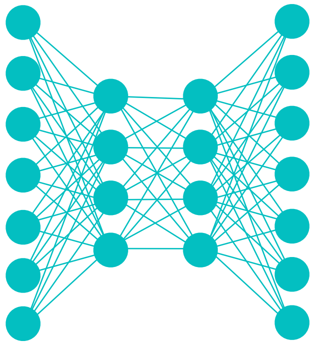
Lecture Notes for

Neural Networks and Machine Learning

Value Iteration and Q-learning

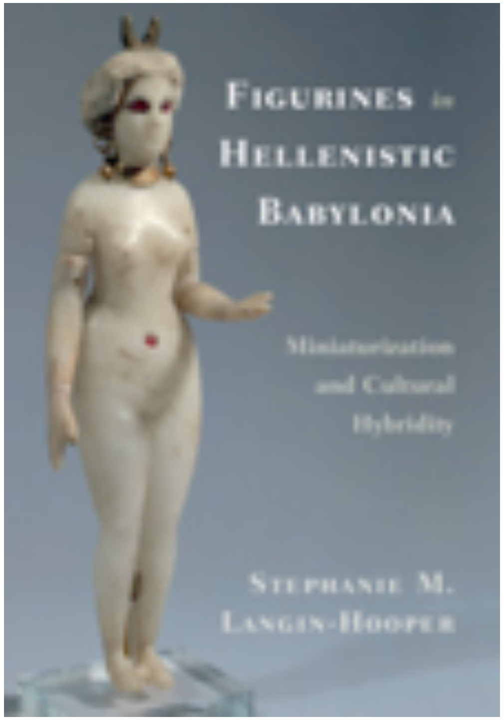


Logistics and Agenda

- Logistics
  - Grading Done for all expect final paper. Contact me if you have anything outstanding.
- Agenda
  - Finish Demo: Cross Entropy Method
  - Student Presentation: AlphaFold
  - Value Iteration (and demo)
  - Tabular Q-Learning
  - Deep Q-Learning (next time?)

Final Project

One Idea from Professor Stephanie Langin-Hooper
SMU Meadows



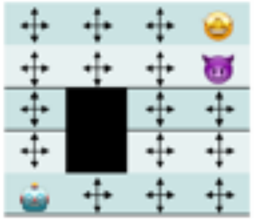
Last Time

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

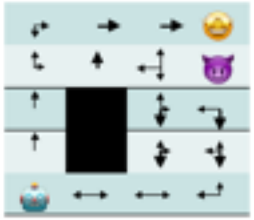
$$\pi_{k+1} = \underset{\pi_k}{\arg\max}\, \mathbf{E}_{z \leftarrow \pi_k}[\mathbf{1}_{\underbrace{R(z)>\psi}_{\text{Performance Measure}}} \log \pi_k]$$

- **State**: Every square in grid
- **Action**: Move to make (l,r,u,d), with probability
- **Reward**: Goal, Death
- **Policy**: Given state, where should we move?
- **Optimal Policy**:
$$\pi^* = \underset{\pi}{\arg\max}\, \mathbf{E}\left[\sum_k \gamma^k R_{t+k+1} | \pi\right]$$



Random Policy

Another Policy



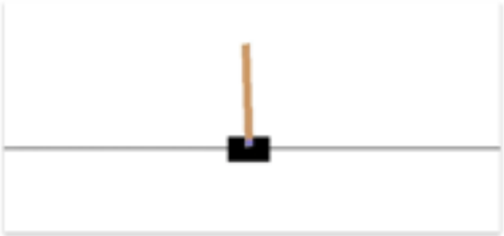
Another Policy

```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
```

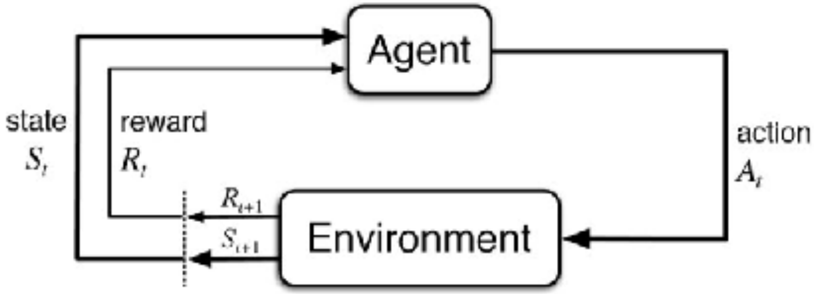


Action Space: One input, [0, 1] pull left or pull right

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



Cross Entropy Reinforcement Learning

M. Lapan Implementation for CartPole and Frozen Lake

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

Paper Presentation

Jürgen Schmidhuber
@SchmidhuberAI

Big news #AlphaFold uses #DeepLearning for #ProteinFolding prediction. This approach was pioneered by Sepp Hochreiter et al. in 2007 when compute was 1000 times more expensive than today. Their LSTM was orders of magnitude faster than the competitors.

PubMed.gov
Fast model-based protein homology detection without alignment - Pub...
pubmed.ncbi.nlm.nih.gov

Value Iteration

When you first start Training with Reinforcement Learning

State Value Function (Review)

- Given:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots = \sum_k \gamma^k R_{t+k+1}$$

- $V(s_t) = \mathbf{E}[G_t \mid s_t{=}s]$, 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, actions, next states
  - the probabilities of transitioning to a given state from selecting an action
  - likelihood of successful action

$$V(s) = \mathbf{E}[G_t \mid s_t = s]$$

$$V(s) = \mathbf{E}[R_{t+1} + \gamma G_{t+1} \mid s_{t+1} = s']$$

$$V(s) = \mathbf{E}[R_{t+1} + \gamma V(s')]$$

The Bellman Equation

- For the case when each action is successful and state is discrete, ideal V has property, $a \rightarrow s$:

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

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

- In general, actions are probabilistic, we need to sum over possible transitions for ideal V, and property becomes:

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

-probabilities of getting to next state x (current value is immediate reward plus value of next state)
-$p_{a,0 \rightarrow s}$ 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, $r_{s,a,\hat{s}}$ and action transition matrix $p_{a,s \rightarrow \hat{s}}$

Defining the Q-Function

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

- Define intermediate function Q

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

- With some nice properties/relations:

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

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

Value Iteration (Value Based)

- **Direct**:
  - Initialize V(s) to all zeros
  - Take a series of random steps, then follow policy
  - Perform value iteration: $V(s) \leftarrow \max_{a \in A} \sum_{\hat{s} \in S} p_{a,s \rightarrow \hat{s}} \cdot (r_{s,a,\hat{s}} + \gamma V(\hat{s}))$
  - Repeat until V(s) stops changing
- **Q-Function Variant**:
  - Initialize Q(s,a) to all zeros
  - Take a series of random steps, then follow policy
  - Perform value iteration: $Q(s,a) \leftarrow \sum_{\hat{s} \in S} p_{a,s \rightarrow \hat{s}} \cdot (r_{s,a,\hat{s}} + \gamma \max_{a'} Q(\hat{s}, a'))$
  - Repeat until Q is not changing

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

With infinite time and exploration, 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
```

Lecture Notes for

Neural Networks and Machine Learning

Value Iteration and Q Learning

Next Time:
DeepQ-Learning, World Models

Reading: None