  • Vanilla policy gradient algorithm in Tensorflow
  • Add baseline for variance reduction
  • Agents trained for Inverted Pendulum and Cheetah environments
    (for Cheetah Mujoco physics engine necessary, 30 day trial license available)
  • Most of the code is prepared, you only need to fill in a couple of blanks

The material was prepared by Abhishek Gupta and Josh Aicham.
Example: trust region policy optimization, policies initialized from demonstration

• Natural gradient with automatic step adjustment
• Discrete and continuous actions
• Using a small number of demonstrations to overcome exploration problem.
Beyond RL: Self-supervised Learning with Video-Prediction and Sampling Based Planning

Self-Supervised Visual Planning with Temporal Skip-Connections, Ebert et al. 2017
Policy gradients suggested lectures and readings

- Lectures online: Berkeley [CS 294](https://example.com), Course at UCL by David Silver
- Classic papers
  - Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm
  - Baxter & Bartlett (2001). Infinite-horizon policy-gradient estimation: temporally decomposed policy gradient (not the first paper on this! see actor-critic section later)
- Deep reinforcement learning policy gradient papers
  - Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient (unrelated to later discussion of guided policy search)