10703 Deep Reinforcement Learning and Control

Russ Salakhutdinov

Machine Learning Department
rsalakhu@cs.cmu.edu

Slides developed and borrowed from Katerina Fragkiadaki

End-to-end Model Based Reinforcement Learning
End-to-end policy optimization through back-propagation
Last week: Trajectory optimization
Last week: Trajectory optimization

\[
\min_{u^0 \ldots u^T} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t)
\]
Last week: Trajectory optimization

\[
\min_{u^0 \ldots u^T} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t) \quad \|x_t - x^*\|
\]
Last week: Trajectory optimization

\[
\min_{u^0 \ldots u^T} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t)
\]
Poor Conditioning

\[
\min_{u_1, \ldots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \cdots \\
\cdots + c(f(f(\cdots)\cdots), u_T)
\]
Poor Conditioning

\[
\min_{\mathbf{u}_1, \ldots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \cdots \\
\cdots + c(f(\cdots \cdots), \mathbf{u}_T)
\]
Poor Conditioning

\[
\min_{u_0 \ldots u_T} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t)
\]
Poor Conditioning

$$\min_{u^0 \ldots u^T} \sum_{t=1}^T c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t)$$
Poor Conditioning

$$\min_{u^0 \ldots u^T} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t)$$
Consider the special case of quadratic $c$ and linear $f$
Trajectory Optimization

- Consider the special case of quadratic $c$ and linear $f$

\[
\min_{u^0 \ldots u^T} \sum_{t=1}^{\infty} c(x_t, u_t) \quad x^{t+1} = f(x^t, u^t)
\]

Solve it using dynamic programming:

- write $u^*_t$ as function of the state $x_t$ at each $t = T, \ldots, 1$
- substitute $x_0$ (known)
- for $t = 1, \ldots, T$ substitute $x_t$ into $u^*_t$ and fire the dynamics forward to obtain next state state ($x_{t+1} = f(x_t, u^*_t)$)
Learning Control Policies

\[ \pi_{\theta} : x \mapsto u \]
Learning Control Policies

\[ \pi_\theta : x \mapsto u \]

\[ \min_{\theta} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, \pi_\theta(x^t)) \]
Learning Control Policies

\[ \pi_\theta : \mathbf{x} \mapsto \mathbf{u} \]

\[
\min_{\theta} \sum_{t=1}^{T} c(x_t, u_t) \quad x^{t+1} = f(x^t, \pi_\theta(x^t))
\]
So far, dynamics are assumed known and deterministic.
Policy is assumed deterministic.
We solve for policy parameters $\theta$ using back-propagating (through time).
So far, dynamics are assumed known and deterministic.

Policy is assumed deterministic.

We solve for policy parameters $\theta$ using back-propagating (through time).
Learning Control Policies through Backpropagation

- Dynamics are unknown.
- Policy is assumed deterministic.
- We alternate solving for dynamics parameters $\phi$ (standard regression) and solving for policy parameters $\theta$ using back-propagating (through time).

\[
\min_{\theta} \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))
\]
Learning Control Policies through Backpropagation

1. run base policy $\pi_0(u_t|x_t)$ (e.g., random policy) to collect $D = \{(x, u, x')_i\}$
Learning Control Policies through Backpropagation

1. run base policy $\pi_0(u_t|x_t)$ (e.g., random policy) to collect $D = \{(x, u, x')_i\}$
2. learn dynamics model $f_\phi(x, u)$ to minimize $\sum_i \|f_\phi(x_i, u_i) - x'_i\|^2$
Learning Control Policies through Backpropagation

1. run base policy $\pi_0(u_t|x_t)$ (e.g., random policy) to collect $D = \{(x, u, x')_i\}$

2. learn dynamics model $f_\phi(x, u)$ to minimize \(\sum_i ||f_\phi(x_i, u_i) - x'_i||^2\)
Learning Control Policies through Backpropagation

1. run base policy \( \pi_0(u_t|x_t) \) (e.g., random policy) to collect \( \mathcal{D} = \{(x, u, x')_i\} \)
2. learn dynamics model \( f_\phi(x, u) \) to minimize \( \sum_i \|f_\phi(x_i, u_i) - x'_i\|^2 \)
3. backpropagate through \( f_\phi(x, u) \) into the policy to optimize \( \pi_\theta(u_t|x_t) \)

while dynamics are frozen
Learning Control Policies through Backpropagation

1. run base policy \( \pi_0(u_t|x_t) \) (e.g., random policy) to collect \( D = \{(x, u, x')_i\} \)

2. learn dynamics model \( f_\phi(x, u) \) to minimize \( \sum_i ||f_\phi(x_i, u_i) - x'_i||^2 \)

3. backpropagate through \( f_\phi(x, u) \) into the policy to optimize \( \pi_\theta(u_t|x_t) \)

4. run \( \pi_\theta(u_t|x_t) \), appending the visited tuples \( (x, u, x') \) to \( D \)
Challenges:

- Poor conditioning
- $\theta$ couples actions across all steps -> no DP
- Chaining inaccurate dynamics naturally leads to errors

Learning Control Policies through Backpropagation

$$\min_{\theta} \sum_{t=1}^{T} c(x_t, \pi_{\theta}(x_t))$$
Learning Control Policies through Imitation

$$\pi_\theta : x \mapsto u$$

Learning from Demonstrations:

supervised learning

$$\min_\theta \sum_i \|\pi_\theta(x^i) - u^i\|^2$$

Training Data

input: $x^i$
output: $u^i$
Learning Control Policies through Imitation

\[ \pi_\theta : x \mapsto u \]

Learning from Demonstrations:

supervised learning

\[
\min_\theta \sum_i \| \pi_\theta(x^i) - u^i \|^2
\]

Training Data

- input: \( x^i \)
- output: \( u^i \)

Where does training data come from?

Optimal controllers trained with trajectory optimization
Learning Control Policies through Imitation

• Joint trajectory and policy optimization (last week):

\[
\min_{u_1, \ldots, u_T, x_1, \ldots, x_T, \theta} \sum_{t=1}^{T} c(x_t, u_t) \quad \text{s.t.} \quad x_t = f(x_{t-1}, u_{t-1})
\]

\[
s.t. \quad u_t = \pi_\theta(x_t)
\]
Learning Control Policies through Backpropagation

Challenges:

• Backproping through **stochastic** policies and **stochastic** environments
• Avoiding error accumulation through chaining of one step dynamics

\[
\min_{\theta} \sum_{t=1}^{T} c(x_t, \pi_\theta(x_t))
\]
This Lecture

• Backproping through stochastic policies and stochastic environments
  • Re-parametrization trick

• Avoiding error accumulation through chaining of one step dynamics
  • Use function approximation for action and state value functions to predict future returns so that you do not rely on your model for long chaining
Policy optimization

- You get a reward when the jeannie appears
Policy optimization

- You get a reward when the jeannie appears
Policy Optimization

maximize $\mathbb{E}_\pi$ [expression]

Fixed-horizon episodic: $\sum_{t=0}^{T-1} r_t$

Average-cost: $\lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} r_t$

Infinite-horizon discounted: $\sum_{t=0}^{\infty} \gamma^t r_t$

Variable-length undiscounted: $\sum_{t=0}^{T_{\text{terminal}}-1} r_t$

Infinite-horizon undiscounted: $\sum_{t=0}^{\infty} r_t$
Episodic Settings

\[ s_0 \sim \mu(s_0) \]
\[ a_0 \sim \pi(a_0 \mid s_0) \]
\[ s_1, r_0 \sim P(s_1, r_0 \mid s_0, a_0) \]
\[ a_1 \sim \pi(a_1 \mid s_1) \]
\[ s_2, r_1 \sim P(s_2, r_1 \mid s_1, a_1) \]
\[ \ldots \]
\[ a_{T-1} \sim \pi(a_{T-1} \mid s_{T-1}) \]
\[ s_T, r_{T-1} \sim P(s_T \mid s_{T-1}, a_{T-1}) \]

Objective:

\[ \text{maximize } \eta(\pi), \text{ where } \]
\[ \eta(\pi) = E[r_0 + r_1 + \cdots + r_{T-1} \mid \pi] \]

\( \mathcal{T} \): trajectory, a sequence of action states
Parameterized Policies

- A family of policies indexed by parameter vector $\theta \in \mathbb{R}^d$
  - Deterministic: $a = \pi(s, \theta)$
  - Stochastic: $\pi(a|s, \theta)$
- Analogous to classification or regression with input $s$, output $a$
  - Discrete action space: network outputs vector of probabilities
  - Continuous actions space: network outputs mean and diagonal covariance of Gaussian
Parametrized policies

deterministic policy

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

- **Deterministic policy**
  \[ a = \pi_\theta(s) \]

- **Stochastic continuous policy:** usually unimodal gaussian
  \[ a \sim \mathcal{N}(\mu_\theta(s), \sigma_\theta^2(s)) \]
Parametrized policies

**Deterministic policy**

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

**Stochastic continuous policy:**

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

**Discrete action space**

- go left
- go right
How do we compute gradients?

\[ \nabla_\theta \mathbb{E} [R_T] \]

- Numerically: finite differencing
How do we compute gradients?

\[ \nabla_\theta \mathbb{E} [R_T] \]

- Numerically: finite differencing
- Score function gradient estimator (a.k.a. likelihood ratio gradient estimator)
Likelihood Ratio Policy Gradient

\[ J(\theta) = \sum \tau P[t; \theta]R(\tau) \]

Taking the gradient w.r.t. \( \theta \) gives

\[ \nabla_\theta J(\theta) = \nabla_\theta \sum \tau P[\tau; \theta]R(\tau) \]
Likelihood Ratio Policy Gradient

\[ J(\theta) = \sum_{\tau} P[t; \theta] R(\tau) \]

Taking the gradient w.r.t. \( \theta \) gives

\[ \nabla_\theta J(\theta) = \nabla_\theta \sum_{\tau} P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} \nabla_\theta P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_\theta P[\tau; \theta] R(\tau) \]
Likelihood Ratio Policy Gradient

\[ J(\theta) = \sum_{\tau} P[t; \theta] R(\tau) \]

Taking the gradient w.r.t. \( \theta \) gives

\[ \nabla_\theta J(\theta) = \nabla_\theta \sum_{\tau} P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} \nabla_\theta P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_\theta P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} P(\tau; \theta) \frac{\nabla_\theta P(\tau; \theta)}{P[\tau; \theta]} R(\tau) \]

\[ = \sum_{\tau} P(\tau; \theta) \nabla_\theta \log P(\tau; \theta) R(\tau) \]
Likelihood Ratio Policy Gradient

\[ J(\theta) = \sum_{\tau} P[t; \theta] R(\tau) \]

Taking the gradient w.r.t. \( \theta \) gives

\[ \nabla_{\theta} J(\theta) = \nabla_{\theta} \sum_{\tau} P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} \nabla_{\theta} P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} \frac{P(\tau; \theta)}{P(\tau; \theta)} \nabla_{\theta} P[\tau; \theta] R(\tau) \]

\[ = \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P(\tau; \theta)}{P[\tau; \theta]} R(\tau) \]

\[ = \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P(\tau; \theta) R(\tau) \]

Approximate with the empirical estimate for m sample paths under policy

\[ \nabla_{\theta} J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} \log P(\tau^{(i)}; \theta) R(\tau^{(i)}) \]
Decompose Path into States and Actions

\[ \nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[ \prod_{t=0}^{H} P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) \cdot \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right] \]
Decompose Path into States and Actions

\[ \nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[ \prod_{t=0}^{H} P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) \cdot \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right] \\
= \nabla_\theta \left[ \sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right] \]
Decompose Path into States and Actions

\[
\nabla_{\theta} \log P(\tau^{(i)}; \theta) = \nabla_{\theta} \log \left[ \prod_{t=0}^{H} P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) \cdot \pi_{\theta}(u_t^{(i)}|s_t^{(i)}) \right] \\

= \nabla_{\theta} \left[ \sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_{\theta}(u_t^{(i)}|s_t^{(i)}) \right] \\

= \nabla_{\theta} \sum_{t=0}^{H} \log \pi_{\theta}(u_t^{(i)}|s_t^{(i)})
\]
Decompose Path into States and Actions

\[
\nabla_\theta \log P(\tau^{(i)}; \theta) = \nabla_\theta \log \left[ \prod_{t=0}^{H} P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) \cdot \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]
\]

\[
= \nabla_\theta \left[ \sum_{t=0}^{H} \log P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)}) + \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \right]
\]

\[
= \nabla_\theta \sum_{t=0}^{H} \log \pi_\theta(u_t^{(i)}|s_t^{(i)})
\]

\[
= \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})
\]

no dynamics model required!!
Gaussian Policy

- Variance may be fixed $\sigma^2$, or can also be parametrized
- Policy is Gaussian, $a \sim \mathcal{N}(\mu(s; \theta), \sigma^2)$
- The score function is

$$
\nabla_\theta \log \pi_\theta(s, a) = \frac{(a - \mu(s; \theta))}{\sigma^2} \frac{\partial \mu(s; \theta)}{\partial \theta}
$$
Likelihood Ratio Gradient: Intuition

\[ \nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)}) \]

- Gradient tries to:
  - **Increase** probability of paths with positive R
  - **Decrease** probability of paths with negative R
Likelihood Ratio Gradient: Intuition

\[ \nabla_\theta J(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)}; \theta) R(\tau^{(i)}) \]

- Gradient tries to:
  - **Increase** probability of paths with positive R
  - **Decrease** probability of paths with negative R

- The reward function is a black box and dynamics are not used anywhere
- The world is a black box.
Likelihood Ratio Gradient Estimate

\[ \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} \log P(\tau^{(i)}; \theta) R(\tau^{(i)}) \]

Here:

\[ \nabla_{\theta} \log P(\tau^{(i)}; \theta) = \sum_{t=0}^{H} \nabla_{\theta} \log \pi_{\theta}(u_{t}^{(i)} | s_{t}^{(i)}) \]

Unbiased means:

\[ \mathbb{E}[\hat{g}] = \nabla_{\theta} U(\theta) \]
Reduce Variance using a Critic

- **A critic** provides an estimate of the expectation of the future reward as opposed to a single return sample.

- Use a function approximator for Q function or the advantage function.

\[
\nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) Q^\pi(s_t, a_t) \right] \\
= \mathbb{E}_\tau \left[ \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t | s_t, \theta) A^\pi(s_t, a_t) \right]
\]
Q-function is state-action-value function:

\[ Q^{\pi,\gamma}(s, a) = \mathbb{E}_\pi \left[ r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s, a_0 = a \right] \]

State-value function:

\[ V^{\pi,\gamma}(s) = \mathbb{E}_\pi \left[ r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \mid s_0 = s \right] \]
\[ = \mathbb{E}_{a \sim \pi} \left[ Q^{\pi,\gamma}(s, a) \right] \]

Advantage function:

\[ A^{\pi,\gamma}(s, a) = Q^{\pi,\gamma}(s, a) - V^{\pi,\gamma}(s) \]
function QAC
    Initialise $s$, $\theta$
    Sample $a \sim \pi_\theta$
    for each step do
        Sample reward $r = R^a_s$; sample transition $s' \sim P^a_s$.
        Sample action $a' \sim \pi_\theta(s', a')$
        $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$
        $\theta = \theta + \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$
        $w \leftarrow w + \beta \delta \phi(s, a)$
        $a \leftarrow a'$, $s \leftarrow s'$
    end for
end function
How do we compute gradients?

\[ \nabla_\theta \mathbb{E} [R_T] \]

- Numerically: finite differencing
- Score function estimator
  - **Problems**: high variance! In particular, as the policy becomes more and more deterministic, the variance explodes.
Variance in the Gaussian case

\[ x \sim \mathcal{N}(\mu(\theta), \sigma(\theta)) \]

\[ g = \nabla_{\theta} \log p_{\theta}(x) f(x) \]

\[ = \frac{(x - \mu(\theta)) \mu'(\theta)}{\sigma^2} f(x) \]
Variance in the Gaussian case

\[ x \sim \mathcal{N}(\mu(\theta), \sigma(\theta)) \]

\[ g = \nabla_\theta \log p_\theta(x) f(x) \]

\[ = \frac{(x - \mu(\theta))\mu'(\theta)}{\sigma^2} f(x) \]

Sample \( x : x = \mu(\theta) + z\sigma, z \sim \mathcal{N}(0, 1) \)

\[ \hat{g} = \frac{z\sigma\mu'(\theta)}{\sigma^2} f(\mu(\theta) + z\sigma) \]

\[ = \frac{z\mu'(\theta)}{\sigma} f(\mu(\theta) + z\sigma) \]

\[ \nabla(\hat{g}) = \mathbb{E}[\hat{g} - g]^2 = \mathbb{E} \left[ \frac{z\mu'(\theta)}{\sigma} f(\mu(\theta) + z\sigma) - g \right]^2 \]
How do we compute gradients?

\[ \nabla_\theta \mathbb{E} [R_T] \]

- Numerically: finite differencing
- Score function estimator
- Deep deterministic policy gradients: giving up stochastic policies
Deep Deterministic Police Gradients

$R_T$: the return of a trajectory

$$\frac{d}{d\theta} \mathbb{E}[R_T] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right]$$
Deep Deterministic Police Gradients

$R_T$: the return of a trajectory

This expectation refers to the actions after time $t$

$$\frac{d}{d\theta} \mathbb{E} [R_T] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right]$$

Continuous control with deep reinforcement learning, Lilicarp et al. 2016
Deep Deterministic Police Gradients

\[ R_T : \text{the return of a trajectory} \]

\[
\frac{d}{d\theta} \mathbb{E} [R_T] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dR_T}{da_t} \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{da_t} \mathbb{E} [R_T | a_t] \frac{da_t}{d\theta} \right] \\
= \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{d\theta} Q(s_t, a_t) \frac{da_t}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{d\theta} Q(s_t, \pi(s_t, z_t; \theta)) \right]
\]

Continuous control with deep reinforcement learning, Lilicrap et al. 2016
Remember: Q learning

- Definition

\[ Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi} [R_t | s_t, a_t] \]
Remember: Q learning

- **Definition**

\[
Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi} [R_t | s_t, a_t]
\]

- **Bellman equation**

\[
Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} [r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi} [Q^\pi(s_{t+1}, a_{t+1})]]
\]

- **Using a deterministic policy**

\[
Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} [r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))]
\]
Remember: Q learning

- **Definition**
  \[ Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i > t} \sim \pi} [R_t | s_t, a_t] \]

- **Bellman equation**
  \[ Q^\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} [r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi} [Q^\pi(s_{t+1}, a_{t+1})]] \]

- **Using a deterministic policy**
  \[ Q^\mu(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E} [r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))] \]

- **Deep Q learning:**
  \[ L(\theta^Q) = \mathbb{E}_{s_t \sim \rho, a_t \sim \beta, r_t \sim E} \left[ (Q(s_t, a_t | \theta^Q) - y_t)^2 \right] \]
  \[ y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1}) | \theta^Q) \]
  \[ \mu(s) = \text{arg max}_a Q(s, a) \]
Q learning in continuous action space

- This optimization takes too long for continuous action spaces and has to be performed at every iteration:

\[ \mu(s) = \arg \max_a Q(s, a) \]
Q learning in continuous action space

- Instead of parametrizing Q let's also parametrize the policy \( a = \mu(\theta) \)

\[
\nabla_{\theta\mu} J \approx \mathbb{E}_{s_t \sim \rho^\beta} \left[ \nabla_{\theta\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)} \right] \\
= \mathbb{E}_{s_t \sim \rho^\beta} \left[ \nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta\mu} \mu(s|\theta^\mu)|_{s=s_t} \right]
\]
How do we compute gradients?

\[ \nabla_\theta \mathbb{E} [R_T] \]

- Numerically: finite differencing
- Score function estimator
- Deep deterministic policy gradients: giving up stochastic policies
- Pathwise derivatives
Pathwise derivatives for Gaussian samples

• Consider normally distributed variable \( y \)

\[
p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x))
\]
Pathwise derivatives for Gaussian samples

- Consider normally distributed variable $y$
  \[ p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x)) \]
- Reparametrization:
  \[ y = \mu(x) + \sigma(x)\xi, \text{ where } \xi \sim \mathcal{N}(0, 1) \]
Pathwise derivatives for Gaussian samples

- Consider normally distributed variable $y$
  \[ p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x)) \]

- Reparametrization:
  \[ y = \mu(x) + \sigma(x)\xi, \text{ where } \xi \sim \mathcal{N}(0, 1) \]

- Sampling: Sample $\xi$ and then deterministically generate $y = f(x, \xi)$

\[
\mathbb{E}_{p(y|x)} g(y) = \int g(f(x, \xi)) \rho(\xi) d\xi
\]

\[
\nabla_x \mathbb{E}_{p(y|x)} g(y) = \mathbb{E}_{\rho(\xi)} g_y f_x \approx \frac{1}{M} \sum_{i=1}^{M} g_y f_x \bigg|_{\xi=\xi_i}
\]
Pathwise derivatives for Gaussian samples

- Consider normally distributed variable $y$
  \[ p(y|x) = \mathcal{N}(y|\mu(x), \sigma^2(x)) \]

- Reparametrization:
  \[ y = \mu(x) + \sigma(x)\xi, \text{ where } \xi \sim \mathcal{N}(0, 1) \]

- Sampling: Sample $\xi$ and then deterministically generate $y = f(x, \xi)$

\[
\mathbb{E}_{p(y|x)} g(y) = \int g(f(x, \xi)) \rho(\xi) d\xi
\]

\[
\nabla_x \mathbb{E}_{p(y|x)} g(y) = \mathbb{E}_{\rho(\xi)} g_y f_x \approx \frac{1}{M} \sum_{i=1}^{M} g_y f_x \bigg|_{\xi=\xi_i}
\]

- Compare to the score function gradient estimator:
  \[
  \frac{1}{M} \sum_{i=1}^{M} \nabla_x \log(p(y|x)) g(y)
  \]
Sampling: sample $\xi$ and then deterministically generate $y$: $y = f(x, \xi)$

$$\mathbb{E}_{p(y|x)} g(y) = \int g(f(x, \xi)) \rho(\xi) d\xi$$

$$\nabla_x \mathbb{E}_{p(y|x)} g(y) = \mathbb{E}_{\rho(\xi)} g_y f_x \approx \frac{1}{M} \sum_{i=1}^{M} g_y f_x \bigg|_{\xi = \xi_i}$$

Compare to the score function gradient estimator:

$$\frac{1}{M} \sum_{i=1}^{M} \nabla_x \log(p(y|x)) g(y)$$

• The pathwise derivative makes use of the gradient of $g$.
• Of course, that assumes we know the function $g$ (our reward function) and how it is related to our actions.
Pathwise derivative for Gaussian Policies

• Gaussian Policies:

\[
a = \mu(s, \theta) + z^* \sigma(s, \theta)
\]

\[
\frac{da}{d\theta} = \frac{d\mu(s, \theta)}{d\theta} + z \frac{d\sigma(s, \theta)}{d\theta}
\]

\[
\nabla_\theta \mathbb{E}_z (R(a(\theta, z)))
\]

\[
\mathbb{E}_z (R'(a(\theta, z)) \frac{da(\theta, z)}{d\theta})
\]

• R should be **known and differentiable**
• To propagate for more than 1 step, dynamics should be known and differentiable
Q learning in continuous **stochastic** action space

- Now use **stochastic policies** using the reparametrization trick!

\[ z \sim \mathcal{N}(0, 1) \]

\[ a = \mu(s; \theta) + z\sigma(s; \theta) \]

\[ Q(s,a) \]

---


Q learning in continuous stochastic action space

\[
\frac{d}{d\theta} \mathbb{E}[R_T] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dR_T}{da_t} \frac{d\theta}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{da_t} \mathbb{E}[R_T | a_t] \frac{d\theta}{d\theta} \right] \\
= \mathbb{E} \left[ \sum_{t=1}^{T} \frac{dQ(s_t, a_t)}{da_t} \frac{d\theta}{d\theta} \right] = \mathbb{E} \left[ \sum_{t=1}^{T} \frac{d}{d\theta} Q(s_t, \pi(s_t, z_t; \theta)) \right]
\]
Learn $Q_\phi$ to approximate $Q^{\pi, \gamma}$, and use it to compute gradient estimates.

Pseudocode:

```
for iteration=1, 2, \ldots \ do
    Execute policy $\pi_\theta$ to collect $T$ timesteps of data
    Update $\pi_\theta$ using $g \propto \nabla_\theta \sum_{t=1}^T Q(s_t, \pi(s_t, z_t; \theta))$
    Update $Q_\phi$ using $g \propto \nabla_\phi \sum_{t=1}^T (Q_\phi(s_t, a_t) - \hat{Q}_t)^2$, e.g. with TD($\lambda$)
end for
```
End-to-end model based RL

- Reparametrization trick for both policies and dynamics

\[
\hat{f}(s, a, \xi) = \mu(s, a) + \sigma(s, a)\xi
\]
\[
a = \mu(s, \theta) + \sigma(s, \theta)z
\]
End-to-end model based RL

- Reparametrization trick for both policies and dynamics
End-to-end model based RL

SVG(∞)

- Just learn dynamics model $f$
- Given whole trajectory, infer all noise variables
- Given transition $(s_t, a_t, s_{t+1})$, infer $\zeta_t = s_{t+1} - f(s_t, a_t)$
- Freeze all policy and dynamics noise, differentiate through entire deterministic computation graph

R should be **known and differentiable**

End-to-end model based RL

SVG(1)

- Instead of learning Q, we learn
  - State-value function $V \approx V^{\pi, \gamma}$
  - Dynamics model $f$, approximating $s_{t+1} = f(s_t, a_t) + \zeta_t$
  - Given transition $(s_t, a_t, s_{t+1})$ infer $\zeta_t = s_{t+1} - f(s_t, a_t)$
  - $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})] = \mathbb{E}[r_t + \gamma V(f(s_t, a_t) + \zeta_t)]$
  - $a_t = \pi(s_t, \theta, \zeta_t)$

Re-parametrization trick for categorical distributions

- Consider variable $y$ following the $K$ categorical distribution:

$$y_k \sim \frac{\exp\left(\frac{\log p_k}{\tau}\right)}{\sum_{j=0}^{K} \exp\left(\frac{\log p_j}{\tau}\right)}$$
Re-parametrization trick for categorical distributions

- Consider variable $y$ following the $K$ categorical distribution:
  
  $$y_k \sim \frac{\exp((\log p_k)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j)/\tau)}$$

- Reparametrization:
  
  $$y_k \sim G(\log p) = \frac{\exp((\log p_k + \varepsilon)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j + \varepsilon)/\tau)}, \quad \varepsilon = -\log(-\log(u)), \quad u \sim \mathcal{U}[0, 1]$$

- Sampling: sample $u$ and then sample from $G(\log p)$ to generate $y_k$
Re-parametrization trick for categorical distributions

• Consider variable $y$ following the $K$ categorical distribution:

$$y_k \sim \frac{\exp((\log p_k)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j)/\tau)}$$

• Reparametrization:

$$y_k \sim G(\log p) = \frac{\exp((\log p_k + \varepsilon)/\tau)}{\sum_{j=0}^{K} \exp((\log p_j + \varepsilon)/\tau)}, \quad \varepsilon = -\log(-\log(u)), \; u \sim \mathcal{U}[0, 1]$$

• Sampling: sample $u$ and then sample from $G(\log p)$ to generate $y_k$

• In the forward pass you sample from the parametrized distribution

$$c \sim G(\log p)$$

• In the backward pass you use the soft distribution:

$$\frac{dc}{d\theta} = \frac{dG}{dp} \frac{dp}{d\theta}$$
Bacproping through discrete variables

For binary neurons:

- **forward pass**
- **backward pass**
  - Straight-through
  - sigmoidal

Bacproping through discrete variables

For categorically distributed neurons:

forward pass

backward pass

Categorical reparametrization with Gumbel-Softmax Sang et al. 2017
Summary

• Recap of estimating gradients

• Backpropagating through sampling using the reparametrization trick.