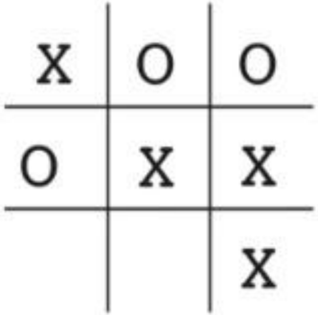
Deep Reinforcement Learning

Examples

Examples of Reinforcement Learning

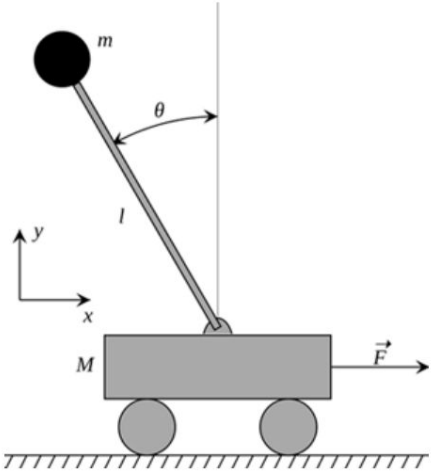


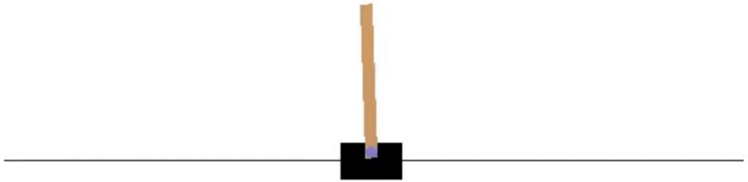
tic-tac-toe board

Tic Tac Toe Game

1. **Goal** - To win the game
2. **State** - Current configuration of the board
3. **Actions** - selecting the cell to place your marker
4. **Reward** - Positve for winning, 0 for tie, negative for loosing

Examples of Reinforcement Learning





Cart-Pole Balancing

- **Goal** — Balance the pole on top of a moving cart
- **State** — Pole angle, angular speed. Cart position, horizontal velocity.
- **Actions** — horizontal force to the cart
- **Reward** — 1 at each time step if the pole is upright

Examples of Reinforcement Learning

Doom*

- **Goal**:
  Eliminate all opponents
- **State:**
  Raw game pixels of the game
- **Actions:**
  Up, Down, Left, Right, Shoot, etc.
- **Reward:**
- Positive when eliminating an opponent, negative when the agent is eliminated

** Added for important thought-provoking considerations of AI safety in the context of autonomous weapons systems (see AGI lectures on the topic).*

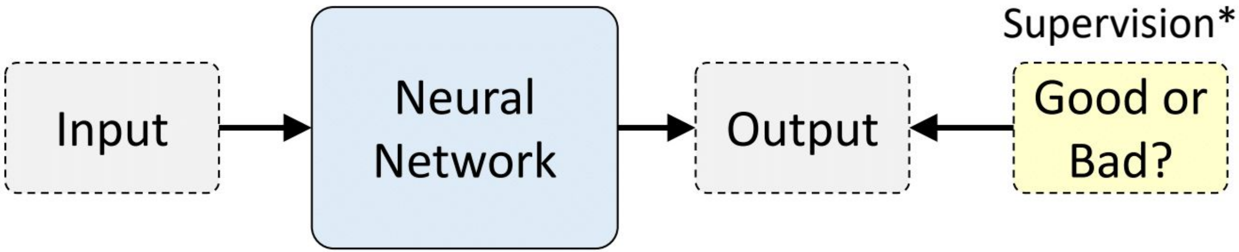
Comparison with Deep Learning

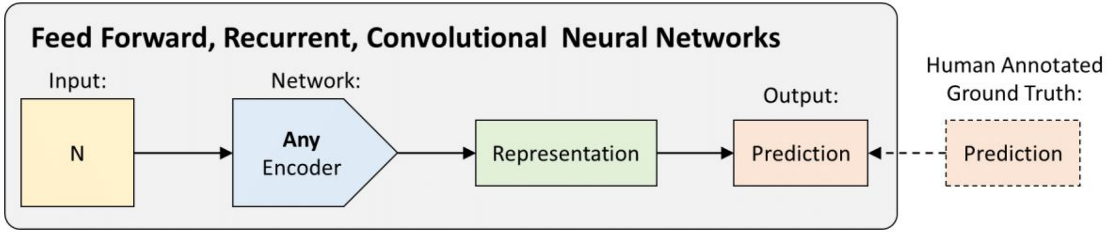
Types of Learning

- Supervised Learning
- Semi-Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

It's all "supervised" by a loss function!

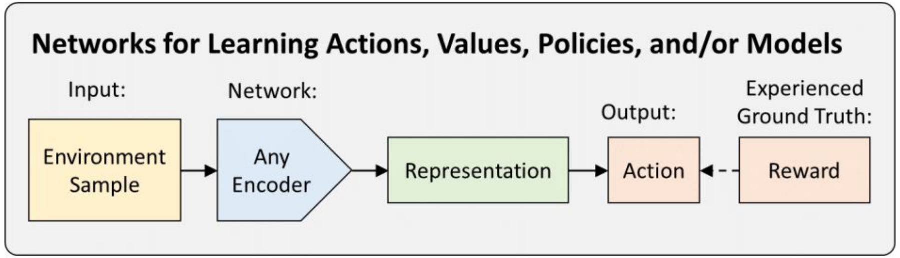
**Someone has to say what's good and what's bad (see Socrates, Epictetus, Kant, Nietzsche, etc.)*

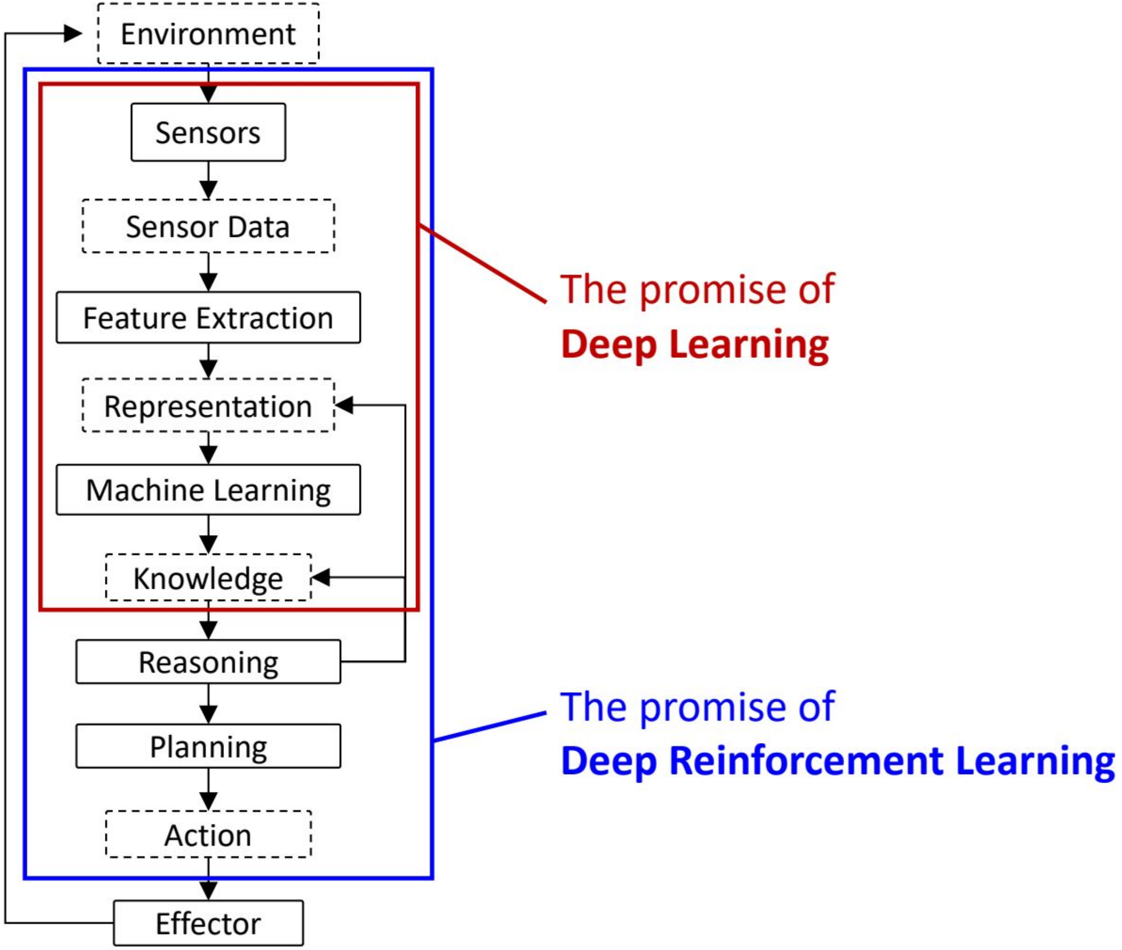




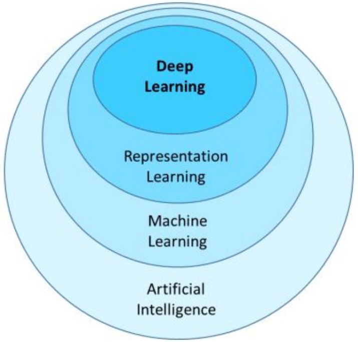
Supervised learning is "teach by **example**":
Here's some examples, now learn patterns in these example.

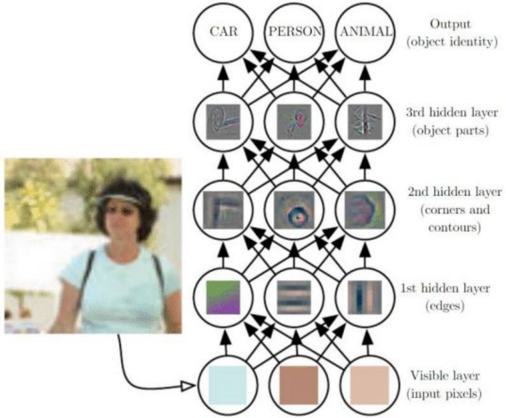
Reinforcement learning is "teach by **experience**":
Here's a world, now learn patterns by exploring it.

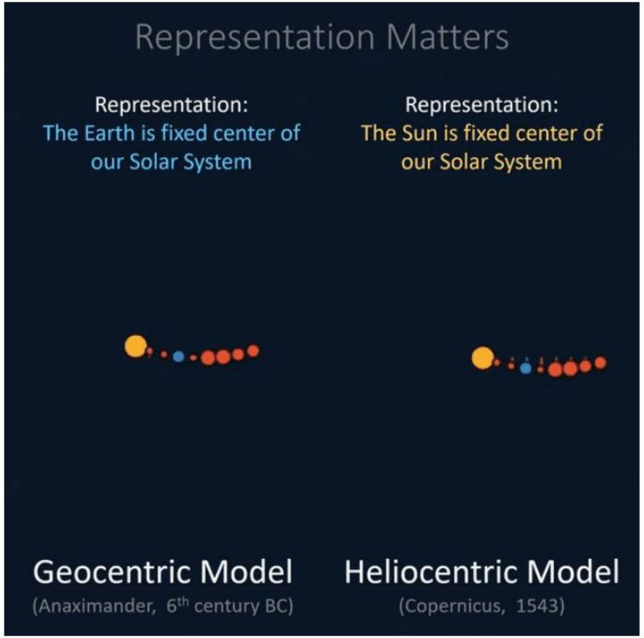



Environment
Sensors
Sensor Data
Feature Extraction
Representation
Machine Learning
Knowledge
Reasoning
Planning
Action
Effector
The promise of
Deep Learning
The promise of
Deep Reinforcement Learning

Deep RL = RL + Neural Networks







Background

Value Functions

- The **value of a state** is the expected return starting from that state; depends on the agent's policy:

State - value function for policy π :

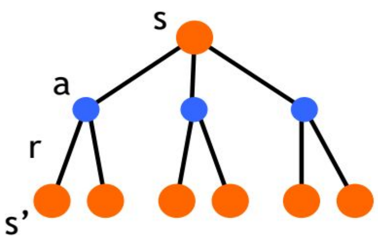
$$V^{\pi}(s) = E_{\pi}\left\{R_t \mid s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\}$$

- The **value of taking an action in a state under policy π** is the expected return starting from that state, taking that action, and thereafter following π :

Action - value function for policy π :

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \mid s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

Q-Learning



- State-action value function: $Q^{\pi}(s,a)$
  - Expected return when starting in *s*, performing *a*, and following π

- Q-Learning: Use **any policy** to estimate Q that maximizes future reward:
  - Q directly approximates Q* (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each (s,a) pair

Learning Rate

Discount Factor

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

New State

Old State

Reward

Q-Learning: Value Iteration

Learning Rate   Discount Factor

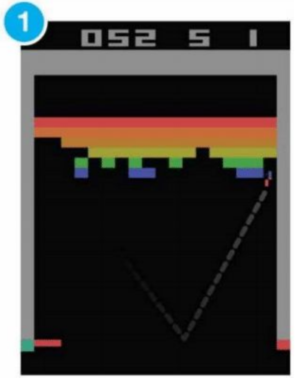
$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

New State   Old State   Reward

	A1	A2	A3	A4
S1	+1	+2	-1	0
S2	+2	0	+1	-2
S3	-1	+1	0	-2
S4	-2	0	+1	+1

```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ maxa' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Q-Learning: Representation Matters



- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

- Think about the **Breakout** game
  - State: screen pixels
    - Image size: $\mathbf{84 \times 84}$ (resized)
    - Consecutive **4** images
    - Grayscale with **256** gray levels

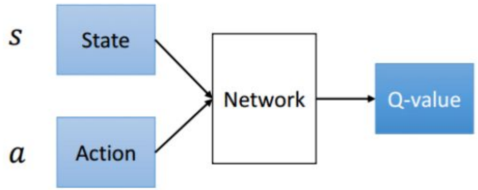
$\mathbf{256^{84\times84\times4}}$ rows in the Q-table!

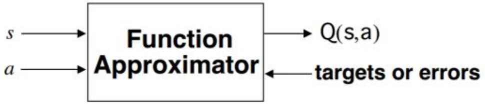
$= 10^{69,970} >> 10^{82}$ atoms in the universe

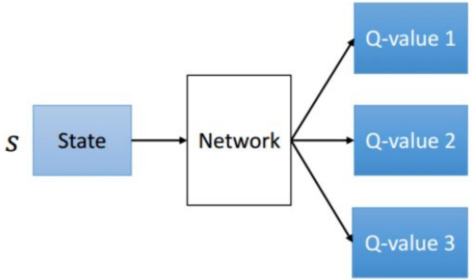
DQN: Deep Q-Learning

Use a neural network to approximate the Q-function:

$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$







Loss Function in Deep Q-Learning

$$L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q\left(s,a;\theta_i\right)\right)^2\right],$$

$$y_i \;=\; \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a\right]$$

$$\nabla_{\theta_i} L_i\left(\theta_i\right) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right].$$

DQN and Double DQN

- Loss function (squared error):

$$L = \mathbb{E}[(\underbrace{\boldsymbol{r + \gamma max_{a'} Q(s', a')}}_{\textbf{target}} - \underbrace{Q(s,a)}_{\textbf{prediction}})^2]$$

- DQN: same network for both Q
- Double DQN: separate network for each Q
  - Helps reduce bias introduced by the inaccuracies of Q network at the beginning of training

DQN Tricks

- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and image x_{t+1}
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for

Policy Gradient Methods

3 Types of Reinforcement Learning

Model-based

- Learn the model of the world, then plan using the model
- Update model often
- Re-plan often

Value-based

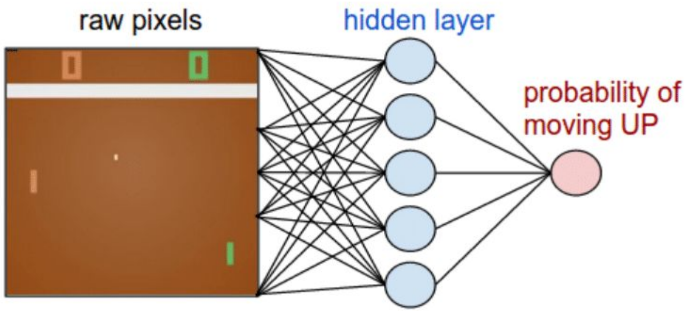
- Learn the state or state-action value
- Act by choosing best action in state
- Exploration is a necessary add-on

Policy-based

- Learn the stochastic policy function that maps state to action
- Act by sampling policy
- Exploration is baked in

Policy Gradient (PG)

- **DQN (off-policy):** Approximate Q and infer optimal policy
- **PG (on-policy):** Directly optimize policy space



Policy Network

Good illustrative explanation:
http://karpathy.github.io/2016/05/31/rl/

"Deep Reinforcement Learning: Pong from Pixels"

Policy Objective Functions

- Goal: given **policy $\pi_\theta(s, a)$**, find best **parameters θ**
- How do we measure the quality of a policy π_θ?
- In episodic environments we can use the **average total return per episode**
- In continuing environments we can use the **average reward per step**

Policy Objective Functions: Episodic

- **Episodic-return objective**:

$$\begin{aligned} J_{\mathrm{G}}(\boldsymbol{\theta}) &= \mathbb{E}_{S_0 \sim d_0, \pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{\infty} \gamma^t R_{t+1}\right] \\ &= \mathbb{E}_{S_0 \sim d_0, \pi_{\boldsymbol{\theta}}}[G_0] \\ &= \mathbb{E}_{S_0 \sim d_0}[\mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_t \mid S_t = S_0]] \\ &= \mathbb{E}_{S_0 \sim d_0}[v_{\pi_{\boldsymbol{\theta}}}(S_0)] \end{aligned}$$

where d_0 is the start-state distribution This objective equals the expected value of the start state

Policy Objective Functions: Average Reward

- **Average-reward objective**

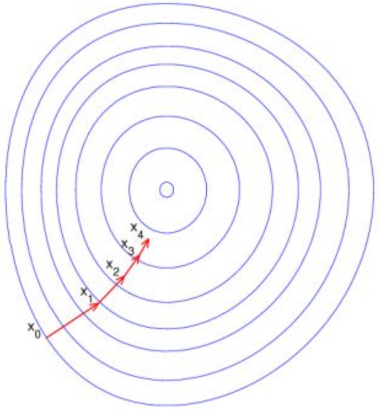
$$
\begin{aligned}
J_{\mathrm{R}}(\boldsymbol{\theta}) &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[R_{t+1}\right] \\
&= \mathbb{E}_{S_t \sim d_{\pi_{\boldsymbol{\theta}}}}\left[\mathbb{E}_{A_t \sim \pi_{\boldsymbol{\theta}}(S_t)}\left[R_{t+1} \mid S_t\right]\right] \\
&= \sum_s d_{\pi_{\boldsymbol{\theta}}}(s) \sum_a \pi_{\boldsymbol{\theta}}(s, a) \sum_r p(r \mid s, a) r
\end{aligned}
$$

where $d_\pi(s) = p(S_t = s \mid \pi)$ is the probability of being in state s in the long run
Think of it as the ratio of time spent in s under policy π

Policy Optimisation

- Policy based reinforcement learning is an **optimization** problem
- Find $\boldsymbol{\theta}$ that maximises $J(\boldsymbol{\theta})$
- We will focus on **stochastic gradient ascent**, which is often quite efficient (and easy to use with deep nets)

Policy Gradient

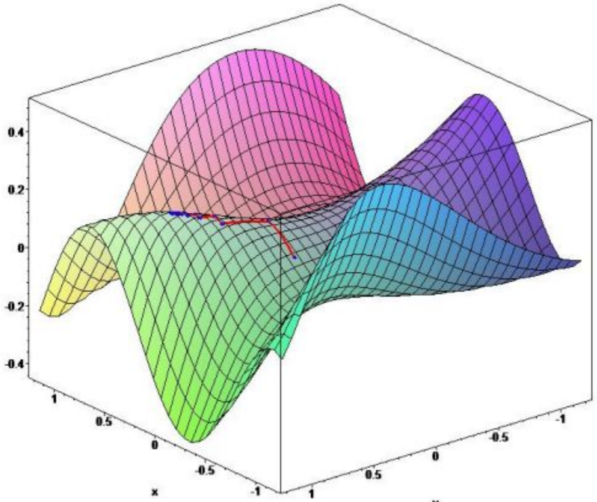


- Idea: ascent the gradient of the objective $J(\boldsymbol{\theta})$

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

- Where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the **policy gradient**

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_1} \\ \vdots \\ \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_n} \end{pmatrix}$$



- and α is a step-size parameter
- Stochastic policies help ensure $J(\boldsymbol{\theta})$ is smooth (typically/mostly)

Gradients on parameterized policies

- How to compute this gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$?
- Assume policy $\pi_{\boldsymbol{\theta}}$ is differentiable almost everywhere (e.g., neural net)
- For average reward

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R] .$$

- How does $\mathbb{E}[R]$ depend on $\boldsymbol{\theta}$?

Contextual Bandits Policy Gradient

- Consider a one-step case (a contextual bandit) such that $\boldsymbol{J(\theta) = \mathbb{E}_{\pi_\theta}[R(S, A)]}$.
  (Expectation is over d (states) and π (actions))
  (For now, d does **not** depend on π)
- We cannot sample R_{t+1} and then take a gradient:
  R_{t+1} is just a number and does not depend on $\boldsymbol{\theta}$!
- Instead, we use the identity:

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(S, A)] = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(S, A) \nabla_{\boldsymbol{\theta}} \log \pi(A|S)] .$$

  (Proof on next slide)
- The right-hand side gives an expected gradient that can be sampled
- Also known as REINFORCE (Williams, 1992)

The score function trick

Let $r_{sa} = \mathbb{E}\left[R(S, A) \mid S = s, A = s\right]$

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[R(S, A)] &= \nabla_{\boldsymbol{\theta}} \sum_{s} d(s) \sum_{a} \pi_{\boldsymbol{\theta}}(a|s)\, r_{sa} \\
&= \sum_{s} d(s) \sum_{a} r_{sa}\, \nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s) \\
&= \sum_{s} d(s) \sum_{a} r_{sa}\, \pi_{\boldsymbol{\theta}}(a|s) \frac{\nabla_{\boldsymbol{\theta}} \pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}}(a|s)} \\
&= \sum_{s} d(s) \sum_{a} \pi_{\boldsymbol{\theta}}(a|s)\, r_{sa}\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) \\
&= \mathbb{E}_{d, \pi_{\boldsymbol{\theta}}}[R(S, A)\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A|S)]
\end{aligned}
$$

Contextual Bandit Policy Gradient

$$\nabla_{\boldsymbol{\theta}} \mathbb{E}[R(S, A)] = \mathbb{E}[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A|S) R(S, A)] \qquad \text{(see previous slide)}$$

- This is something we **can** sample
- Our stochastic policy-gradient update is then

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha R_{t+1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}_t}(A_t|S_t) \,.$$

- In expectation, this is the following the actual gradient
- So this is a pure (unbiased) stochastic gradient algorithm
- Intuition: increase probability for actions with high rewards

Policy Gradient Theorem

- The policy gradient approach also applies to (multi-step) MDPs
- Replaces reward R with long-term return G_t or value $q_\pi(s, a)$
- There are actually two policy gradient theorems (Sutton et al., 2000):

average return per episode & **average reward per step**

Policy gradient theorem (episodic)

Theorem

For any differentiable policy $\pi_{\boldsymbol{\theta}}(s, a)$, let d_0 be the starting distribution over states in which we begin an episode. Then, the policy gradient of $J(\boldsymbol{\theta}) = \mathbb{E}\left[G_0 \mid S_0 \sim d_0\right]$ is

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T} \gamma^t q_{\pi_{\boldsymbol{\theta}}}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t | S_t) \mid S_0 \sim d_0\right]$$

where

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \end{aligned}$$

Episodic policy gradients: proof

- Consider trajectory $\tau = S_0, A_0, R_1, S_1, A_1, R_1, S_2, \ldots$ with return $G(\tau)$

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) = \nabla_{\boldsymbol{\theta}} \mathbb{E}\left[G(\tau)\right] = \mathbb{E}\left[G(\tau) \nabla_{\boldsymbol{\theta}} \log p(\tau)\right] \qquad \text{(score function trick)}$$

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \log p(\tau) &= \nabla_{\boldsymbol{\theta}} \log \left[p(S_0)\pi(A_0|S_0)p(S_1|S_0, A_0)\pi(A_1|S_1)\cdots \right] \\
&= \nabla_{\boldsymbol{\theta}} \left[\log p(S_0) + \log \pi(A_0|S_0) + \log p(S_1|S_0, A_0) + \log \pi(A_1|S_1) + \cdots \right] \\
&= \nabla_{\boldsymbol{\theta}} \left[\log \pi(A_0|S_0) + \log \pi(A_1|S_1) + \cdots \right]
\end{aligned}$$

So:

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) = \mathbb{E}_{\pi}[G(\tau) \nabla_{\boldsymbol{\theta}} \sum_{t=0}^{T} \log \pi(A_t|S_t)]$$

Episodic policy gradients: proof (continued)

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) &= \mathbb{E}_{\pi}[G(\tau) \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)] \\
&= \mathbb{E}_{\pi}[\sum_{t=0}^{T} G(\tau) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)] \\
&= \mathbb{E}_{\pi}[\sum_{t=0}^{T} \left(\sum_{k=0}^{T} \gamma^k R_{k+1} \right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)] \\
&= \mathbb{E}_{\pi}[\sum_{t=0}^{T} \left(\sum_{k=\boldsymbol{t}}^{T} \gamma^k R_{k+1} \right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)] \\
&= \mathbb{E}_{\pi}[\sum_{t=0}^{T} \left(\gamma^t \sum_{k=t}^{T} \gamma^{k-t} R_{k+1} \right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)] \\
&= \mathbb{E}_{\pi}[\sum_{t=0}^{T} \left(\gamma^t G_t \right) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)] \qquad = \mathbb{E}_{\pi}[\sum_{t=0}^{T} \gamma^t q_{\pi}(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)]
\end{aligned}
$$

Episodic policy gradients algorithm

$$\nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t q_\pi(S_t, A_t) \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)]$$

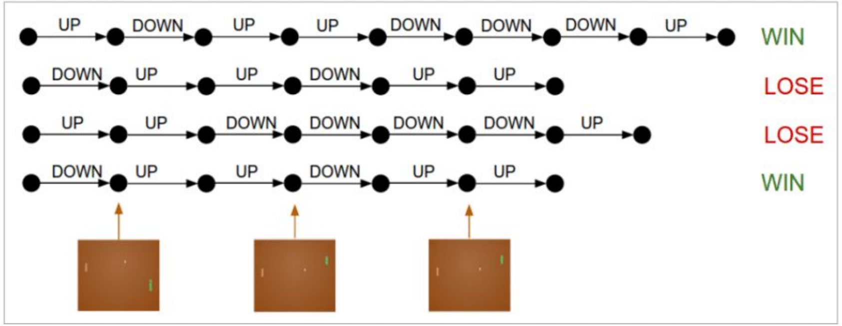
- We can sample this, given a whole episode
- Typically, people pull out the sum, and split up this into separate gradients, e.g.,

$$\Delta\boldsymbol{\theta}_t = \gamma^t G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)$$

  such that $\mathbb{E}_\pi[\sum_t \Delta\boldsymbol{\theta}_t] = \nabla_{\boldsymbol{\theta}} J_{\boldsymbol{\theta}}(\pi)$
- Typically, people ignore the γ^t term, use $\Delta\boldsymbol{\theta}_t = G_t \nabla_{\boldsymbol{\theta}} \log \pi(A_t|S_t)$
- This is actually okay-ish — we just partially pretend on each step that we could have started an episode in that state instead
  (alternatively, view it as a slightly biased gradient)

Policy Gradient – Training

Policy Gradients: Run a policy for a while. See what actions led to high rewards. Increase their probability.



- **REINFORCE:** Policy gradient that increases probability of good actions and decreases probability of bad action:

$$\nabla_\theta E[R_t] = E[\nabla_\theta log P(a) R_t]$$