Motion learning in robotics

Karel Zimmermann

http://cmp.felk.cvut.cz/~zimmerk/

Vision for Robotics and Autonomous Systems
https://cyber.felk.cvut.cz/vras/

Center for Machine Perception
https://cmp.felk.cvut.cz

Department for Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
Tasks often formalised as MDP

States: $\mathbf{x} \in \mathcal{R}^n$
Both tasks formalised as reinforcement learning problems

States: \( x \in \mathcal{R}^n \)

Actions: \( a \in \mathcal{R}^m \)
Both tasks formalised as reinforcement learning problems

States: $\mathbf{x} \in \mathcal{R}^n$

Actions: $\mathbf{a} \in \mathcal{R}^m$

Model: $p(\mathbf{x}'|\mathbf{x}, \mathbf{a})$
Both tasks formalised as reinforcement learning problems

States: \( \mathbf{x} \in \mathcal{R}^n \)

Actions: \( \mathbf{a} \in \mathcal{R}^m \)

Model: \( p(\mathbf{x}'|\mathbf{x}, \mathbf{a}) \)

Rewards: \( r(\mathbf{x}, \mathbf{a}, \mathbf{x}') \in \mathcal{R} \)
Both tasks formalised as reinforcement learning problems

States: \( x \in \mathcal{R}^n \)

Actions: \( a \in \mathcal{R}^m \)

Model: \( p(x'|x, a) \)

Rewards: \( r(x, a, x') \in \mathcal{R} \)

Policy: \( \pi(a|x) \)
Both tasks formalised as reinforcement learning problems

States: \( x \in \mathcal{R}^n \)

Actions: \( a \in \mathcal{R}^m \)

Model: \( p(x'|x, a) \)

Rewards: \( r(x, a, x') \in \mathcal{R} \)

Policy: \( \pi(a|x) \)

Goal: \( \pi^* = \arg \max_\pi J_\pi \) (e.g. \( J_\pi = \mathbb{E} \left[ \sum_{t=0}^{T} r_t \right] \))
Challenges in real tasks

States: $x \in \mathcal{R}^n$ incomplete, noisy

Actions: $a \in \mathcal{R}^m$ continuous high-dimensional

Model: $p(x'|x, a)$ inaccurate model

Rewards: $r(x, a, x') \in \mathcal{R}$ hard to engineer

Policy: $\pi(a|x)$ execution endanger the robot

Goal: $\pi^* = \arg \max_\pi J_\pi$ (e.g. $J_\pi = \mathbb{E}\left[\sum_{t=0}^T r_t\right]$)
Challenges in real tasks

• Can I learn something without the model \( p(x' | x, a) \) just from interactions?
Taxonomy of policy search methods

• Direct policy search (primal task)
  e.g. gradient ascent for \( \pi^* = \arg \max_{\pi} J_\pi \)

  \[ \pi^* = \arg \max_{\pi} J_\pi \]

  e.g. search for \( Q(x, a) = r(x, a, x') + \gamma \max_{a'} Q(x', a') \)

  \( \pi^* = \arg \max_a Q(x, a) \)

Episodic REPS [Peters, 2010]
PILCO [Deisenroth, ICML 2011]
Actor-critic (e.g. DPG [Silver, JMLR 2014])
Deep Q-learning (e.g. [Mnih, Nature 2015])

• Value-based methods (dual function [Kober, 2013])

  e.g. search for \( Q(x, a) = r(x, a, x') + \gamma \max_{a'} Q(x', a') \)

  \( \pi^* = \arg \max_a Q(x, a) \)
Value-based methods: Q-learning

States

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics
Value-based methods: Q-learning

Actions

R-right

D-down
Value-based methods: Q-learning

Terminal states

Rewards

-1
-1
-1
-1
-1
-1
-1
-1
+10

-10

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics
State-action value function

\[ Q(x, u) : X \times U \rightarrow \mathbb{R} \]

The best sum of rewards I can get, when following action \( u \) in state \( x \) and then controlling optimally.

- Search for the \( Q \), which satisfies Bellman equation

\[
Q(x, u) = r(x, u, x') + \max_{u'} Q(x', u')
\]
The best sum of rewards I can get, when following action $u$ in state $x$ and then controlling optimally

- Search for the $Q$, which satisfies Bellman equation

$$Q(x, u) = r(x, u, x') + \max_{u'} Q(x', u')$$

- Once we find it, we can control optimally as follows:

$$\pi^*(x) = \arg\max_u Q(x, u) = \arg\max_\pi J_\pi$$
### State-action value function

\[ Q(x, u) : X \times U \to \mathbb{R} \]

The best sum of rewards I can get, when following action \( u \) in state \( x \) and then controlling optimally

- Search for the \( Q \), which satisfies Bellman equation
  \[ Q(x, u) = r(x, u, x') + \max_{u'} Q(x', u') \]

- Once we find it, we can control optimally as follows:
  \[ \pi^*(x) = \arg \max_u Q(x, u) = \arg \max_\pi J_\pi \]

- Search without model is based on collecting trajectories
\( \tau_1 : (a, R, -1), (b, R, -1), (c, R, 10) \)
<table>
<thead>
<tr>
<th>Q</th>
<th>R - right</th>
<th>D - down</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>b</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>c</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>d</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>e</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Having a trajectory, each transition gives one equation:

\[
\tau_2 : (a, R, -1), \ (b, D, -1), \ (d, R, -1), \ (e, R, -10)
\]
Having a trajectory, each transition gives one equation

\[
\tau_2 : (a, R, -1), \ (b, D, -1), \ (d, R, -1), \ (e, R, -10)
\]

\[Q(e, R) = r(e)\]
Having a trajectory, each transition gives one equation.

\[
\begin{align*}
\tau_2: & & (a, R, -1), & (b, D, -1), \\
& & (d, R, -1), & (e, R, -10) \\
Q(e, R) &= r(e) \\
Q(b, R) &= r(b) + \max_u Q(d, u)
\end{align*}
\]
\( \tau_2 : \\
(a, R, -1), (b, D, -1), \\
(d, R, -1), (e, R, -10) \\
Q(e, R) = r(e) \\
Q(b, R) = r(b) + \max_u Q(d, u) \\
Q(d, R) = r(d) + \max_u Q(e, u) \\
Q(a, R) = r(a) + \max_u Q(b, u) \\

Having a trajectory, each transition gives one equation
Having a trajectory, each transition gives one equation.

\[
\tau_2 : \quad (a, R, -1), \ (b, D, -1), \ (d, R, -1), \ (e, R, -10)
\]

\[
\begin{align*}
Q(c, R) &= r(c) \\
Q(b, R) &= r(b) + \max_u Q(d, u) \\
Q(d, R) &= r(d) + \max_u Q(e, u) \\
Q(a, R) &= r(a) + \max_u Q(b, u)
\end{align*}
\]
(1) Substitute transitions and current Q-values to the right side and solve for left side.
(1) Substitute transitions and current Q-values to the right side and solve for left side.
$\tau_2 : (a, R, -1), (b, D, -1), (d, R, -1), (e, R, -10)$

$$
\begin{align*}
Q(e, R) &= -10 \\
Q(b, R) &= -1 \\
Q(d, R) &= r(d) + \max_u Q(e, u) \\
Q(a, R) &= r(a) + \max_u Q(b, u)
\end{align*}
$$

(1) Substitute transitions and current Q-values to the right side and solve for left side.
\[ Q(a, R) = r(a) + \max_u Q(b, u) \]

(1) Substitute transitions and current Q-values to the right side and solve for left side.
(1) Substitute transitions and current Q-values to the right side and solve for left side.

\[ \tau_2 : (a, R, -1), (b, D, -1), (d, R, -1), (e, R, -10) \]

<table>
<thead>
<tr>
<th>Q</th>
<th>R - right</th>
<th>D - down</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>e</td>
<td>-10</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ \begin{align*}
\tau_2 : & \\
(a, R, -1), & (b, D, -1), \\
(d, R, -1), & (e, R, -10) \\
\end{align*} \]

\[
\begin{align*}
Q(e, R) &= r(e) \\
Q(b, R) &= r(b) + \max_u Q(d, u) \\
Q(d, R) &= r(d) + \max_u Q(e, u) \\
Q(a, R) &= r(a) + \max_u Q(b, u) \\
\end{align*} \]

(1) Substitute transitions and current Q-values to the right side and solve for left side.
(2) Repeat several times
\( \tau_2 : \)

\[
(a, R, -1), (b, D, -1), (d, R, -1), (e, R, -10)
\]

\[
\begin{align*}
Q(e, R) &= r(e) \\
Q(b, R) &= r(b) + \max_u Q(d, u) \\
Q(d, R) &= r(d) + \max_u Q(e, u) \\
Q(a, R) &= r(a) + \max_u Q(b, u)
\end{align*}
\]

(1) Substitute transitions and current Q-values to the right side and solve for left side.

(2) Repeat several times (search for the fixed point of the Bellman operator)
\[ Q(b, R) = r(b) + \max \limits_{u} Q(d, u) \]
\[ Q(d, R) = r(d) + \max \limits_{u} Q(e, u) \]
\[ Q(a, R) = r(a) + \max \limits_{u} Q(b, u) \]

(1) Substitute transitions and current Q-values to the right side and solve for left side.

(2) Repeat several times (search for the fixed point of the Bellman operator)

\[ Q = \mathcal{B}(Q) \]

\[ \tau_2 : \]
\[ (a, R, -1), (b, D, -1), (d, R, -1), (e, R, -10) \]

Iterations of the Bellman operator always converge to a fixed point !!!
Bellman equation

\[ Q(x, u) = r(x, u, x') + \max_{u'} Q(x', u') \]

reward for transition

the best you can do from

the following state

Which path is better?
Bellman equation

\[ Q(x, u) = r(x, u, x') + \gamma \max_{u'} Q(x', u') \]

the best you can do from the following state

reward for transition

discount factor \( \gamma \in [0; 1] \)
Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$
2. Solve $Q(x, u) = r(x, u, x') + \gamma \max_{u'} Q(x', u')$
3. Repeat from 1
Q-learning

1. Collect trajectories \( \tau_1, \tau_2, \tau_3, \ldots \)
2. Solve \( Q(x, u) = r(x, u, x') + \gamma \max_{u'} Q(x', u') \)
3. Repeat from 1
   - Curse of dimensionality
Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$
2. Solve $Q(x, u) = r(x, u, x') + \gamma \max_{u'} Q(x', u')$
3. Repeat from 1

• Curse of dimensionality
• Replace table $Q(x, u)$ by function $Q_\theta(x, u)$
Q-learning

1. Collect trajectories \( \tau_1, \tau_2, \tau_3, \ldots \)
2. Solve \( Q(x, u) = r(x, u, x') + \gamma \max_{u'} Q(x', u') \)
3. Repeat from 1

- Curse of dimensionality
- Replace table \( Q(x, u) \) by function \( Q_\theta(x, u) \)

Approximate Q-learning

1. Collect trajectories \( \tau_1, \tau_2, \tau_3, \ldots \), initialize \( \theta = \text{rand} \)
2. Estimate \( y = r(x, u, x') + \gamma \max_{u'} Q_\theta(x', u') \)
3. Update parameters by learning
   \[
   \arg \min_{\theta} \sum_{x,u,y} \| Q_\theta(x, u) - y \|
   \]
4. Repeat from 2
5. Repeat from 1
Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$
2. Solve $Q(x, u) = r(x, u, x') + \gamma \max_{u'} Q(x', u')$
3. Repeat from 1

- Curse of dimensionality
- Replace table $Q(x, u)$ by function $Q_\theta(x, u)$

Approximate Q-learning

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, initialize $\theta = \text{rand}$
2. Estimate $\hat{y} = r(x, u, x') + \gamma \max_{u'} Q_\theta(x', u')$
3. Update parameters by learning
   $$\arg \min_{\theta} \sum_{x, u, y} \| Q_\theta(x, u) - \hat{y} \|$$
4. Repeat from 2
5. Repeat from 1

Approximated Q-learning does not have to converge to a fixed-point !!!
Mnih et al. Nature 2015

- 2600 atari games
- **state space:** pixels (e.g. VGA resolution)
- **action space:** discrete joystic actions (8 direction + 8 direction with button + neutral action)
- replay buffer (decorrelates samples to be “more i.i.d”)
- two Q-networks (suppress oscillations)
Mnih et al. Nature 2015

- 2600 atari games
- **state space**: pixels (e.g. VGA resolution)
- **action space**: discrete joystic actions (8 directions + 8 directions with button)
- collection of control tasks: [https://gym.openai.com](https://gym.openai.com)
In additional simulations (see Supplementary Discussion and Extended Data Tables 3 and 4), we demonstrate the importance of the individual core components of the DQN agent—the replay memory, separate target Q-network and deep convolutional network architecture—by disabling them and demonstrating the detrimental effects on performance.

We next examined the representations learned by DQN that underpinned the successful performance of the agent in the context of the game Space Invaders (see Supplementary Video 1 for a demonstration of the performance of DQN), by using a technique developed for the visualization of high-dimensional data called 't-SNE'. As expected, the t-SNE algorithm tends to map the DQN representation of perceptually similar states to nearby points. Interestingly, we also found instances in which the t-SNE algorithm generated similar embeddings for DQN representations of states that are close in terms of expected reward but perceptually dissimilar (Fig. 4, bottom right, top left and middle), consistent with the notion that the network is able to learn representations that support adaptive behaviour from high-dimensional sensory inputs.

Furthermore, we also show that the representations learned by DQN are able to generalize to data generated from policies other than its own—in simulations where we presented as input to the network game states experienced during human and agent play, recorded the representations of the last hidden layer, and visualized the embeddings generated by the t-SNE algorithm (Extended Data Fig. 1 and Supplementary Discussion). Extended Data Fig. 2 provides an additional illustration of how the representations learned by DQN allow it to accurately predict state and action values.

It is worth noting that the games in which DQN excels are extremely varied in their nature, from side-scrolling shooters (River Raid) to boxing games (Boxing) and three-dimensional car-racing games (Enduro).
Hessel et. al Rainbow DQN, 2017

![Graph showing the performance of different algorithms over millions of frames. The Y-axis represents the median human-normalized score, and the X-axis represents millions of frames. The graph compares DQN, DDQN, Prioritized DDQN, Dueling DDQN, A3C, Distributional DQN, Noisy DQN, and Rainbow.]
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)

\[ J_\pi \]

well-chosen dense rewards
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)

$J_\pi$ vs $\pi$

badly chosen dense rewards
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Half cheetah:
  - sparse rewards (for reaching the goal position fast)
  - dense rewards (for velocity)
Reward shaping

- Sparse rewards are easier to design correctly
- Dense rewards are easier to learn
- Boat racing (bad dense rewards):
  - sparse rewards (winning the race)
  - dense rewards (collecting powerups, checkpoints …)
Source: Peter Pastor
Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup
Learning from expert demonstrations

• Sometimes easier to provide good trajectories than good rewards.

• Imitation learning setup
  1. Collect expert trajectories $\tau^*_1, \tau^*_2, \tau^*_3, \ldots$
  2. Find policy $\arg \min_{\theta} \sum_{(x_i, a_i) \in \tau^*} ||\pi_\theta(x_i) - a_i||^2_2$
Learning from expert demonstrations

• Sometimes easier to provide good trajectories than good rewards.

• Imitation learning setup (statistically inconsistent + blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find policy $\arg \min_\theta \sum_{(x_i, a_i) \in \tau^*} \| \pi_\theta(x_i) - a_i \|^2$

• Inverse reinforcement learning setup
Learning from expert demonstrations

- Sometimes easier to provide good trajectories than good rewards.

- Imitation learning setup (statistically inconsistent+ blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find policy

$$\text{arg min}_{\theta} \sum_{(x_i, a_i) \in \tau^*} \| \pi_{\theta}(x_i) - a_i \|^2_2$$

- Inverse reinforcement learning setup
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_w$

$$\text{arg min}_w \| w \|^2_2$$

subject to:
$$\sum_{(x, u, x') \in \tau^*} r_w(x, u, x') \leq \sum_{(x, u, x') \in \{ \mathcal{T} \setminus \tau^* \}} r_w(x, u, x')$$
Learning from expert demonstrations

• Sometimes easier to provide good trajectories than good rewards.

• Imitation learning setup (statistically inconsistent + blackbox)
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find policy $\arg\min_{\theta} \sum_{(x_i,a_i) \in \tau^*} \| \pi_\theta(x_i) - a_i \|^2_2$

• Inverse reinforcement learning setup
  1. Collect expert trajectories $\tau_1^*, \tau_2^*, \tau_3^*, \ldots$
  2. Find reward function $r_w$

\[
\arg\min_w \| w \|^2_2
\]

subject to: $\sum_{(x,u,x') \in \tau^*} r_w(x,u,x') \leq \sum_{(x,u,x') \in \{\mathcal{T} \setminus \tau^*\}} r_w(x,u,x')$

3. Solve underlying RL task
inverse reinforcement learning

- **state space:** angular and euclidean position, velocity, acceleration
- **action space:** motor torques
- learning reward function from expert pilot

Abbeel et al. IJRR 2010
Silver et al. IJRR 2010

input image (state)  learned reward function (traversability map)

Taxonomy of policy search methods

- **Direct policy search (primal task)**
  e.g. gradient ascent for \( \pi^* = \arg \max_\pi J_\pi \)

- **Value-based methods (dual function [Kober, 2013])**
  e.g. search for \( Q(x, a) = r(x, a, x') + \gamma \max_{a'} Q(x', a') \)
  \[ \pi^* = \arg \max_a Q(x, a) \]

- Episodic REPS [Peters, 2010]
- PILCO [Deisenroth, ICML 2011]
- Actor-critic (e.g. DPG [Silver, JMLR 2014])
- Deep Q-learning (e.g. [Mnih, Nature 2015])
Primal task

1. Randomly initialize policy $\pi_\theta$
Primal task

1. Randomly initialize policy $\pi_\theta$
2. Collect trajectories $\tau$ with policy $\pi_\theta$
Primal task

1. Randomly initialize policy $\pi_\theta$
2. Collect trajectories $\tau$ with policy $\pi_\theta$
3. Denote $p(\tau | \pi_\theta)$ probability of $\tau$ occurs when following $\pi_\theta$
Primal task

1. Randomly initialize policy $\pi_\theta$
2. Collect trajectories $\tau$ with policy $\pi_\theta$
3. Denote $p(\tau|\pi_\theta)$ probability of $\tau$ occurs when following $\pi_\theta$
4. Define criterion

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \{ r(\tau) \} = \int_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau) \, d\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$
Primal task

1. Randomly initialize policy $\pi_\theta$
2. Collect trajectories $\tau$ with policy $\pi_\theta$
3. Denote $p(\tau|\pi_\theta)$ probability of $\tau$ occurs when following $\pi_\theta$
4. Define criterion

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau|\pi_\theta)} \{ r(\tau) \} = \int_{\tau \in \mathcal{T}} p(\tau|\pi_\theta) r(\tau) \, d\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i)$$

5. Optimize criterion (e.g. gradient descent)

$$\theta^* = \arg \min_{\theta} J(\theta)$$

6. Repeat from 2
Primal task
\[ J(\theta) = \mathbb{E}_{\tau \sim p(\tau | \pi_{\theta})} \{ r(\tau) \} = \int_{\tau \in \mathcal{T}} p(\tau | \pi_{\theta}) r(\tau) \, d\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i) \]
\[ \theta^* = \arg \min_{\theta} J(\theta) \]

- What do I need for gradient descent optimization?
  \[ \frac{\partial J(\theta)}{\partial \theta} \]

- Perturb parameters by \( \Delta \theta_i \) and estimate \( J(\theta + \Delta \theta_i) \)
  \[ J(\theta + \Delta \theta_i) = J(\theta) + \frac{\partial J(\theta)}{\partial \theta} \Delta \theta_i \]
  \[ \Delta \theta_i^\top \frac{\partial J(\theta)}{\partial \theta} = J(\theta) - J(\theta + \Delta \theta_i) \]
What do I need for gradient descent optimization?

- Perturb parameters by $\Delta \theta_i$ and estimate $J(\theta + \Delta \theta_i)$.

$$ J(\theta + \Delta \theta_i) = J(\theta) + \left( \frac{\partial J(\theta)}{\partial \theta} \right)^\top \Delta \theta_i $$

$$ \Delta \theta_i^\top \frac{\partial J(\theta)}{\partial \theta} = J(\theta) - J(\theta + \Delta \theta_i) $$

\[
\begin{bmatrix}
\Delta \theta_1^\top \\
\vdots \\
\Delta \theta_n^\top 
\end{bmatrix} \frac{\partial J(\theta)}{\partial \theta} = 
\begin{bmatrix}
J(\theta) - J(\theta + \Delta \theta_1) \\
\vdots \\
J(\theta) - J(\theta + \Delta \theta_n)
\end{bmatrix}
\]

\[
\text{matrix } A \quad \text{vector } b
\]

Primal task

$$ J(\theta) = \mathbb{E}_{\tau \sim p(\tau | \pi_\theta)} \left\{ r(\tau) \right\} = \int_{\tau \in \mathcal{T}} p(\tau | \pi_\theta) r(\tau) \, d\tau \approx \frac{1}{N} \sum_{i=1}^{N} r(\tau_i) $$

$$ \theta^* = \arg \min_{\theta} J(\theta) $$
Primal task

\[
\begin{bmatrix}
\Delta \theta_1^T \\
\vdots \\
\Delta \theta_n^T
\end{bmatrix}
\begin{bmatrix}
\frac{\partial J(\theta)}{\partial \theta} = \\
\end{bmatrix}
\begin{bmatrix}
J(\theta) - J(\theta + \Delta \theta_1) \\
\vdots \\
J(\theta) - J(\theta + \Delta \theta_n)
\end{bmatrix}
\]

\[
\begin{bmatrix}
\Delta \theta_1^T \\
\vdots \\
\Delta \theta_n^T
\end{bmatrix}^+ 
\begin{bmatrix}
J(\theta) - J(\theta + \Delta \theta_1) \\
\vdots \\
J(\theta) - J(\theta + \Delta \theta_n)
\end{bmatrix}
\]
Primal task

1. Randomly initialize $\theta$

2. Collect trajectories randomly perturbed policy $\pi_\theta + \Delta \theta_i$

3. Compute gradient $\frac{\partial J(\theta)}{\partial \theta}^\top$ using pseudo-inverse

$$\frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \Delta \theta_1^\top \\ \vdots \\ \Delta \theta_n^\top \end{bmatrix}^+ \cdot \begin{bmatrix} J(\theta) - J(\theta + \Delta \theta_1) \\ \vdots \\ J(\theta) - J(\theta + \Delta \theta_n) \end{bmatrix}$$

4. Update parameters

$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}$$
Primal task

REINFORCE: better gradient approximation

• stochastic policy
  \[ \pi_\theta(u|x) : X \times U \to [0; 1] \]

• gradient of the criterion
  \[ \nabla_\theta J(\theta) = \int_T \nabla_\theta p(\tau|\theta)r(\tau)d\tau \]

• likelihood ratio trick express gradient of the prob distr.
  \[ \nabla_\theta p(\tau|\theta) = p(\tau|\theta) \nabla_\theta \log p(\tau|\theta) \]
Primal task

- after substitution

\[ \nabla_\theta J(\theta) = \int_T p(\tau | \theta) \nabla_\theta \log p(\tau | \theta) r(\tau) \, d\tau = \]

\[ = E[\nabla_\theta \log p(\tau | \theta) r(\tau)] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(\tau_i | \theta) r(\tau_i) \]

- where prob distribution simplified using MDP assumption

\[ p(\tau | \theta) = p(x_0) \prod_k p(x_{k+1} | x_k, u_k) \pi_\theta(u_k | x_k) \]

\[ \nabla_\theta \log p(\tau | \theta) = \nabla_\theta \left[ \log p(x_0) + \sum_k \log p(x_{k+1} | x_k, u_k) + \sum_k \log \pi_\theta(u_k | x_k) \right] \]

\[ = \sum_k \nabla_\theta \log \pi_\theta(u_k | x_k) \]
Primal task

REINFORCE algorithm:
• collect N trajectories

\[ \tau_1 = [(u_{1,1}, x_{1,1}) \ldots u_{M,1}, x_{M,1})] \]

\[ \vdots \]

\[ \tau_N = [(u_{1,N}, x_{1,N}) \ldots u_{M,N}, x_{M,N})] \]

• compute gradient

\[ \nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{M} \nabla_\theta \log \pi_\theta (u_{k,i}|x_{k,i}) \]

• update parameters

\[ \theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta} \]
Primal task

- No motion model required
- Converges to local optima (good initialization needed)
- High-dimensional parameters are requires many samples
- Imitation learning from expert trajectories

- There are better gradient approximations [Deisenroth 2013] (e.g. REINFORCE, GPREPS, …)

Peters et al. NOW 2013

• imitation learning from human demonstration
• **state space:** joint positions, velocities, acceler.
• **action space:** motor torques
• gradient minimization in policy parameter space
Primal task

- No motion model required
- Converges to local optima (good initialization needed)
- High-dimensional parameters are requires many samples
- Imitation learning from expert trajectories

- There are better gradient approximations [Deisenroth 2013] (e.g. REINFORCE, GPREPS, …)

- If motion model is available then trajectory optimization
  [Tassa 2013] Tassa, Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization, IROS2013
Taxonomy of policy search methods

- Direct policy search (primal task)
  \[ \pi^* = \arg \max_{\pi} J_\pi \]
  e.g. gradient ascent for

- Value-based methods (dual function [Kober, 2013])
  \[ Q(x, a) = r(x, a, x') + \gamma \max_{a'} Q(x', a') \]
  \[ \pi^* = \arg \max_a Q(x, a) \]

---

Episodic REPS [Peters, 2010]
PILCO [Deisenroth, ICML 2011]
Actor-critic (e.g. DPG [Silver, JMLR 2014])
Deep Q-learning (e.g. [Mnih, Nature 2015])
Actor-critic methods

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$. Initialize $\theta = \text{rand}$
2. Estimate $y = r(x, u, x') + \gamma \max_{u'} Q_\theta(x', u')$
3. Update parameters by learning

$$\arg \min_{\theta} \sum_{x, u, y} \| Q_\theta(x, u) - y \|$$

Approximated Q-learning
Actor-critic methods

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, initialize $\theta = \text{rand}$
2. Estimate $y = r(x, u, x') + \gamma \max_{u'} Q_\theta(x', u')$
3. Update parameters by learning
   $$\arg \min_{\theta} \sum_{x, u, y} \| Q_\theta(x, u) - y \|$$
4. Learn policy $\pi_\omega$ which do actions maximizing the state-action value function on the collected trajectories
   $$\arg \max_\omega \sum_{x \in \tau} Q_\theta(x, \pi_\omega(x))$$

Direct policy optimization on $Q$
Unrolling in time

1. Collect trajectories $\tau_1, \tau_2, \tau_3, \ldots$, ini: $\theta = \text{rand}, \omega = \text{rand}$

2. Estimate motion model

$$\arg \min_\theta \sum_{(x, x') \in \tau^*} \| p_\theta(x) - x' \|_2^2$$

3. Learn policy maximizing the rewards on model-based trajectories

$$\arg \max_\omega \sum_{x_0} r(p_\theta(\ldots \pi_\omega(p_\theta(x_0, \pi_\omega(x_0)))\ldots))$$

- penalizing distance from training trajectories

Czech Technical University in Prague
Faculty of Electrical Engineering, Department of Cybernetics
3D humanoid

Degrees-of-freedom: 22

Control dimensions: 16 all joints

Cost:
- CoM over mean of feet, (in xy)
- torso over CoM (in xy)
- torso 1.3m over mean of feet (in z)
- minimize horizontal torso velocity
- minimize actuation

This agent, trained on several terrain types, has never seen the "see-saw" terrain.
Levine et al JMLR 2016

- guides policy gradient method by optimal trajectories
- **state space**: RGB camera images
- **action space**: motor torques

(a) hanger  
(b) cube  
(c) hammer  
(d) bottle
Learned Visuomotor Policy: Bottle Task
Boston dynamics - Atlas - NO RL AT ALL
Boston dynamics - Big dog - NO RL AT ALL
Known RL successes

- SearchTrees has no chance in huge state-action spaces
  - AlphaGo:
    - beat professional Go player
    - 9 dan professional ranking
  - Alpha Zero: Top Chess Engine Championship 2017
    - 9h of self-play, no openingbooks nor endgames tables
    - 1 minute per move, 1GB RAM
    - 28 wins, 72 withdraws
- DOTA 2 openAI+ bot [https://blog.openai.com/dota-2/](https://blog.openai.com/dota-2/)
- AutoML [https://cloud.google.com/automl/](https://cloud.google.com/automl/)
  - [Zoph 2016] REINFORCE learns RCNN policy which generates deep CNN architectures.
Summary

• If accurate differentiable motion model and reward functions are known, than optimal control in MDP is straightforward optimization problem (efficiently tackled by DP or DDP)
• State-action value function is dual variable wrt policy. It serves as auxiliary function in the policy optimization:
  • actor-critic methods
  • heuristic in planning methods (LQR trees)
• Holy grail is to efficiently combine motion model, state-action value function and the policy optimization with efficient exploration
• RL will be much more useful for motion control, when accurate domain transfer methods (from simulators to reality) become available.