Deep Reinforcement Learning

Learning with dynamic datasets

Classes of Learning Problems

Supervised Learning

Data (x, y)

x is data, y is label

Goal

Learn function to map $x \rightarrow y$

Unsupervised Learning

Data x

x is data, no labels

Goal

Learn underlying structure

Reinforcement Learning

Data

State - action pairs

Goal

Maximize future rewards over many time steps

Apple example:

"Eat this to survive"

RL Key Concepts:

- Agent: takes actions (algorithm)
- Environment: where the agent operates
- Actions: a move the agent can make in the environment
- Action Space: the set of possible actions an agent can make in environment
- State: a situation that the agent percieves
- Reward: feedback that measures the success of an agents action

RL Key Concepts (cont.)

- Total Reward (Return)
  - $R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + r_{t+2} + \ldots + r_{t+n} + \ldots$
- Discounted Total Reward (Return)
  - $R_t = \sum_{i=t}^{\infty} \gamma^i r_i = \gamma^t r_t + \gamma^{(t+1)} r_{t+1} + \ldots + \gamma^{(t+n)} r_{t+n} + \ldots$
  - Discount factor γ: $0 < \gamma < 1$

Defining the Q-function

Total reward, R_t, is the discounted sum of all rewards obtained from time t

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

The Q-function captures the expected total future reward an agent in state, s, can recieve by executing a certain action, a.

How to take actions given a Q-function

$$Q(s_t, a_t) = \mathbb{E}(R_t | s_t, a_t)$$

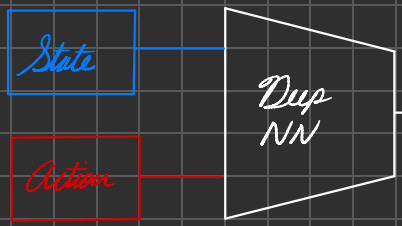
Ultimately, the agent needs a policy $\pi(s)$ to infer the best action to take at its state s

$$\pi^*(s) = \underset{a}{\text{argmax}}\, Q(s,a)$$

Deep Reinforcement Learning Algorithms

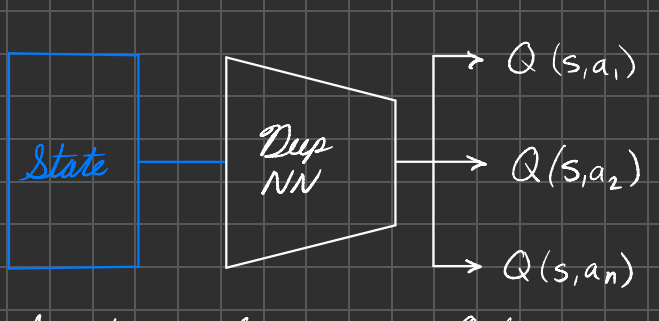
- Value Learning
  - Find $Q(s,a)$
  - $a = \underset{a}{\text{argmax}}\, Q(s,a)$
- Policy Learning
  - Find $\pi(s)$
  - Sample $a \sim \pi(s)$

Deep Q Neural Networks (DQN)



Action + State = Expected Return

Input Agent Output



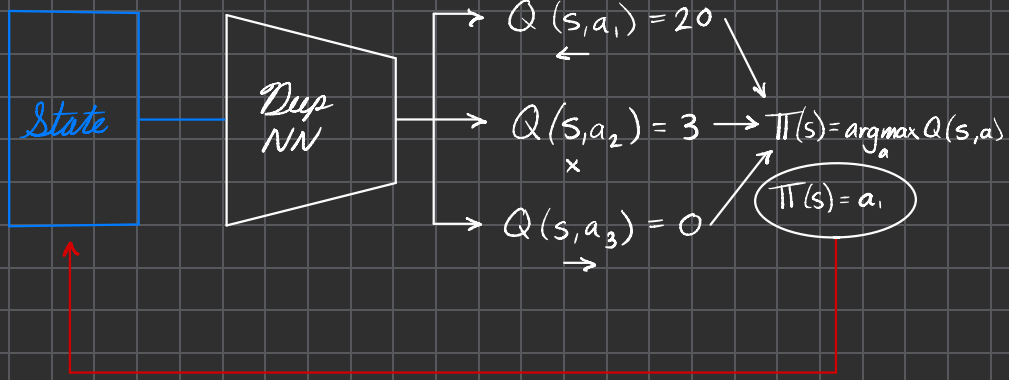
Input Agent Output

$$\text{Target} = \left(r + \gamma \max_{a'} Q(s',a')\right)$$

$$\text{Predicted} = Q(s,a)$$

$$\text{Q-Loss} = L = \mathbb{E}\left[\left\|\left(r + \gamma \max_{a'} Q(s',a')\right) - Q(s,a)\right\|^2\right]$$

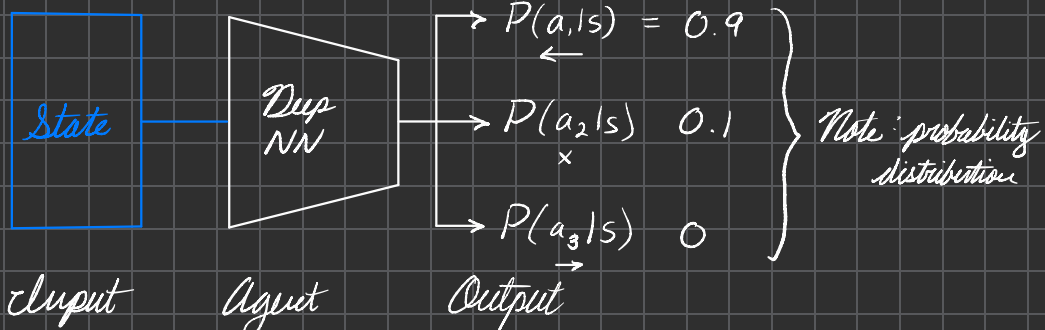
Deep Q Network Summary



Downsides of Q-Learning

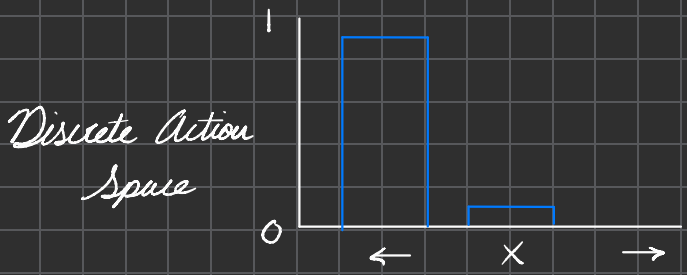
- Complexity
  - Can model scenarios with discrete action spaces
  - Cannot handle continuous action spaces
- Flexibility
  - Policy is deterministically computed from the Q function by maximizing the reward

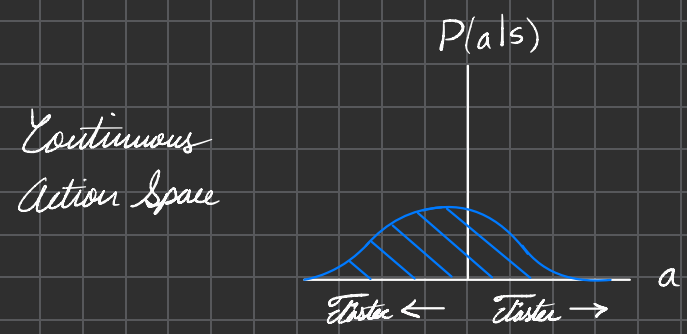
Policy Gradient (PG) Key idea



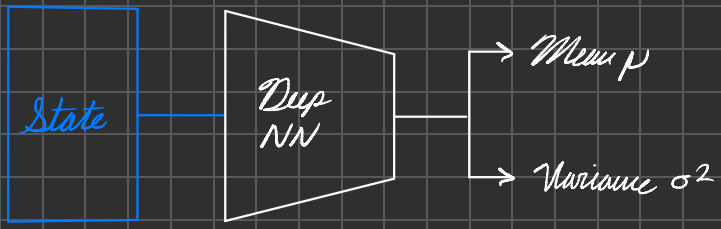
Directly optimize the policy $\pi(s)$

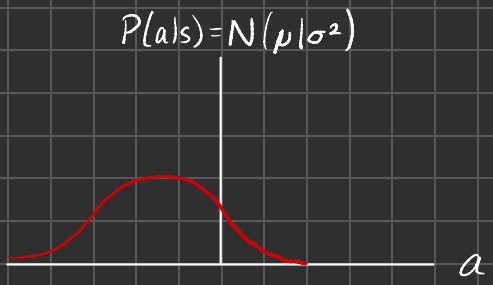
$\pi(s) \sim P(a|s) = a_1$ (in example above)





Policy Gradient Key Idea (cont.)





Training Policy Gradients: Case Study

Case Study: Self-Driving Cars

- Agent: Vehicle
- State: Camera, sensors, etc.
- Action: Steering wheel angle
- Reward: Distance traveled

Training Policy Gradients

Algorithm

1.) Initialize the agent

2.) Run a policy until termination

3.) Record all states, actions, rewards

4.) Decrease probability of actions that resulted in low reward

5.) Increase probability of actions that resulted in high reward

Loss Function

$$\text{loss} = \underbrace{-\log P(a_t|s_t)}_{\text{log-likelihood}} \underbrace{R_t}_{\text{Reward}}$$

Gradient Descent Update

$$w' = w - \nabla \text{loss}$$

$$w' = w + \nabla \log P(a_t|s_t) R_t$$

Reinforcement Learning in Real Life

- Try training models in simulation
  - Modern vision simulation is not accurate enough!
  - Sim-to-Real gap

Popular Applications of RL

- Game of Go
  - More legal board states than atoms in the universe

1.) Initial Training: Human Data

2.) Self-play and reinforcement learning

3.) "Intuition" about board state