

# **Value-Based Reinforcement Learning**

**Shusen Wang**

# Action-Value Functions

# Discounted Return

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$



- The return depends on actions  $A_t, A_{t+1}, A_{t+2}, \dots$  and states  $S_t, S_{t+1}, S_{t+2}, \dots$
- Actions are random:  $\mathbb{P}[A = a \mid S = s] = \pi(a|s)$ . (Policy function.)
- States are random:  $\mathbb{P}[S' = s' \mid S = s, A = a] = p(s'|s, a)$ . (State transition.)

# Action-Value Functions $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$

**Definition:** Action-value function for policy  $\pi$ .

- $Q_\pi(s_t, a_t) = \mathbb{E} [U_t | S_t = s_t, A_t = a_t].$



- Taken w.r.t. actions  $A_{t+1}, A_{t+2}, A_{t+3}, \dots$  and states  $S_{t+1}, S_{t+2}, S_{t+3}, \dots$
- Integrate out everything except for the observations:  $A_t = a_t$  and  $S_t = s_t$ .

# Action-Value Functions $Q(s, a)$

**Definition:** Discounted return (aka cumulative discounted future reward).

- $U_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots$

**Definition:** Action-value function for policy  $\pi$ .

- $Q_\pi(s_t, a_t) = \mathbb{E} [U_t | S_t = s_t, A_t = a_t].$

**Definition:** Optimal action-value function.

- $Q^*(s_t, a_t) = \max_{\pi} Q_\pi(s_t, a_t).$
- Whatever policy function  $\pi$  is used, the result of taking  $a_t$  at state  $s_t$  cannot be better than  $Q^*(s_t, a_t).$

# Deep Q-Network (DQN)

# Approximate the Q Function

**Goal:** Win the game ( $\approx$  maximize the total reward.)


**Question:** If we know  $Q^*(s, a)$ , what is the best action?

# Approximate the Q Function

**Goal:** Win the game ( $\approx$  maximize the total reward.)

**Question:** If we know  $Q^*(s, a)$ , what is the best action?

- Obviously, the best action is  $a^* = \operatorname{argmax}_a Q^*(s, a)$ .



$Q^*$  is an indication for how good it is for an agent to pick action  $a$  while being in state  $s$ .



# Approximate the Q Function

**Goal:** Win the game ( $\approx$  maximize the total reward.)

**Question:** If we know  $Q^*(s, a)$ , what is the best action?

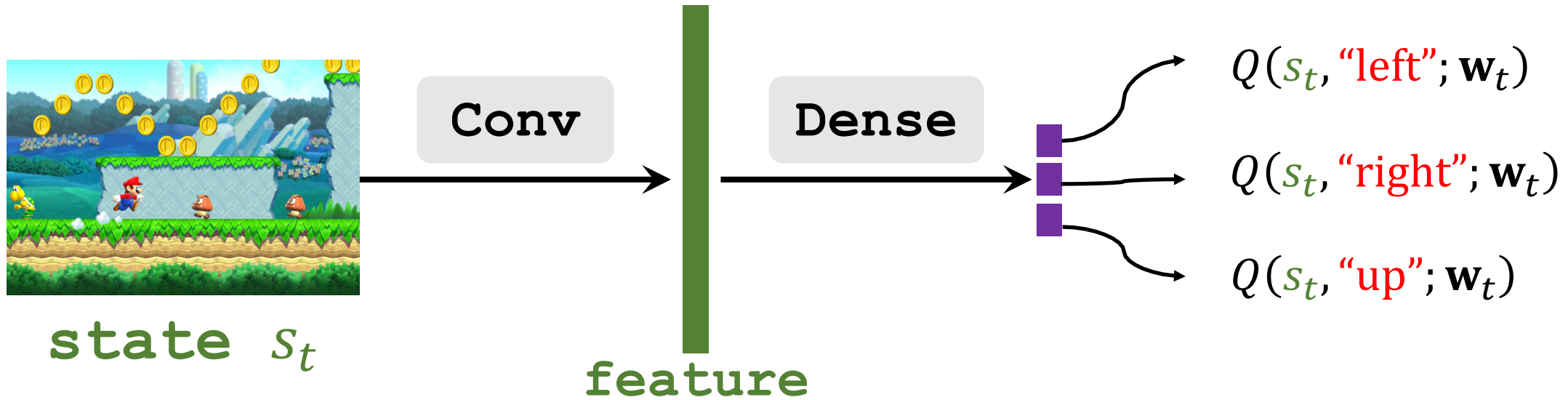
- Obviously, the best action is  $a^* = \underset{a}{\operatorname{argmax}} Q^*(s, a)$ .

**Challenge:** We do not know  $Q^*(s, a)$ .

- Solution: Deep Q Network (DQN)
- Use neural network  $Q(s, a; \mathbf{w})$  to approximate  $Q^*(s, a)$ .

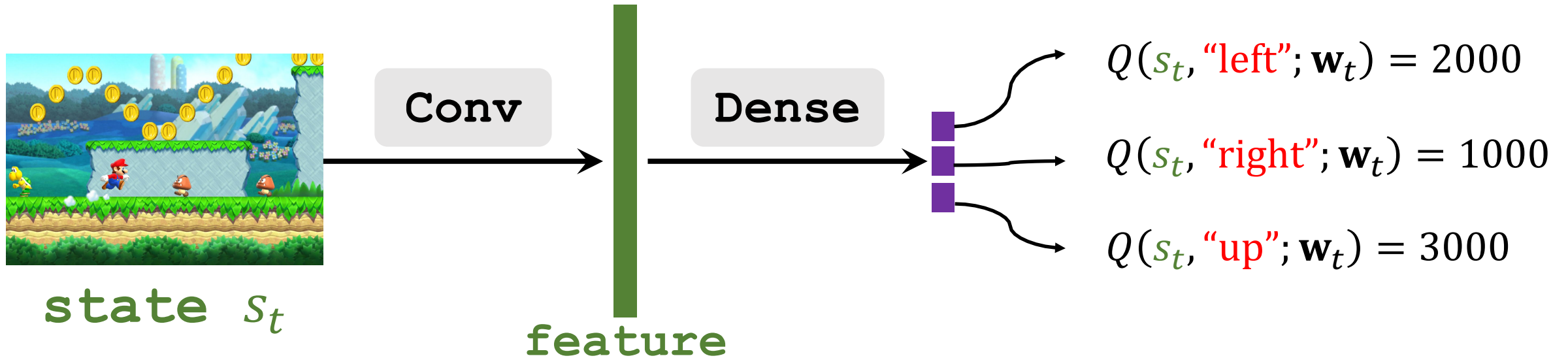
# Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



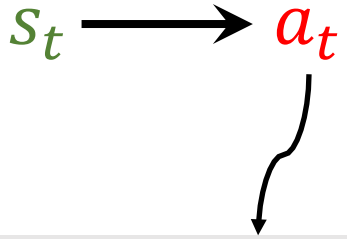
# Deep Q Network (DQN)

- Input shape: size of the screenshot.
- Output shape: dimension of action space.



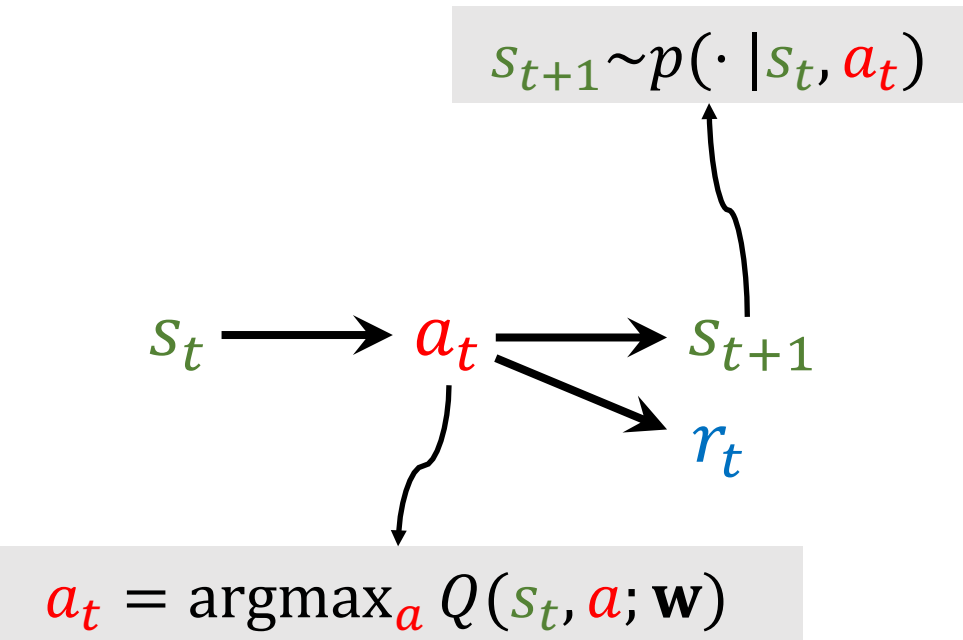
**Question:** Based on the predictions, what should be the **action**?

# Apply DQN to Play Game

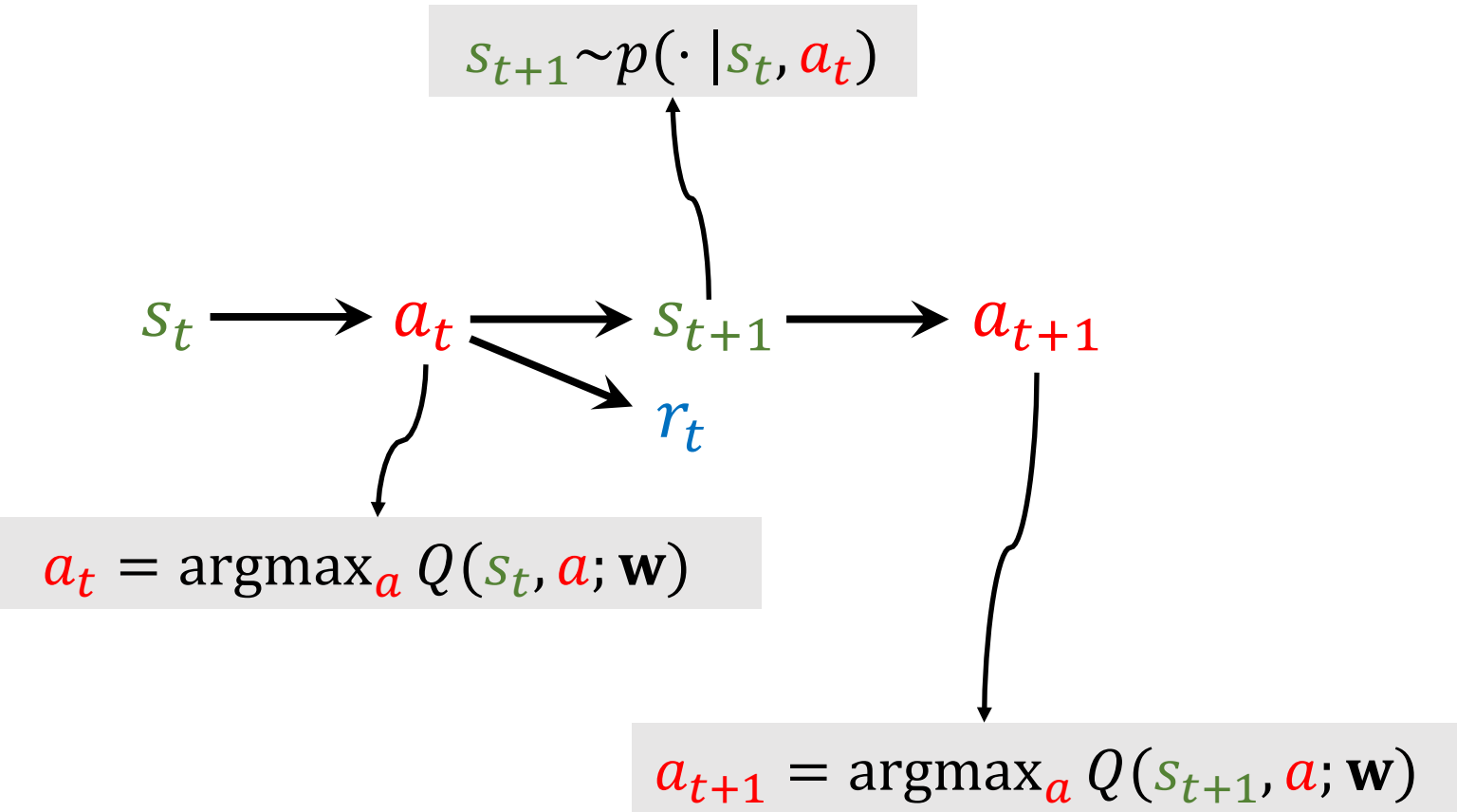


$$a_t = \operatorname{argmax}_a Q(s_t, a; \mathbf{w})$$

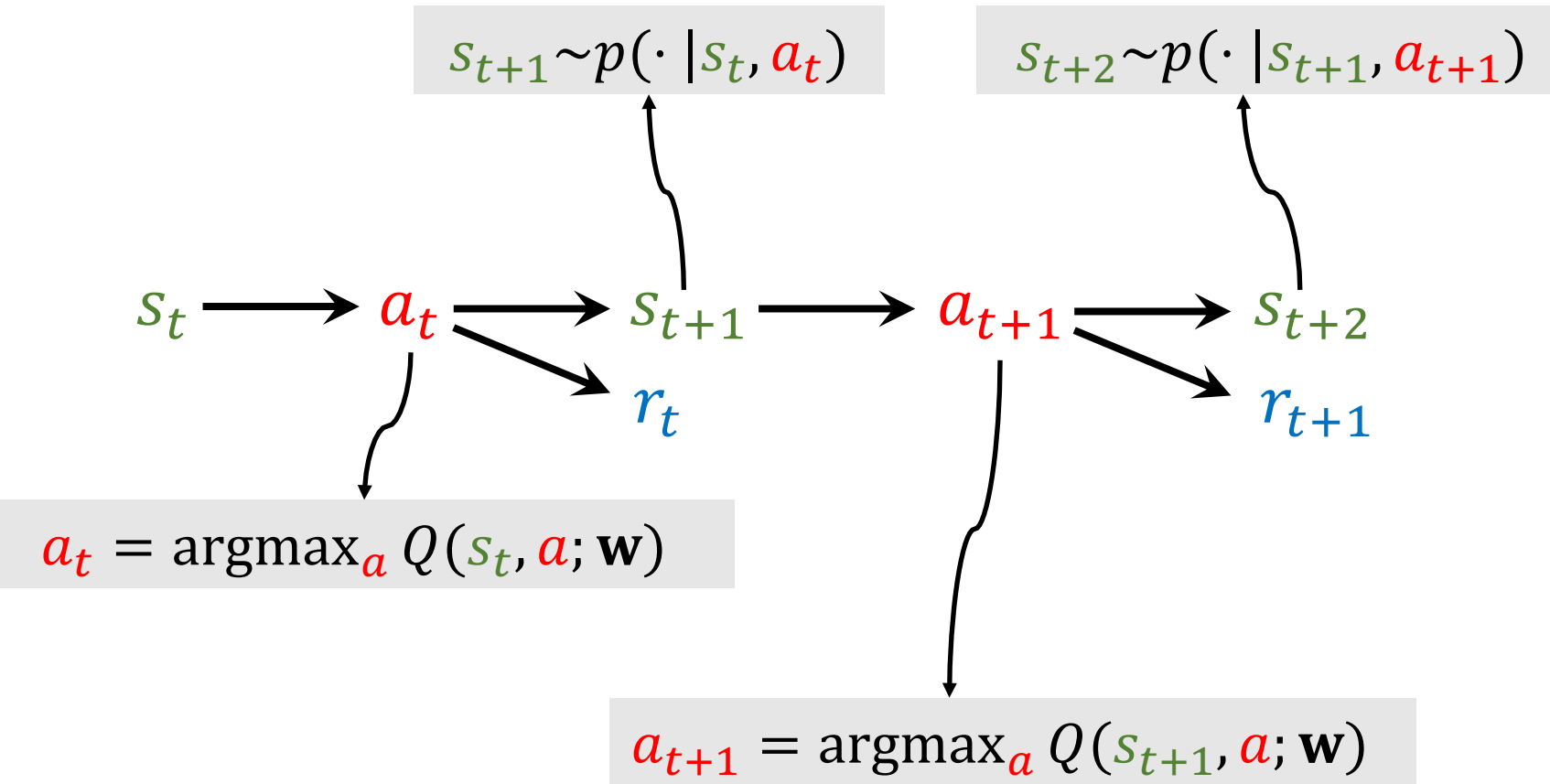
# Apply DQN to Play Game



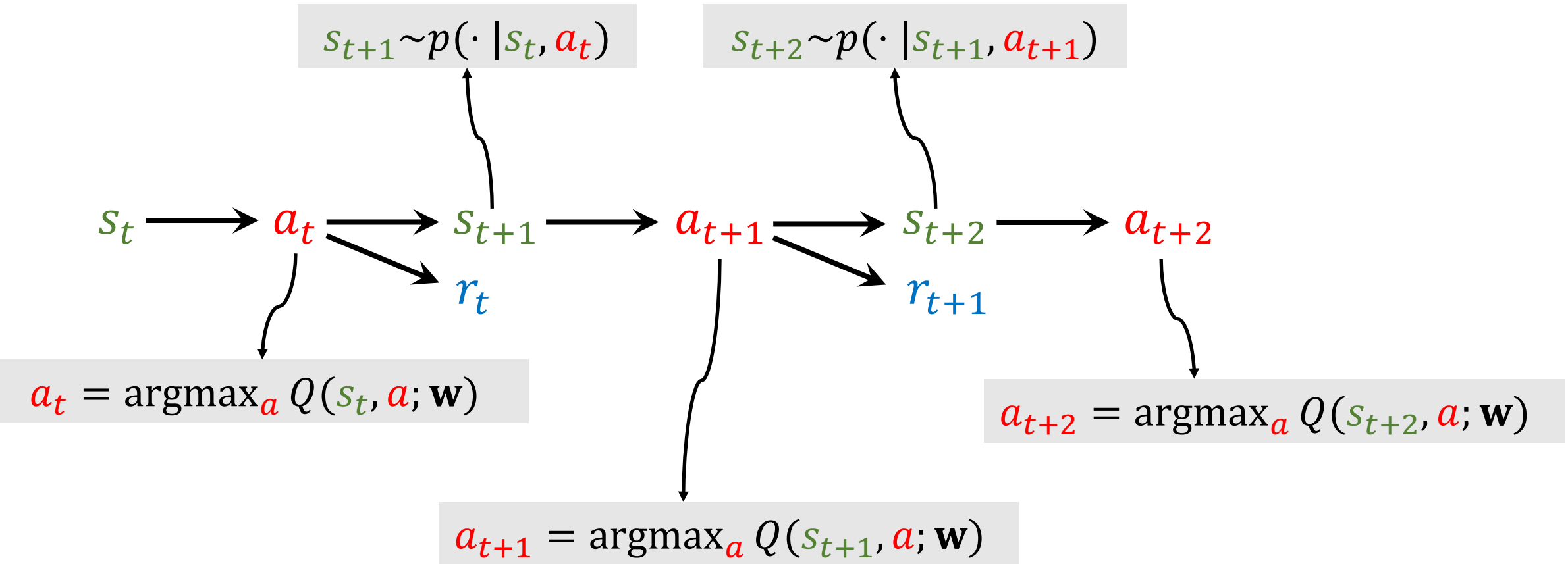
# Apply DQN to Play Game



# Apply DQN to Play Game

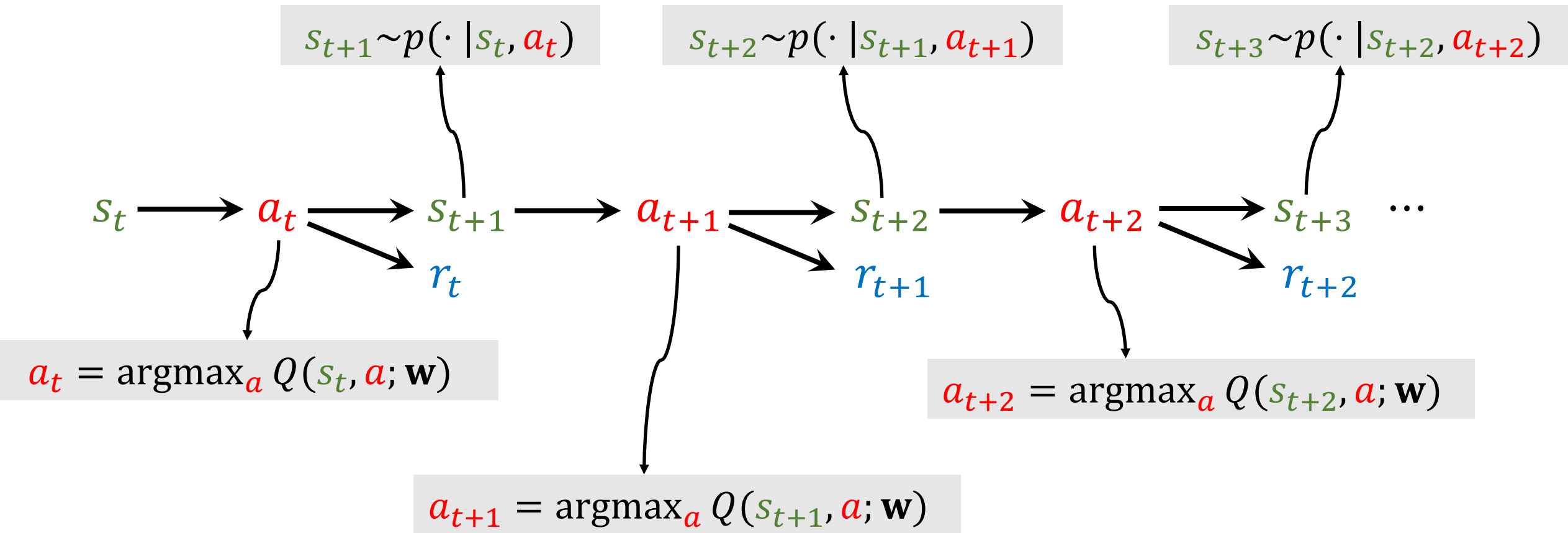


# Apply DQN to Play Game





# Apply DQN to Play Game



# Temporal Difference (TD) Learning

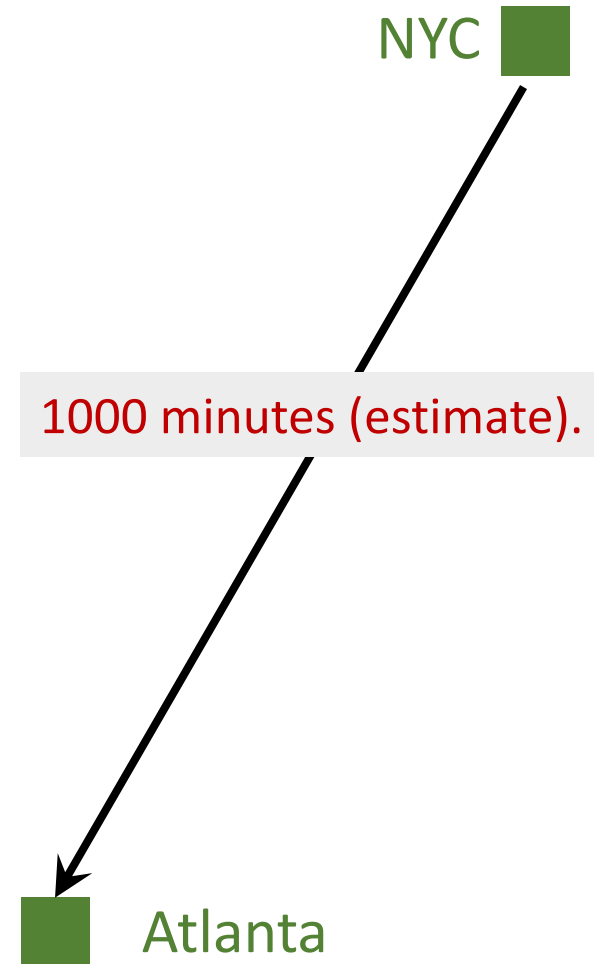
## Reference

1. Sutton and others: [A convergent  \$O\(n\)\$  algorithm for off-policy temporal-difference learning with linear function approximation](#). In *NIPS*, 2008.
2. Sutton and others: [Fast gradient-descent methods for temporal-difference learning with linear function approximation](#). In *ICML*, 2009.

# Example

- I want to drive from NYC to Atlanta.
- Model  $Q(\mathbf{w})$  estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

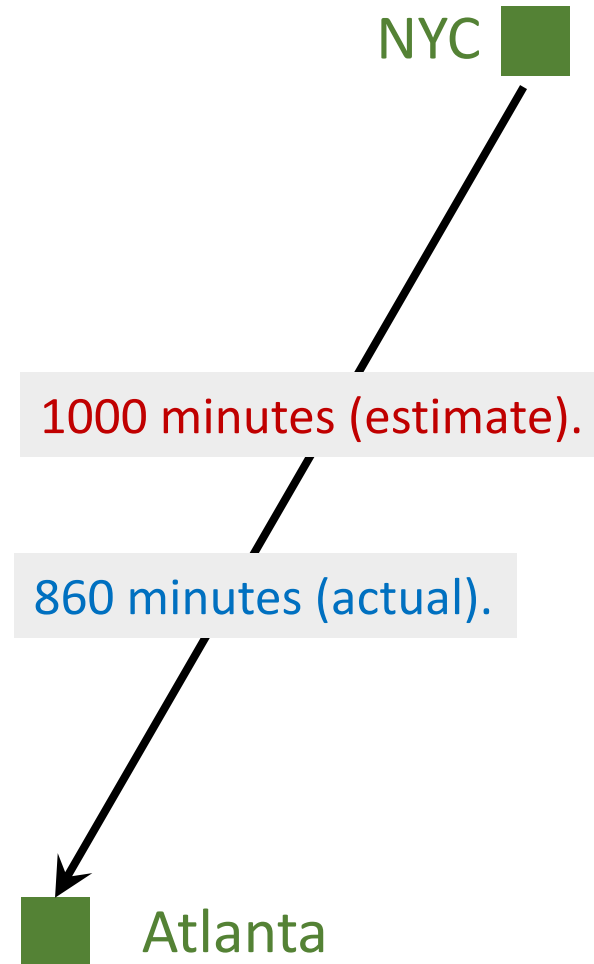


# Example

- I want to drive from NYC to Atlanta.
- Model  $Q(\mathbf{w})$  estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

- Make a prediction:  $q = Q(\mathbf{w})$ , e.g.,  $q = 1000$ .
- Finish the trip and get the target  $y$ , e.g.,  $y = 860$ .

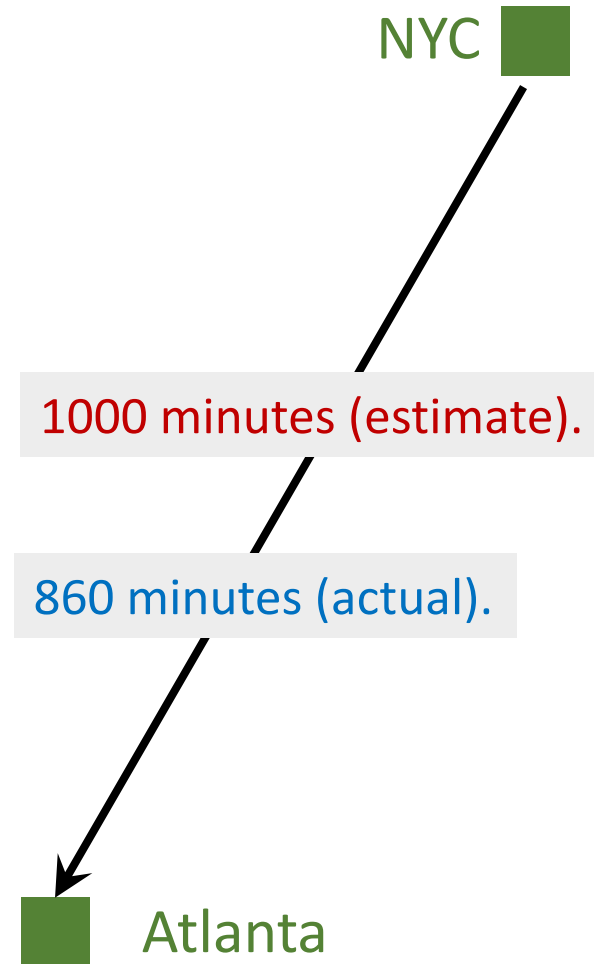


# Example

- I want to drive from NYC to Atlanta.
- Model  $Q(\mathbf{w})$  estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

- Make a prediction:  $q = Q(\mathbf{w})$ , e.g.,  $q = 1000$ .
- Finish the trip and get the target  $y$ , e.g.,  $y = 860$ .
- Loss:  $L = \frac{1}{2}(q - y)^2$ .
- Gradient:  $\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial q}{\partial \mathbf{w}} \cdot \frac{\partial L}{\partial q} = (q - y) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$ .
- Gradient descent:  $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$ .

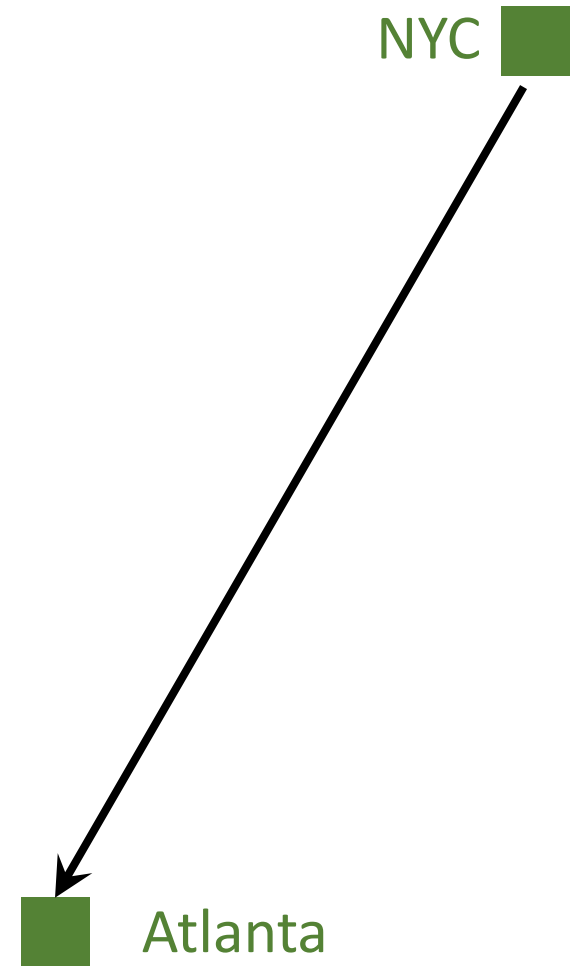


# Example

- I want to drive from NYC to Atlanta.
- Model  $Q(\mathbf{w})$  estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

- Can I update the model **before finishing the trip**?

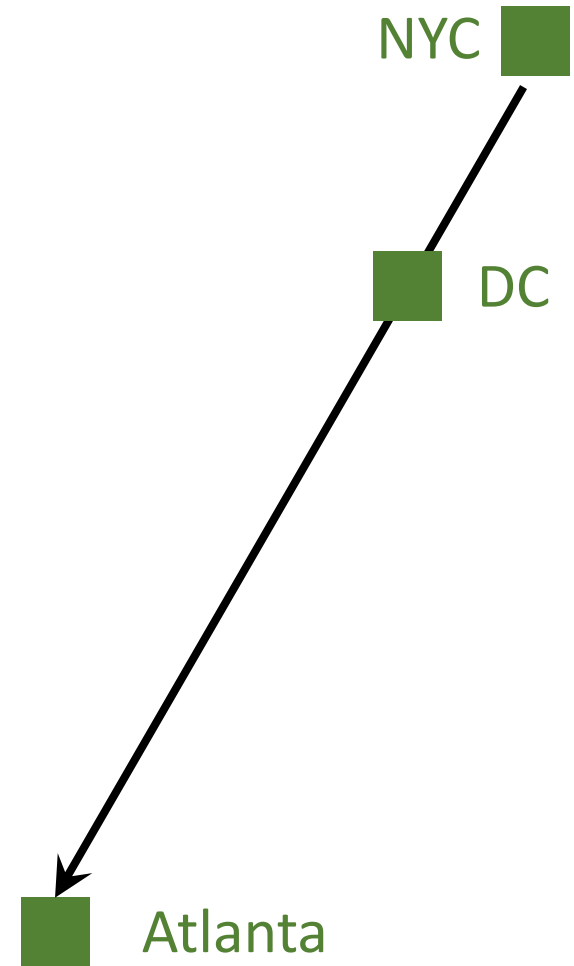


# Example

- I want to drive from NYC to Atlanta (via DC).
- Model  $Q(\mathbf{w})$  estimates the time cost, e.g., 1000 minutes.

**Question:** How do I update the model?

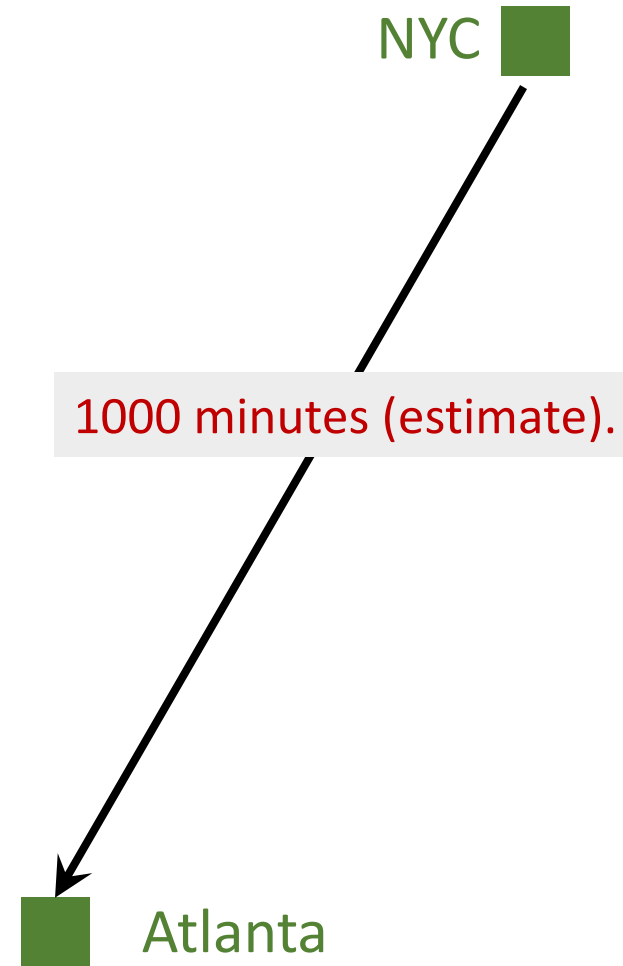
- Can I update the model before finishing the trip?
- Can I get a better  $\mathbf{w}$  as soon as I arrived at DC?



# Temporal Difference (TD) Learning

- Model's estimate:

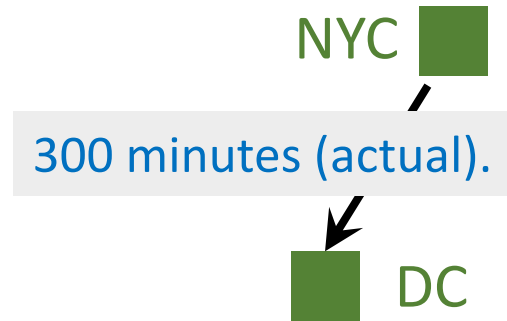
NYC to Atlanta: 1000 minutes (estimate).





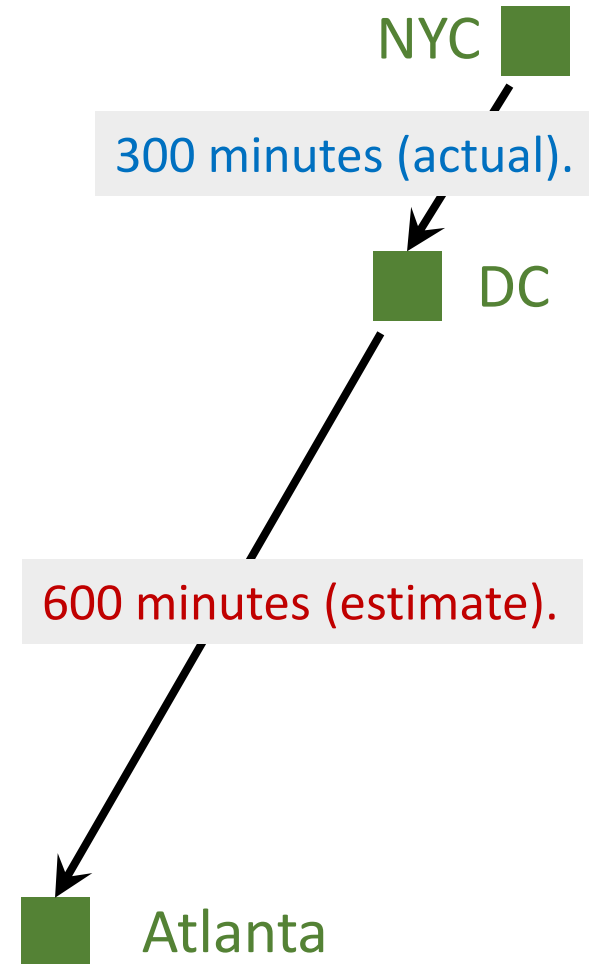
# Temporal Difference (TD) Learning

- Model's estimate:  
    NYC to Atlanta: 1000 minutes (estimate).
- I arrived at DC; actual time cost:  
    NYC to DC: 300 minutes (actual).



# Temporal Difference (TD) Learning

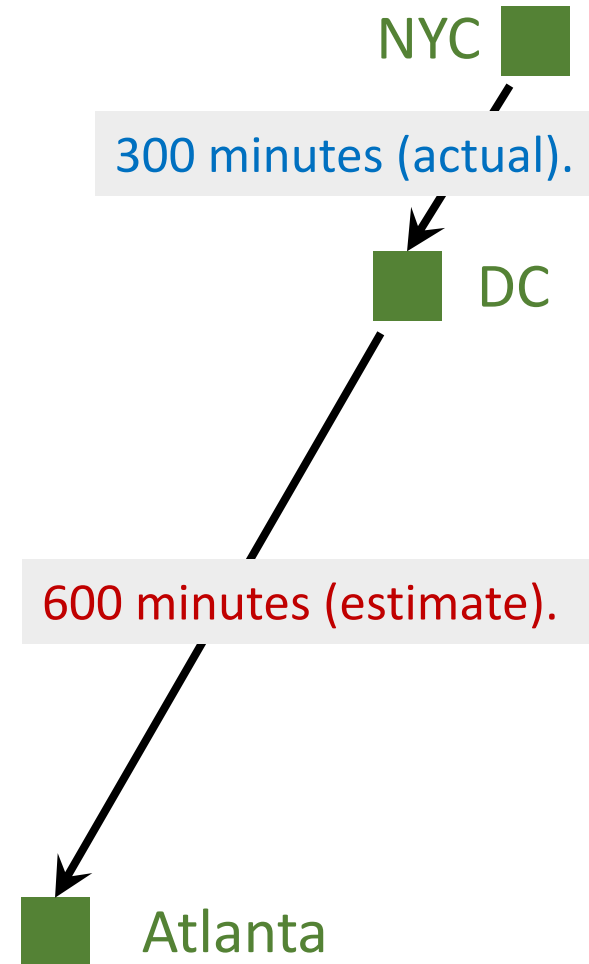
- Model's estimate:  
NYC to Atlanta: 1000 minutes (estimate).
- I arrived at DC; actual time cost:  
NYC to DC: 300 minutes (actual).
- Model now updates its estimate:  
DC to Atlanta: 600 minutes (estimate).



# Temporal Difference (TD) Learning

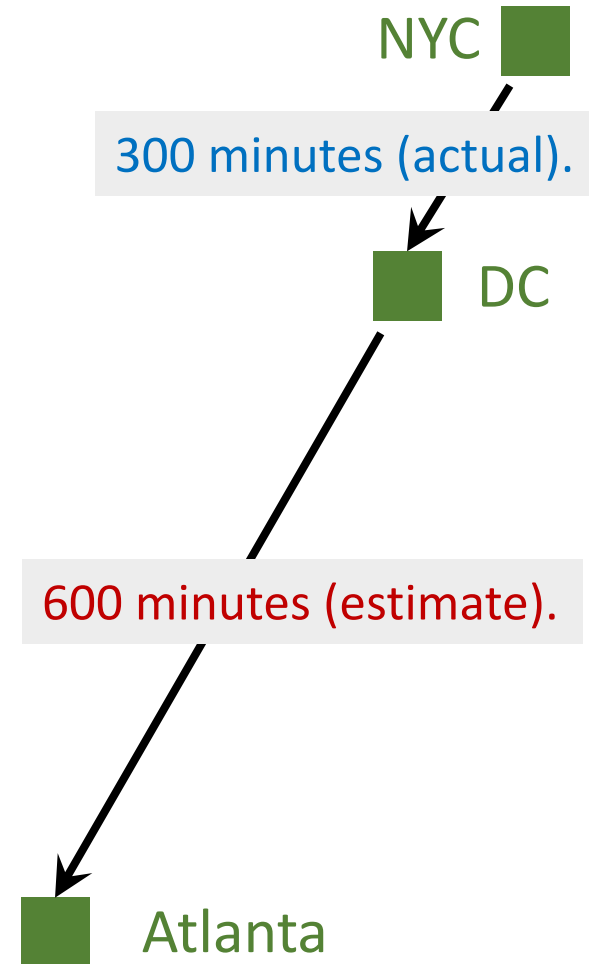
- Model's estimate:  $Q(\mathbf{w}) = 1000$  minutes.
- Updated estimate:  $300 + 600 = 900$  minutes.

TD target.



# Temporal Difference (TD) Learning

- Model's estimate:  $Q(\mathbf{w}) = 1000$  minutes.
- Updated estimate:  $300 + 600 = 900$  minutes.  
↓  
TD target.
- TD target  $y = 900$  is a more reliable estimate than  $1000$ .



# Temporal Difference (TD) Learning

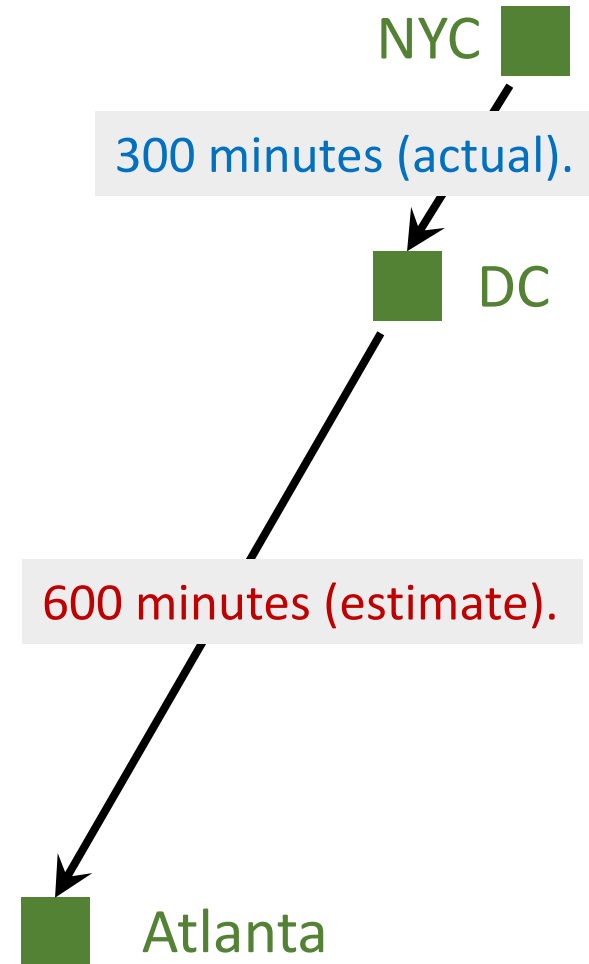
- Model's estimate:  $Q(\mathbf{w}) = 1000$  minutes.
- Updated estimate:  $300 + 600 = 900$  minutes.

TD target.

- TD target  $y = 900$  is a more reliable estimate than  $1000$ .

- Loss:  $L = \frac{1}{2} (Q(\mathbf{w}) - y)^2$ .

TD error



# Temporal Difference (TD) Learning

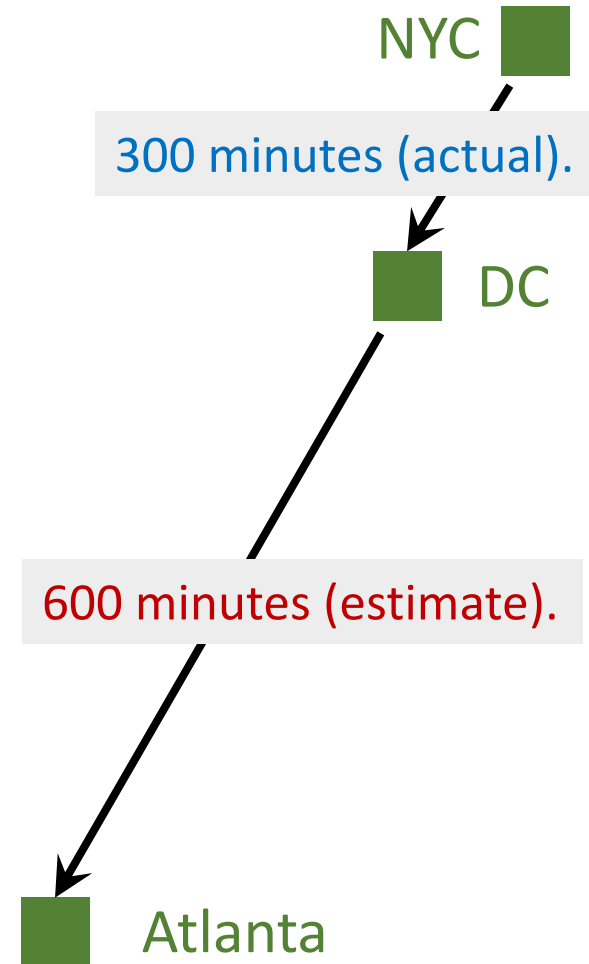
- Model's estimate:  $Q(\mathbf{w}) = 1000$  minutes.
- Updated estimate:  $300 + 600 = 900$  minutes.

TD target.

- TD target  $y = 900$  is a more reliable estimate than  $1000$ .

- Loss:  $L = \frac{1}{2} (Q(\mathbf{w}) - y)^2$ .

- Gradient:  $\frac{\partial L}{\partial \mathbf{w}} = \underbrace{(1000 - 900)}_{\text{TD error}} \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$ .

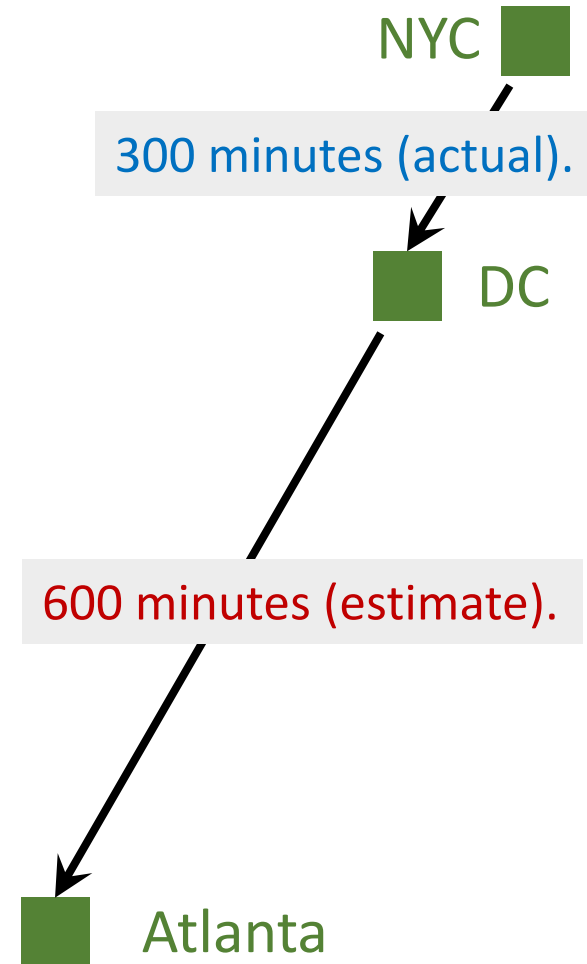


# Temporal Difference (TD) Learning

- Model's estimate:  $Q(\mathbf{w}) = 1000$  minutes.
- Updated estimate:  $300 + 600 = 900$  minutes.

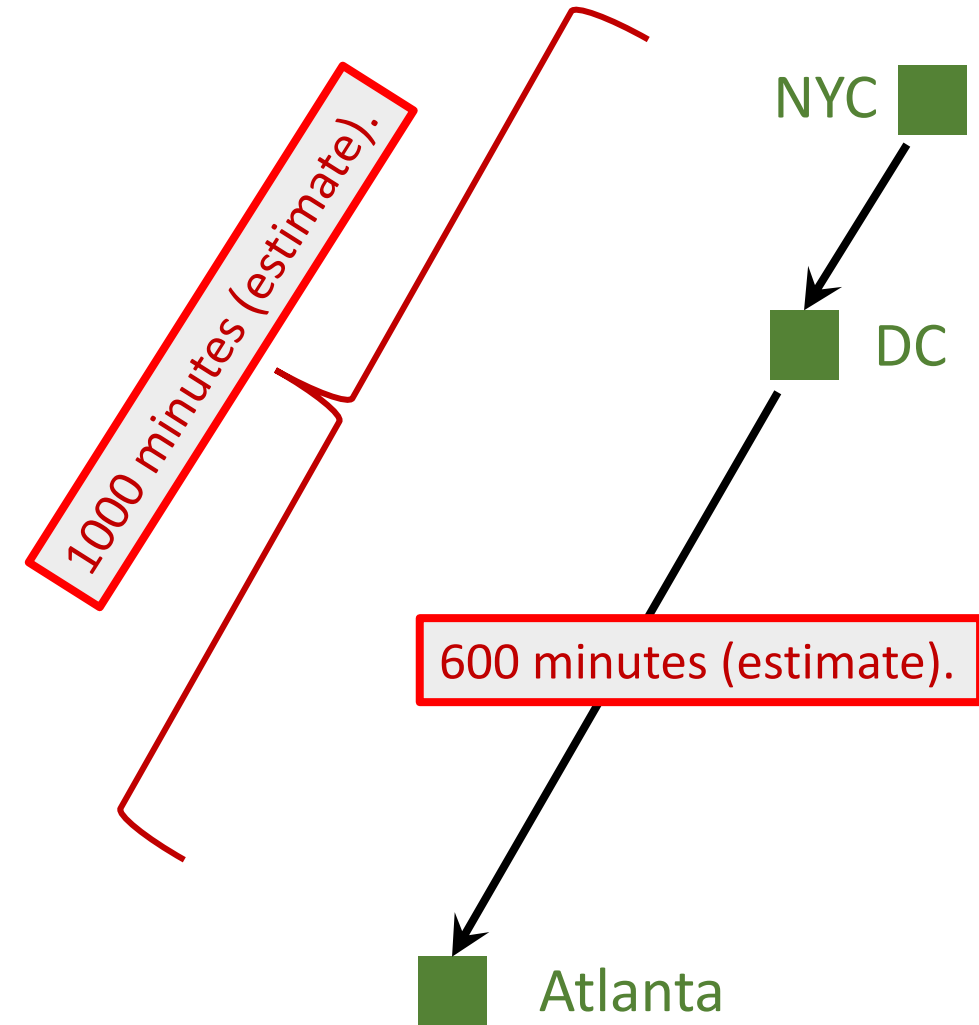
↘  
TD target.

- TD target  $y = 900$  is a more reliable estimate than  $1000$ .
- Loss:  $L = \frac{1}{2} (Q(\mathbf{w}) - y)^2$ .
- Gradient:  $\frac{\partial L}{\partial \mathbf{w}} = (1000 - 900) \cdot \frac{\partial Q(\mathbf{w})}{\partial \mathbf{w}}$ .
- Gradient descent:  $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$ .



# Why does TD learning work?

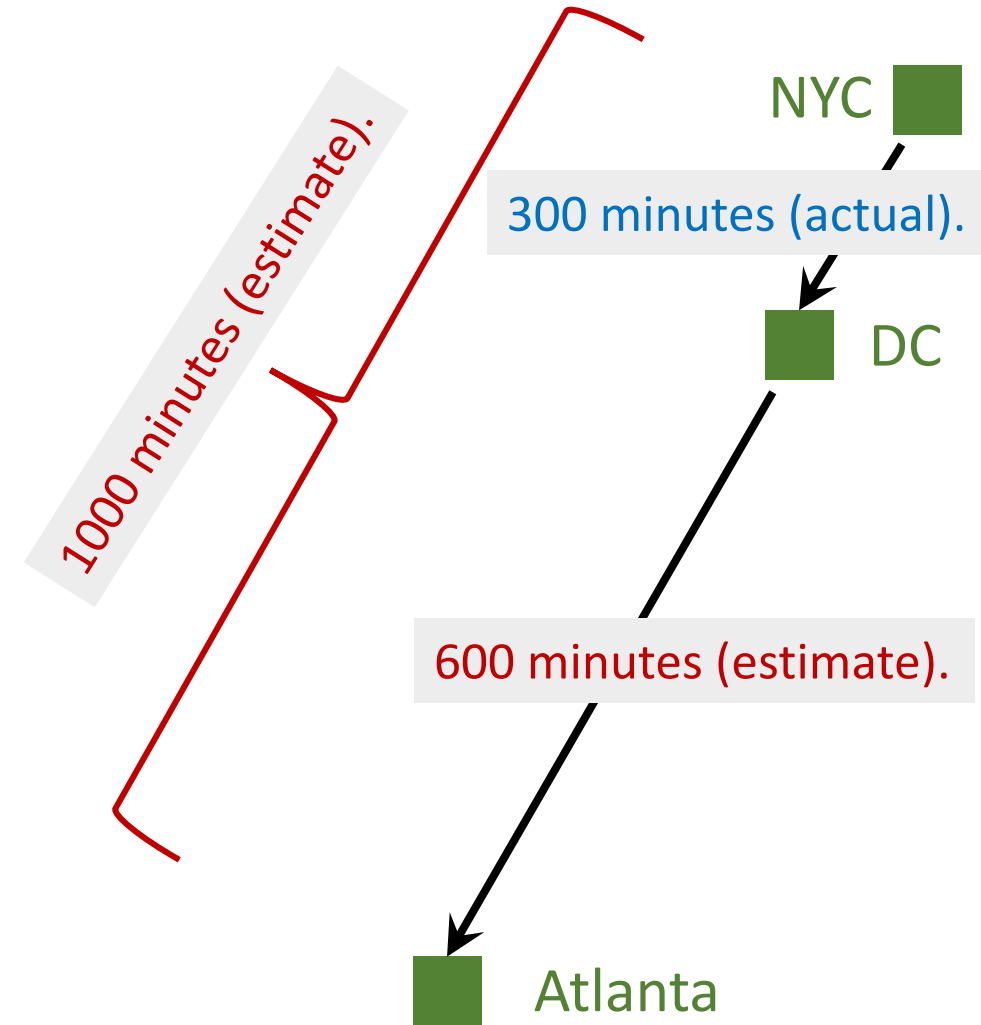
- Model's estimates:
  - NYC to Atlanta: 1000 minutes.
  - DC to Atlanta: 600 minutes.
  - → NYC to DC: 400 minutes.





# Why does TD learning work?

- Model's estimates:
  - NYC to Atlanta: 1000 minutes.
  - DC to Atlanta: 600 minutes.
  - → NYC to DC: 400 minutes.
- Ground truth:
  - NYC to DC: 300 minutes.
- TD error:  $\delta = 400 - 300 = 100$



# TD Learning for DQN

# How to apply TD learning to DQN?

- In the “driving time” example, we have the equation:

$$T_{\text{NYC} \rightarrow \text{ATL}} \approx T_{\text{NYC} \rightarrow \text{DC}} + T_{\text{DC} \rightarrow \text{ATL}} .$$

Model's estimate

Actual time

Model's estimate

# How to apply TD learning to DQN?

- In the “driving time” example, we have the equation:

$$T_{\text{NYC} \rightarrow \text{ATL}} \approx T_{\text{NYC} \rightarrow \text{DC}} + T_{\text{DC} \rightarrow \text{ATL}} .$$

The diagram illustrates the equation above with three boxes below it: "Model's estimate" (red text), "Actual time" (blue text), and "Model's estimate" (red text). A curved arrow points from the first box to the red term  $T_{\text{NYC} \rightarrow \text{ATL}}$ . A straight arrow points from the second box to the blue term  $T_{\text{NYC} \rightarrow \text{DC}}$ . A curved arrow points from the third box to the red term  $T_{\text{DC} \rightarrow \text{ATL}}$ .

- In deep reinforcement learning:


$$Q(s_t, a_t; \mathbf{w}) \approx r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}).$$

# How to apply TD learning to DQN?

Definition of discounted return:

$$\begin{aligned} \bullet U_t &= R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^4 \cdot R_{t+4} + \dots \\ &\quad \underbrace{\hspace{15em}} \\ &= \gamma \cdot (R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+4} + \dots) \end{aligned}$$


# How to apply TD learning to DQN?

- $$U_t = R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^4 \cdot R_{t+4} + \dots$$
$$= R_t + \gamma \cdot (R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+4} + \dots)$$

$$= U_{t+1}$$

# How to apply TD learning to DQN?

**Identity:**  $U_t = R_t + \gamma \cdot U_{t+1}$ .

- $$\begin{aligned} U_t &= R_t + \gamma \cdot R_{t+1} + \gamma^2 \cdot R_{t+2} + \gamma^3 \cdot R_{t+3} + \gamma^4 \cdot R_{t+4} + \dots \\ &= R_t + \gamma \cdot (R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \gamma^3 \cdot R_{t+4} + \dots) \end{aligned}$$

  
 $= U_{t+1}$

# How to apply TD learning to DQN?

Identity:  $U_t = R_t + \gamma \cdot U_{t+1}$ .

TD learning for DQN:

- DQN's output,  $Q(s_t, a_t; \mathbf{w})$ , is an estimate of  $U_t$ .
- DQN's output,  $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ , is an estimate of  $U_{t+1}$ .



# How to apply TD learning to DQN?

Identity:  $U_t = R_t + \gamma \cdot U_{t+1}$ .

## TD learning for DQN:

- DQN's output,  $Q(s_t, a_t; \mathbf{w})$ , is an estimate of  $U_t$ .
- DQN's output,  $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ , is an estimate of  $U_{t+1}$ .

• Thus, 
$$\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{estimate of } U_t} \approx \mathbb{E}[R_t] + \gamma \cdot \underbrace{Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\text{estimate of } U_{t+1}}.$$

# How to apply TD learning to DQN?

Identity:  $U_t = R_t + \gamma \cdot U_{t+1}$ .

## TD learning for DQN:

- DQN's output,  $Q(s_t, a_t; \mathbf{w})$ , is an estimate of  $U_t$ .
- DQN's output,  $Q(s_{t+1}, a_{t+1}; \mathbf{w})$ , is an estimate of  $U_{t+1}$ .

• Thus, 
$$\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Prediction}} \approx \underbrace{r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\text{TD target}}.$$

# Train DQN using TD learning

- Prediction:  $Q(s_t, a_t; \mathbf{w}_t)$ .
- TD target:

$$y_t = r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}_t)$$

# Train DQN using TD learning

- Prediction:  $Q(s_t, a_t; \mathbf{w}_t)$ .
- TD target:

$$\begin{aligned} y_t &= r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}_t) \\ &= r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t). \end{aligned}$$

# Train DQN using TD learning

- Prediction:  $Q(s_t, a_t; \mathbf{w}_t)$ .

- TD target:

$$\begin{aligned} y_t &= r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w}_t) \\ &= r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t). \end{aligned}$$

- Loss:  $L_t = \frac{1}{2} [Q(s_t, a_t; \mathbf{w}) - y_t]^2$ .

- Gradient descent:  $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L_t}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$ .

# Summary

# Value-Based Reinforcement Learning

**Definition:** Optimal action-value function.

- $Q^*(s_t, a_t) = \max_{\pi} \mathbb{E} [U_t | S_t = s_t, A_t = a_t].$

# Value-Based Reinforcement Learning

**Definition:** Optimal action-value function.

- $Q^*(s_t, a_t) = \max_{\pi} \mathbb{E} [U_t | S_t = s_t, A_t = a_t].$

**DQN:** Approximate  $Q^*(s, a)$  using a neural network (DQN).

- $Q(s, a; \mathbf{w})$  is a neural network parameterized by  $\mathbf{w}$ .
- Input: observed state  $s$ .
- Output: scores for every action  $a \in \mathcal{A}$ .



# Temporal Difference (TD) Learning

**Algorithm:** One iteration of TD learning.

1. Observe state  $S_t = s_t$  and perform action  $A_t = a_t$ .
2. Predict the value:  $q_t = Q(s_t, a_t; \mathbf{w}_t)$ .
3. Differentiate the value network:  $\mathbf{d}_t = \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$ .

# Temporal Difference (TD) Learning

**Algorithm:** One iteration of TD learning.

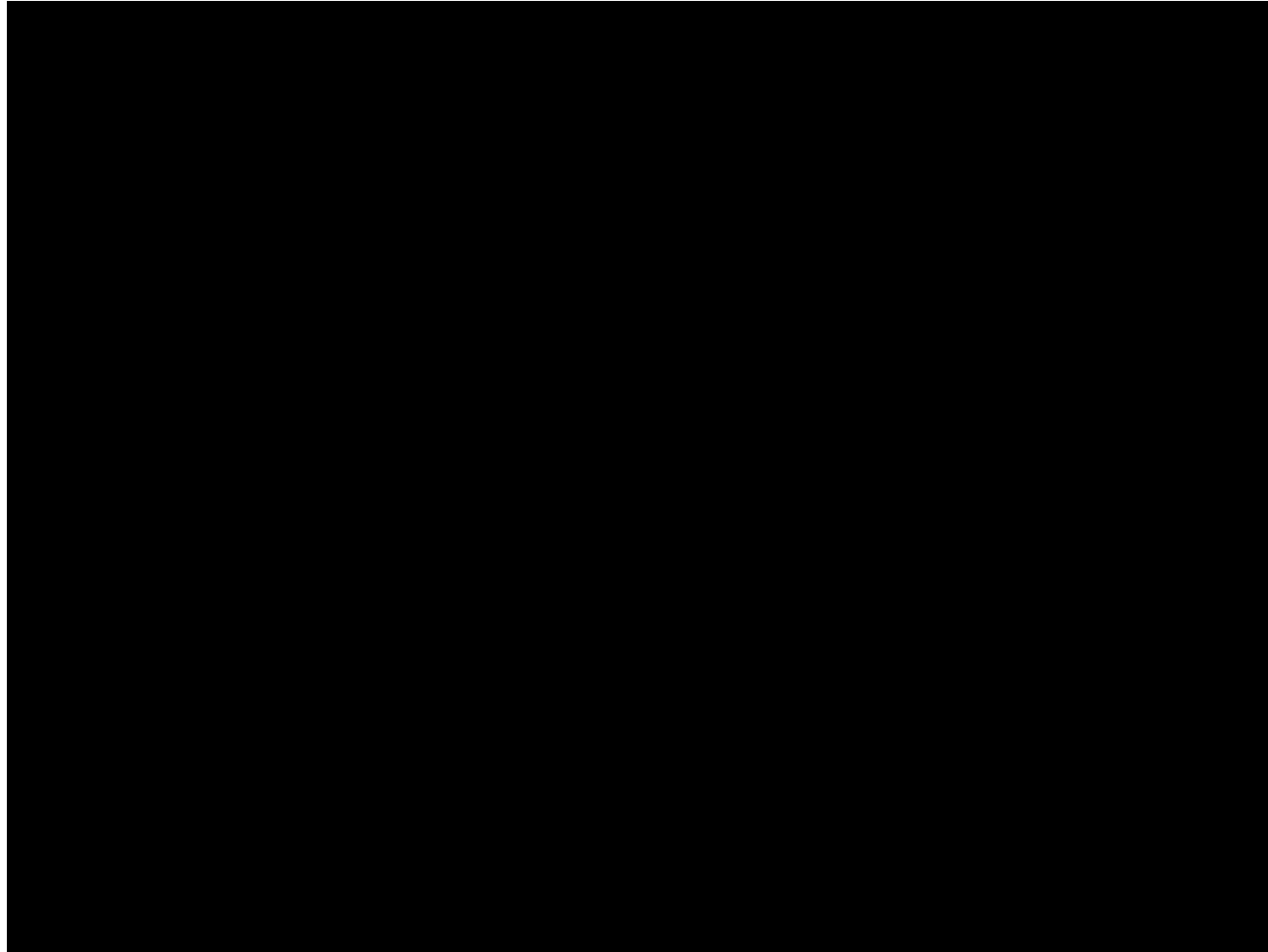
1. Observe state  $S_t = s_t$  and perform action  $A_t = a_t$ .
2. Predict the value:  $q_t = Q(s_t, a_t; \mathbf{w}_t)$ .
3. Differentiate the value network:  $\mathbf{d}_t = \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$ .
4. Environment provides new state  $s_{t+1}$  and reward  $r_t$ .
5. Compute TD target:  $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$ .

# Temporal Difference (TD) Learning

**Algorithm:** One iteration of TD learning.

1. Observe state  $S_t = s_t$  and perform action  $A_t = a_t$ .
2. Predict the value:  $q_t = Q(s_t, a_t; \mathbf{w}_t)$ .
3. Differentiate the value network:  $\mathbf{d}_t = \left. \frac{\partial Q(s_t, a_t; \mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_t}$ .
4. Environment provides new state  $s_{t+1}$  and reward  $r_t$ .
5. Compute TD target:  $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$ .
6. Gradient descent:  $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$ .

# Play Breakout using DQN



(The video was posted on YouTube by DeepMind)

**Thank you!**