Lecture Notes for

Neural Networks and Machine Learning

Deep Q-Learning

Last Time

How to Make this more Mathy?

- If we have all possible policies p(x) and a reward function H(x), then maximize

$$\mathbf{E}_{x \leftarrow p(x)}[H(x)] = \mathbf{E}_{x \leftarrow q(x)}[\frac{p(x)}{q(x)} H(x)]$$

- We can approximate the distribution by: $\frac{1}{N} \sum_i \frac{p(x_i)}{q(x_i)} H(x_i)$
- Proven that this is optimized when $\mathbf{KL}(q(x) \,||\, p(x)H(x))$ is minimized. But its intractable, so we drop terms … and end up just optimizing (neg) cross entropy of samples

$$\pi_{k+1} = \arg\max_{\pi_k} \mathbf{E}_{z \leftarrow \pi_k}[\mathbf{1}_{R(z)>\psi} \log \pi_k]$$

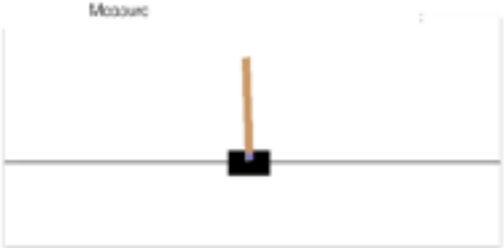
Performance Measure

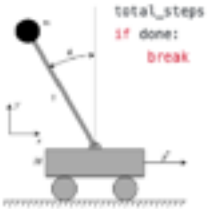
```python
import gym


if __name__ == "__main__":
    env = gym.make("CartPole-v0")

    total_reward = 0.0
    total_steps = 0
    obs = env.reset()

    while True:
        action = env.action_space.sample()
        obs, reward, done, _ = env.step(action)
        total_reward += reward
        total_steps += 1
        if done:
            break
```





Action Space: One input, [0, 1] pull left or pull right

Obs Space: Dynamic state variables (continuous and four dimensional)

End: When more than 15 degrees off or too far from center

Reward: +1 for each time step

Value Iteration (Value Based)

- **Direct**:
  - Initialize V(s) to all zeros
  - Take a series of random steps
  - Perform for each state: $V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} p_{a,s \to s'} \cdot (r_{s,a} + \gamma V(s'))$
  - Repeat until V(s) stops changing
- **Q-Function Variant**:
  - Initialize Q(s,a) to all zeros
  - Take a series of random steps
  - For each state and action: $Q(s,a) \leftarrow \sum_{s' \in S} p_{a,s \to s'} \cdot (r_{s,a} + \gamma \max_{a'} Q(s',a'))$
  - Repeat until Q is not changing

Need to estimate $p_{a,s \to s'}$
Via observed **Transitions**

This Update Will **Converge to Optimal Policy**

Defining the Q-Function

$$V_0 = \max_{a \in A} \mathbf{E}[r_{s,a} + \gamma V_s] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s} \cdot (r_{s,a} + \gamma V_s)$$

- Define intermediate function Q

$$Q(s,a) = \sum_{s' \in S} p_{a,s \to s'} \cdot (r_{s,a} + \gamma V_{s'})$$

- With some nice properties/relations:

$$V_s = \max_{a \in A} Q(s,a)$$

$$Q(s,a) = r_{s,a} + \gamma \max_{a' \in A} Q(s',a')$$

Value Iteration

State Value Review

- Given: $$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_k \gamma^k R_{t+k+1}$$
- $V(s) = \mathbf{E}[G \mid s_t{=}s]$, expected Value of a given state over all future iterations
- **Important**: we can only calculate this exactly if we know:
    - all the rewards for all the states
    - the probabilities of transitioning to a given state from selecting an action
    - likelihood of successful action
    - Most of the time **we know none of this** when we approach the problem, because it assumes a model of the system

The Bellman Equation

- For the case when each action is successful and state is discrete, current V is easy to calculate:

$$V_0 = \max_{a \in 1 \ldots A} (r_a + \gamma V_a)$$

current value is immediate reward plus value of next state with highest value because we will choose this next state and will be successful in reaching it

- General extension for when actions are probabilistic, we need to sum over possible transitions:

$$V_0 = \max_{a \in A} \mathbf{E}[r_{s,a} + \gamma V_s] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s} \cdot (r_{s,a} + \gamma V_s)$$

-probabilities of getting to next state x (current value is immediate reward plus value of next state)
-$p_{a,0 \to s}$ probability of getting to state s from state 0, given that you perform action a

- **Needs:** To select action with best value we need reward matrix, $r_{s,a}$ and action transition matrix $p_{a,0 \to s}$

Defining the Q-Function

$$V_0 = \max_{a \in A} \mathbf{E}[r_{s,a} + \gamma V_s] = \max_{a \in A} \sum_{s \in S} p_{a,0 \to s} \cdot (r_{s,a} + \gamma V_s)$$

- Define intermediate function Q

$$Q(s,a) = \sum_{s' \in S} p_{a,s \to s'} \cdot (r_{s,a} + \gamma V_{s'})$$

- With some nice properties/relations:

$$V_s = \max_{a \in A} Q(s,a)$$

$$Q(s,a) = r_{s,a} + \gamma \max_{a' \in A} Q(s',a')$$

Value Iteration (Value Based)

- **Direct**:
  - Initialize V(s) to all zeros
  - Take a series of random steps
  - Perform for each state: $$V(s) \leftarrow \max_{a \in A} \sum_{s' \in S} p_{a,s \to s'} \cdot (r_{s,a} + \gamma V(s'))$$
  - Repeat until V(s) stops changing
- **Q-Function Variant**:
  - Initialize Q(s,a) to all zeros
  - Take a series of random steps
  - For each state and action: $$Q(s,a) \leftarrow \sum_{s' \in S} p_{a,s \to s'} \cdot (r_{s,a} + \gamma \max_{a'} Q(s',a'))$$
  - Repeat until Q is not changing

Need to estimate $p_{a,s \to s'}$
Via observed **Transitions**

This Update Will **Converge to Optimal Policy**

Value Iteration Reinforcement Learning

M. Lapan Implementation for
and Frozen Lake

```
Follow Along:
08a_Basics_Of_Reinforcement_Learning.ipynb
```

Some Limitations

- Q function can get really big for **large states and action spaces**
- Infinite when the spaces are **continuous**
  - We will solve this by using a neural network to **approximate** the Q function
- Transition matrix, similarly, can get gigantic for large state and action spaces
  - We will solve this by dropping the transition probabilities in Q function update
- This Variant is known as Q-Learning

Atari Paper Presentation

Playing Atari with Deep Reinforcement Learning

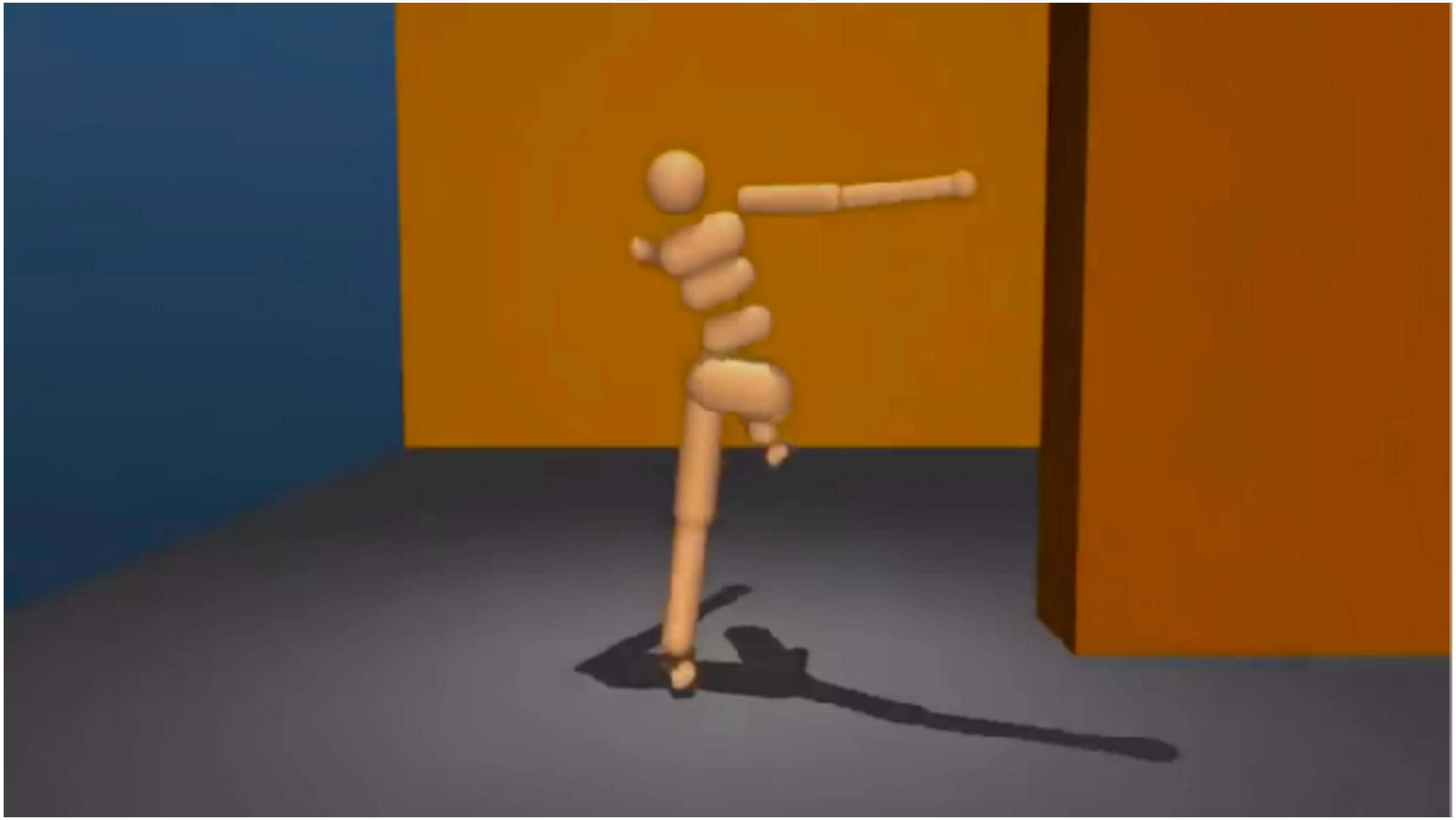
Volodymyr Mnih **Koray Kavukcuoglu** **David Silver** **Alex Graves** **Ioannis Antonoglou**

Daan Wierstra **Martin Riedmiller**

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Q-Learning



Tabular Q-Learning Algorithm

- In update, ignore the transition probability, making use of the iterative nature of Q:

$$Q(s,a) = r_{s,a} + \gamma \max_{a' \in A} Q(s',a')$$

- Add momentum to the update equation

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot [r_{s,a} + \gamma \max_{a' \in A} Q(s',a')]$$

- Algorithm:
  - Sample (with rand) from environment, (s, a, r, s')
  - Make **Bellman Update** with Momentum
  - Repeat until convergence

Tabular Q-Learning Reinforcement Learning

M. Lapan Implementation for and Frozen Lake

Follow Along:
`08a_Basics_Of_Reinforcement_Learning.ipynb`

Q-Learning with a Neural Network

- Want to approximate $Q(s,a)$ when the state space is potentially large. Given s_t, we want the network to give us a row of actions that we can choose from:
  $[\, Q(s_t,a_1),\ Q(s_t,a_2),\ Q(s_t,a_3),\ \ldots\ Q(s_t,a_A) \,]$
- This allows us to make a loss function which incentives the actual Q-function behavior we desire from a sampled tuple (s, a, r, s')

$$\mathscr{L} = \left[Q(s,a) - [r_{s,a} + \gamma \max_{a' \in A} Q^*(s',a')] \right]^2$$

from current network params

from older network params (better stability)

Periodically Update
Params of Q^* from Q

$$\mathscr{L} = \left[Q(s,a) - [r_{s,a}] \right]^2$$

if no next state (env is done)

But we need more power!

- We need to do some **random actions** before following the policy or else we won't learn
- Also, we need to follow the policy more and more during training to get to better places in the environment
- **Epsilon-Greedy** Approach:
    - Start randomly doing actions with prob epsilon
    - Slowly make epsilon smaller as training progresses
- And also we need to have larger amounts of uncorrelated training batches so we will again use **experience replay**

Deep Q-Learning Reinforcement Learning

M. Lapan Implementation for Frozen Lake and Atari!

$$Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot [r_{s,a} + \gamma \max_{a' \in A} Q(s',a')]$$

$$\mathscr{L} = \Big[\underbrace{Q(s,a)}_{\text{from current network params}} - [r_{s,a} + \gamma \underbrace{\max_{a' \in A} Q^*(s',a')]}_{\text{from older network params (better stability)}} \Big]^2$$

from current network params

from older network params (better stability)

$$\mathscr{L} = \big[Q(s,a) - [r_{s,a}] \big]^2$$

if no next state (env is done)

Follow Along:
`08a_Basics_Of_Reinforcement_Learning.ipynb`