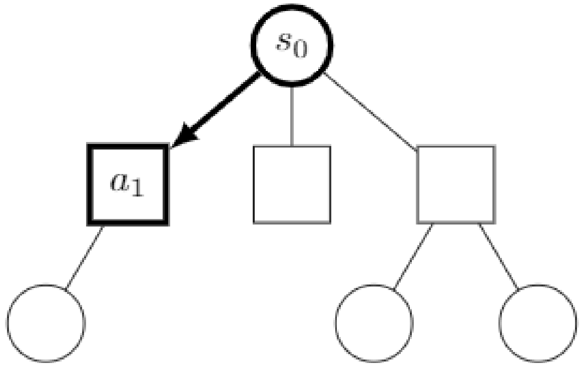
CSE 291 – AI Agents
Deep RL, pre-LLMs

Prithviraj Ammanabrolu

Thanks to David Silver's DeepMind RL Course and Rich Sutton's RL Book. Some slides were adapted form there.

Monte Carlo Tree Search

- 4 phases of building out and simulating paths along a search tree
- Various forms of this used in everything from Alpha Zero to modern LLM inference
- For arbitrary problem with start state s_0 and actions a_i
- All states have attributes:
  - Total simulation reward Q(s) and
  - Total no. of visits N(s)



Why Reinforcement Learning?

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy a.k.a. deliberation, reasoning, introspection, pondering, thought, search

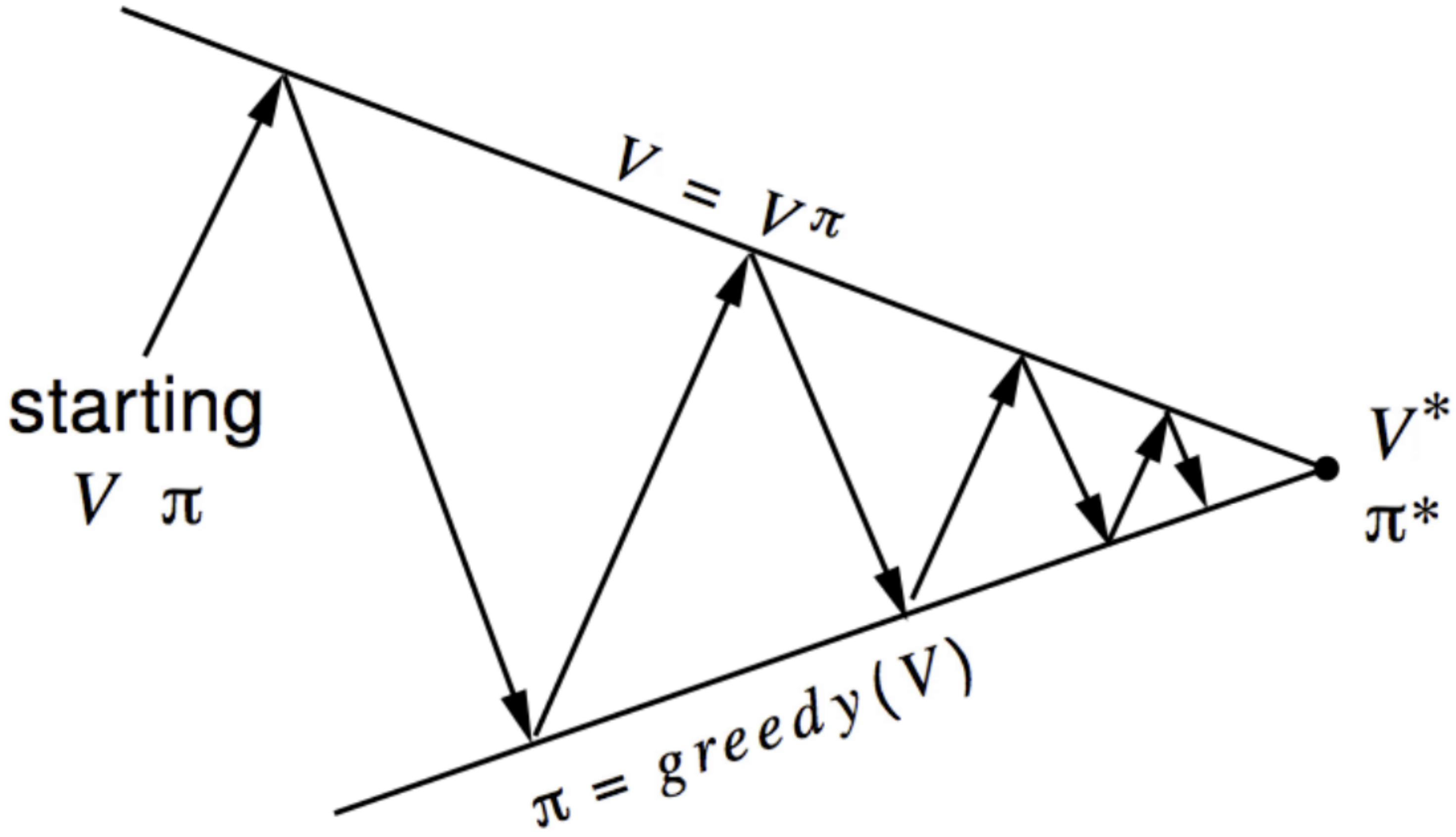
When to use DP

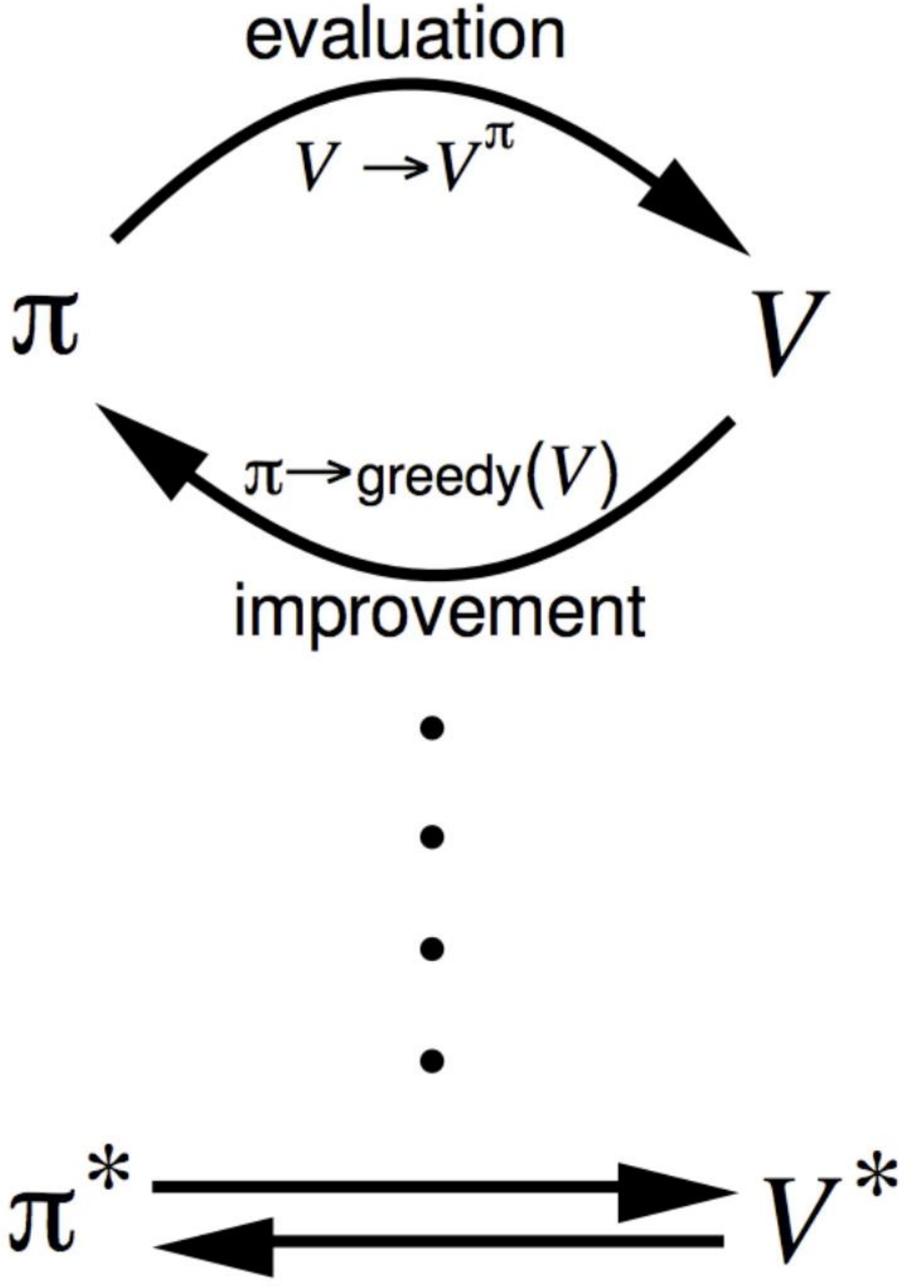
Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure:
  - Principle of optimality applies
  - Optimal solution can be decomposed into subproblems
- Overlapping subproblems:
  - Subproblems recur many times
  - Solutions can be cached and reused
- Markov decision processes satisfy both properties Bellman equation gives recursive decomposition Value function stores and reuses solutions

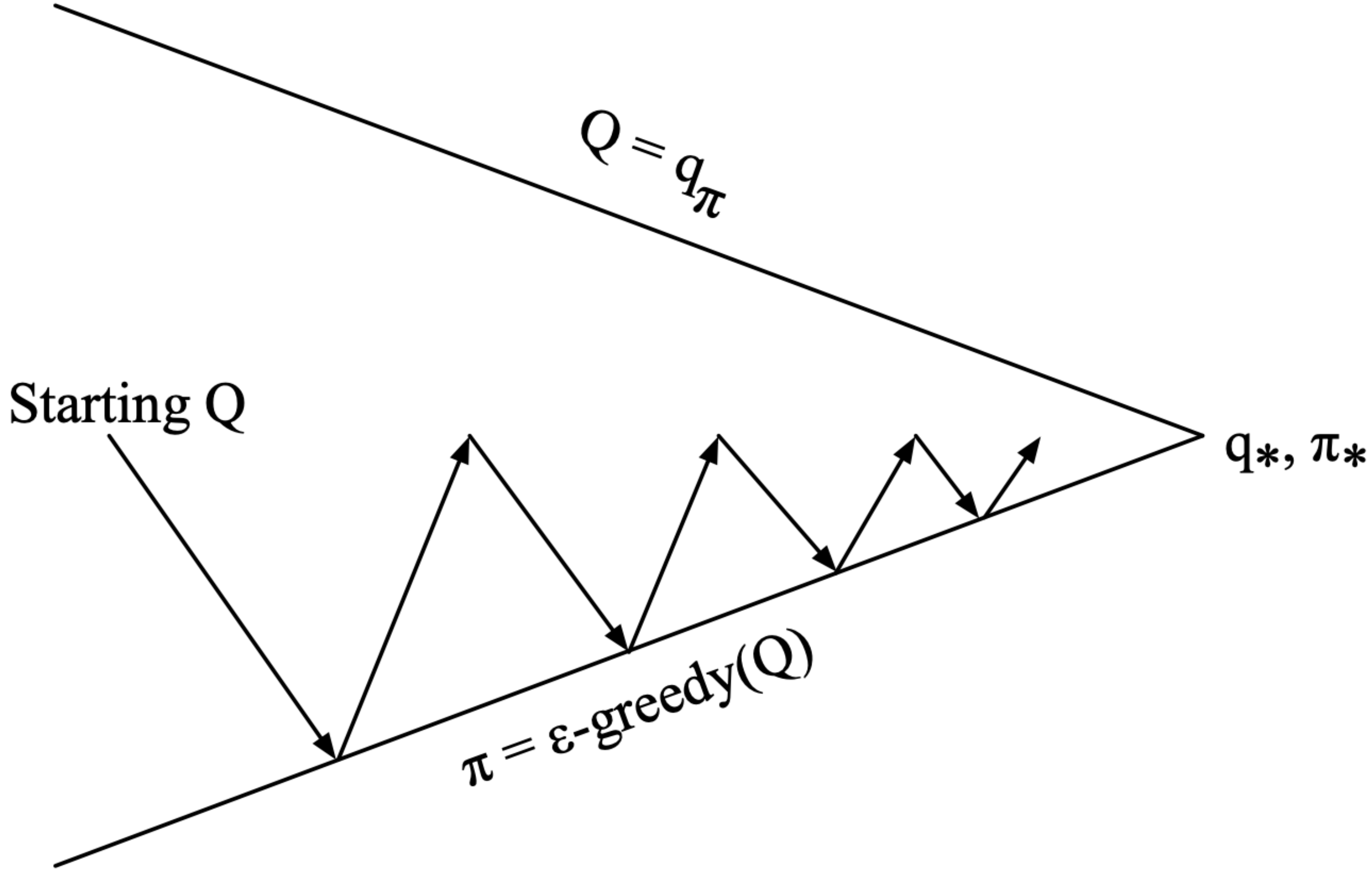
Generalized Policy Iteration

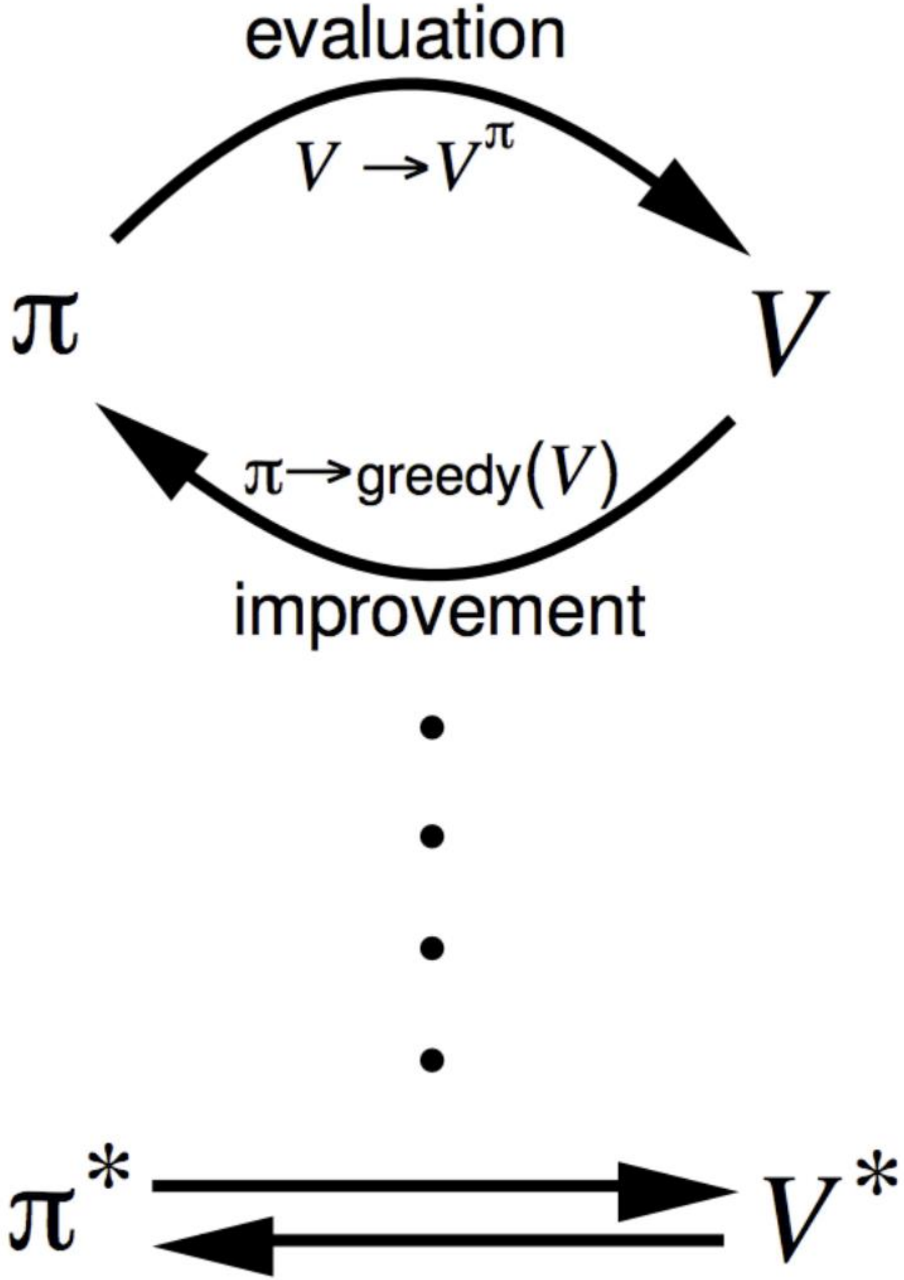
- Both are iterative versions of this





Generalized Policy Iteration with Fn Approximation + Monte Carlo Eval





You can't fully evaluate the entire state space each time

Issues with Monte Carlo estimates

- Need returns for whole trajectory
- The larger the state space is and the longer the horizon, the harder it is to get good estimates
- High variance, very dependent on “getting lucky” and seeing high return trajectories

How to fix? Temporal Difference

- With *Monte Carlo*, we update the value function from a complete episode, and so we **use the actual accurate discounted return of this episode.**

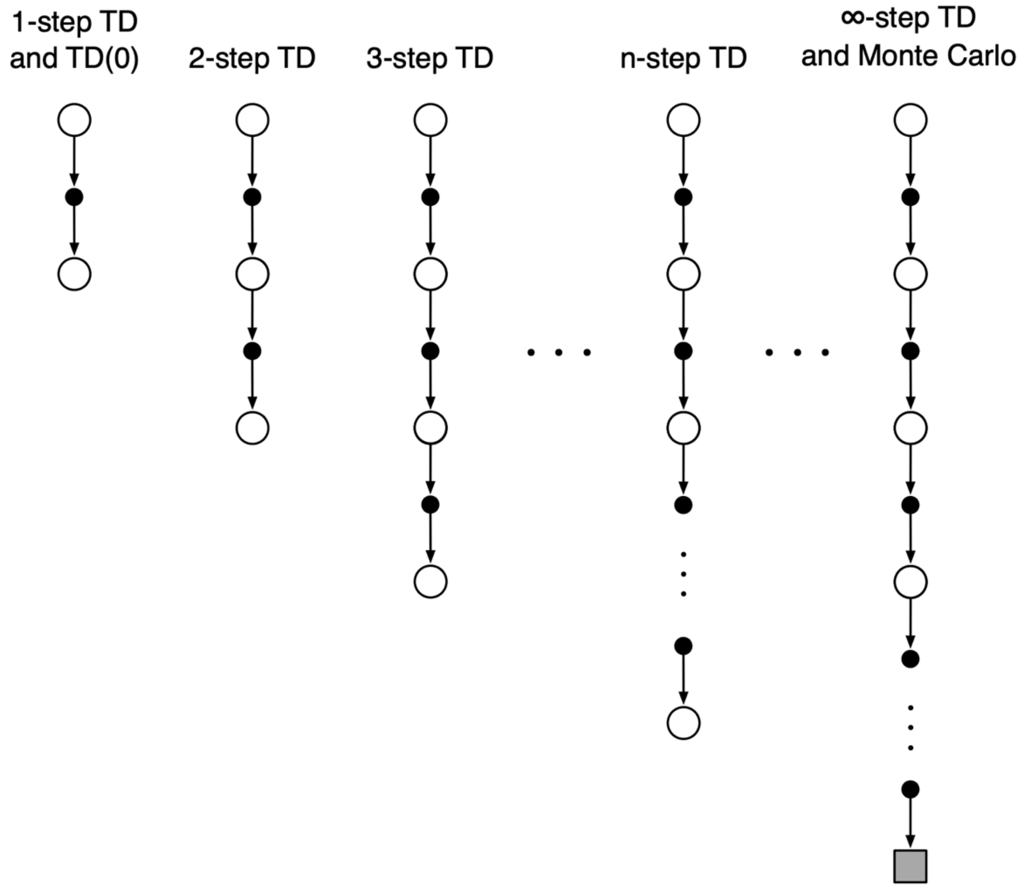
$$\text{Monte Carlo:} \quad V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- With *TD Learning*, we update the value function from a step, and we replace G_t, which we don't know, with **an estimated return called the TD target – a bootstrapping method similar to DP**

$$\text{TD Learning:} \quad V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

TD(0) → TD(∞)

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$



TD Advantages

- Temporal-difference (TD) learning has several advantages over Monte-Carlo (MC)
  - Lower variance
  - Online
  - Incomplete sequences
- Natural idea: use TD instead of MC in our control loop Apply TD to $Q(S, A)$
  - Use ε-greedy policy improvement
  - Update every time-step

TD Disadvantages

- Bootstrapping means you are chasing a moving target, stability of training very dependent on initialization

How to fix state space is very large

1. Learn from prior experiences

2. Function approximation

On Policy TD Learning - SARSA

- On Policy = learning the policy you are evaluating
- Will not cover SARSA as it is not really used anymore but will cover On Policy later on

Off-policy Learning

- Evaluate target policy $\pi(a|s)$ to compute $v_\pi(s)$ or $q_\pi(s, a)$
- While following behavior policy $\mu(a|s)$

$$\{S_1, A_1, R_2, \ldots, S_T\} \sim \mu$$

Why is this important?

- Learn from observing humans or other agents
- Re-use experience generated from old policies $\pi_1, \pi_2, \ldots, \pi_{t-1}$
- Learn about optimal policy while following exploratory policy
- Learn about multiple policies while following one policy

Q-Learning

- We now consider off-policy learning of action-values $Q(s, a)$
- Next action is chosen using behavior policy $A_{t+1} \sim \mu(\cdot|S_t)$
- But we consider alternative successor action $A' \sim \pi(\cdot|S_t)$
- And update $Q(S_t , A_t)$ towards value of alternative action from policy you're actually evaluating

$$Q(S_t , A_t) \leftarrow Q(S_t , A_t) + \alpha \, R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t , A_t)$$

Q-Learning

- We now allow both behavior and target policies to improve
- The target policy π is greedy w.r.t. Q(s, a)
- $\pi(S_{t+1}) = \text{argmax}_{a'} Q(S_{t+1}, a')$
- The behavior policy μ is e.g. -greedy w.r.t. Q(s, a)
- The Q-learning target then simplifies:

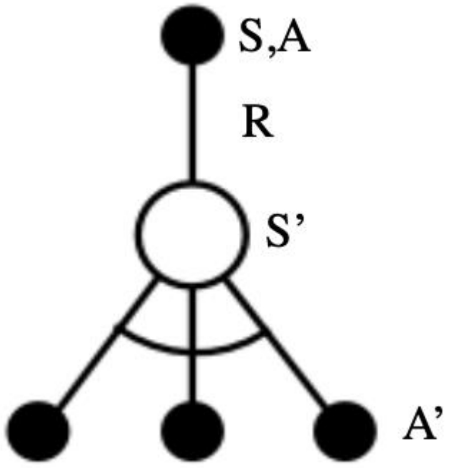
$$R_{t+1} + \gamma Q(S_{t+1}, A\ 0\ )$$

$$= R_{t+1} + \gamma Q(S_{t+1}, \text{argmax}_{a'} Q(S_{t+1}, a'))$$

$$= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')$$

Q-Learning

- $Q(S, A) \leftarrow Q(S, A) + \alpha\,(R + \gamma \max_{a'} Q(S', a') - Q(S, A))$
- Q-learning control converges to the optimal action-value function, $Q(s, a) \rightarrow q_*(s, a)$



Q-Learning Full Algorithm

Initialize $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ε-greedy)
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$;
    until S is terminal

Kinda Large Scale RL

- Reinforcement learning can be used to solve large problems, e.g.
  - Backgammon: 10^{20} states
  - Computer Go: 10^{170} states
  - Helicopter: continuous state space
- How can we scale up the model-free methods for prediction and control from the last two lectures?

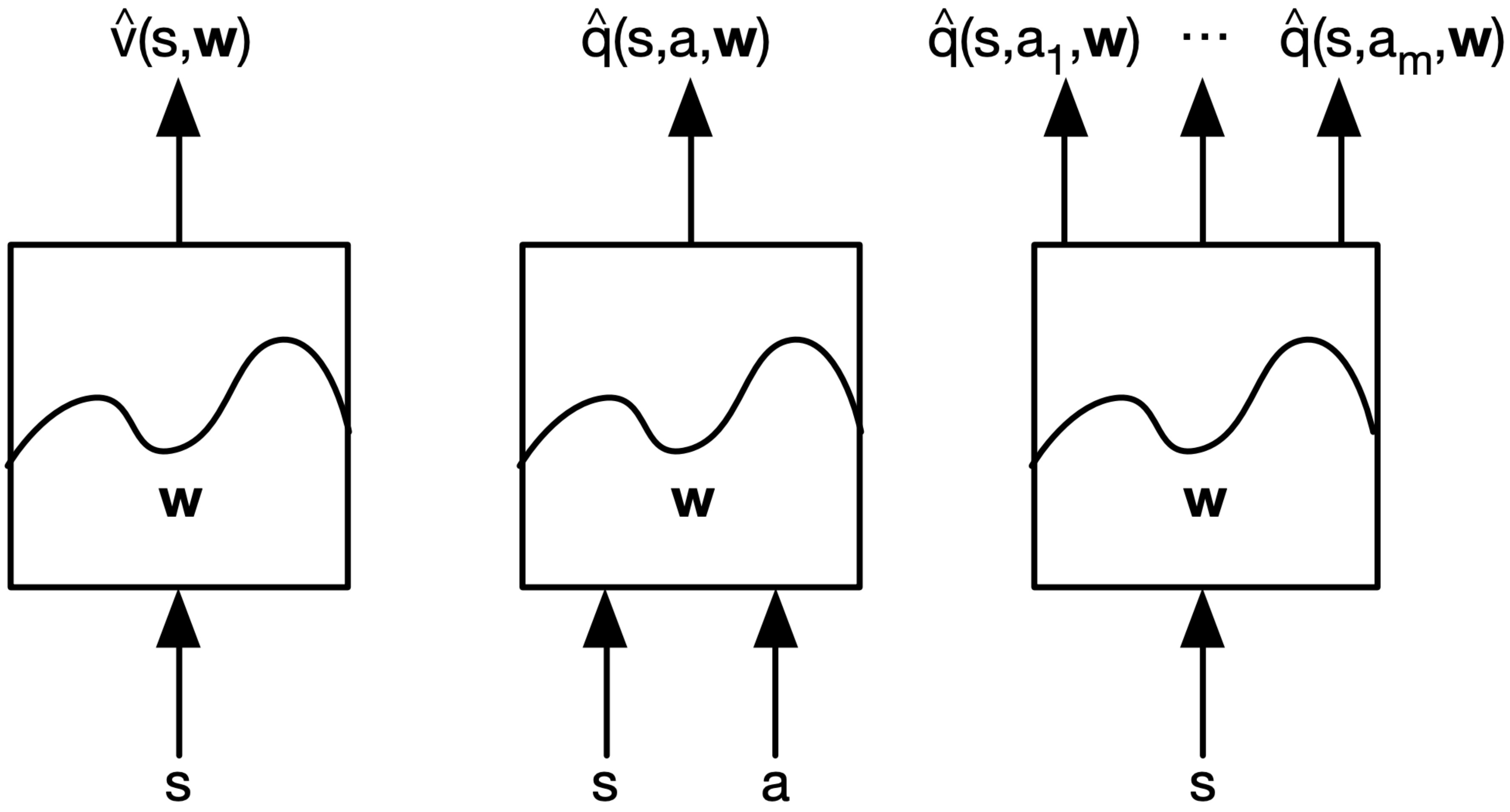
Value Function Approximation

- So far we have represented value function by a lookup table
- Every state s has an entry V(s)
- Or every state-action pair s, a has an entry Q(s, a)
- Problem with large MDPs:
  - There are too many states and/or actions to store in memory
  - It is too slow to learn the value of each state individually
- Solution for large MDPs:
  - Estimate value function with function approximation

$$\hat{v}(s, w) \approx v_\pi(s) \text{ or } \hat{q}(s, a, w) \approx q_\pi(s, a)$$

  - Generalize from seen states to unseen states
  - Update parameter w using MC or TD learning

Types of Value Function Approximators



Action-value Function Approximation

- Approximate the action-value function

  $$\hat{q}(S, A, w) \approx q_\pi(S, A)$$

- Minimize mean-squared error between approximate action-value fn $\hat{q}(S, A, w)$ and true action-value fn $q_\pi(S, A)$

  $$J(w) = E_\pi \left[(q_\pi(S, A) - \hat{q}(S, A, w))^2\right]$$

- Use stochastic gradient descent to find a local minimum

  $$-1/2\ \nabla_w J(w) = (q_\pi(S, A) - \hat{q}(S, A, w))\nabla_w \hat{q}(S, A, w)$$

  $$\Delta w = \alpha(q_\pi(S, A) - \hat{q}(S, A, w))\nabla_w \hat{q}(S, A, w)$$

Deep Neural Nets as function approx.

- Need a Neural Net that is actually able to effectively encode observations and actions
- For the original Atari, this was CNNs
- These days, it is transformers
- Note that you generally need hundreds of k to millions of steps for most environments. The bigger your policy the slower this is

Deep Q Network - DQN

- You actually know all the pieces now
- You put Q-learning together with the function approximation

General loop

Algorithm 1 Deep Q-learning with Experience Replay

$\boxed{\begin{array}{l}\text{Initialize replay memory } \mathcal{D} \text{ to capacity } N \\ \text{Initialize action-value function } Q \text{ with random weights}\end{array}}$

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

General loop

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

General loop

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

General loop

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}
    $\boxed{\text{Set } s_{t+1} = s_t, a_t, x_{t+1} \text{ and preprocess } \phi_{t+1} = \phi(s_{t+1})}$
    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

General loop

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 $\phi_{t+1} = \phi(s_{t+1})$
    $\boxed{\text{Store transition } (\phi_t, a_t, r_t, \phi_{t+1}) \text{ in } \mathcal{D}}$
    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

General loop

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

General loop

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 $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    $\boxed{\text{Perform a gradient descent step on } (y_j - Q(\phi_j, a_j; \theta))^2 \text{ according to equation 3}}$
  end for
end for

Where are we now?

- GPU go brrrr as solution to large state space RL
- Algorithms still not particularly efficient
- No guarantees on anything