Value-Based Reinforcement Learning

Shusen Wang

Action-Value Functions

Discounted Return

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

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

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

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

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

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

Definition: Action-value function for policy π.

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

  - Taken w.r.t. actions $A_{t+1}, A_{t+2}, A_{t+3}, \cdots$ and states $S_{t+1}, S_{t+2}, S_{t+3}, \cdots$
  - 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).

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

Definition: Action-value function for policy π.

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

Definition: Optimal action-value function.

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

Deep Q-Network (DQN)

Approximate the Q Function

Goal: Win the game ($\approx$ maximize the discounted return.)

Question: If we know $Q^{\star}(s, a)$, what is the best action?

Approximate the Q Function

Goal: Win the game ($\approx$ maximize the discounted return.)

Question: If we know $Q^{\star}(s, a)$, what is the best action?

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

$Q^{\star}$ 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 (≈ maximize the discounted return.)

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

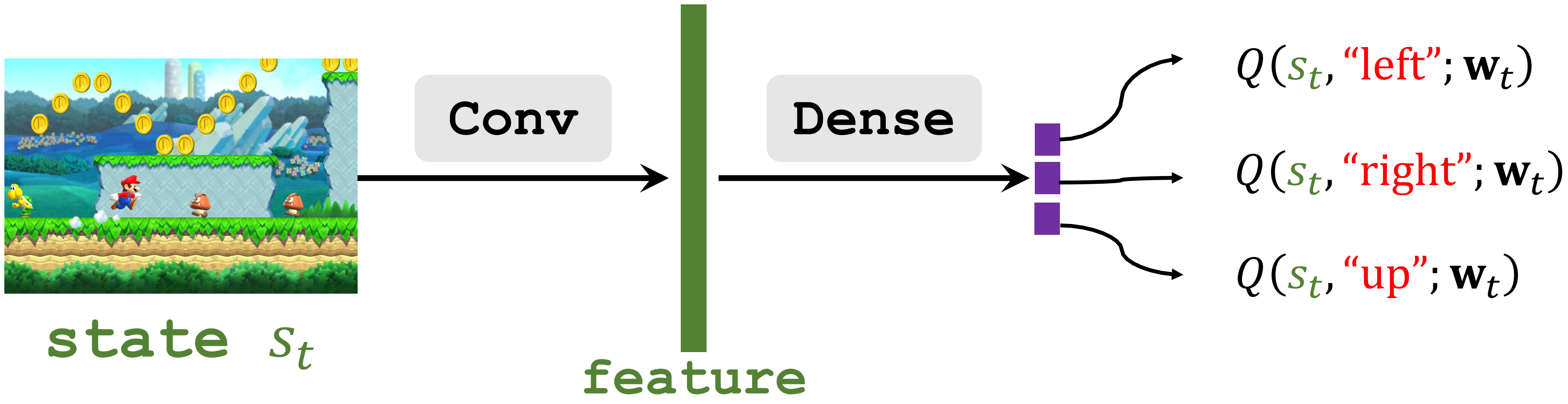
- Obviously, the best action is $a^\star = \underset{a}{\mathrm{argmax}}\, Q^\star(s, a)$.

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

- **Solution:** Use a neural network to approximate $Q^\star(s, a)$.
- Let $Q(s, a; \mathbf{w})$ be a neural network parameterized by $\mathbf{w}$.
- The input is state; the outputs are values for every a.

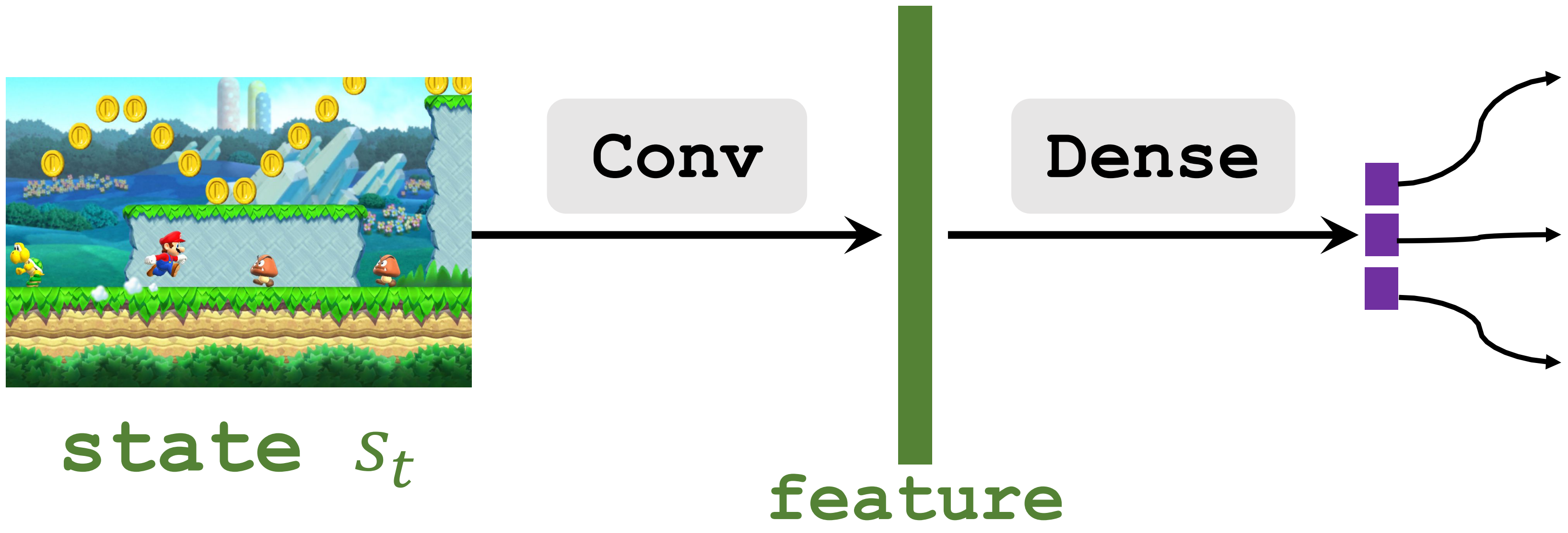
Deep Q Network (DQN)

- 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.



$$Q(s_t, \text{“left”}; \mathbf{w}_t) = 2000$$

$$Q(s_t, \text{“right”}; \mathbf{w}_t) = 1000$$

$$Q(s_t, \text{“up”}; \mathbf{w}_t) = 3000$$

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

Apply DQN to Play Game

$$s_t \longrightarrow a_t$$

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

Apply DQN to Play Game

$$s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

$$\begin{tikzcd} s_t \arrow[r] & a_t \arrow[r] \arrow[dr] & s_{t+1} \\ & & r_t \end{tikzcd}$$

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

Apply DQN to Play Game

$$s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

$$\begin{tikzcd} s_t \arrow[r] & a_t \arrow[r] \arrow[dr] & s_{t+1} \arrow[r] & a_{t+1} \\ & & r_t & \end{tikzcd}$$

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

$$a_{t+1} = \text{argmax}_a \, Q(s_{t+1}, a; \mathbf{w})$$

Apply DQN to Play Game

$$s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

$$s_{t+2} \sim p(\cdot \mid s_{t+1}, a_{t+1})$$

$$\begin{tikzcd}
s_t \arrow[r] & a_t \arrow[r] \arrow[dr] & s_{t+1} \arrow[r] & a_{t+1} \arrow[r] \arrow[dr] & s_{t+2} \\
 & & r_t & & r_{t+1}
\end{tikzcd}$$

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

$$a_{t+1} = \text{argmax}_a \, Q(s_{t+1}, a; \mathbf{w})$$

Apply DQN to Play Game

$$s_{t+1} \sim p(\cdot \mid s_t, a_t) \qquad s_{t+2} \sim p(\cdot \mid s_{t+1}, a_{t+1})$$

$$s_t \rightarrow a_t \rightarrow s_{t+1} \rightarrow a_{t+1} \rightarrow s_{t+2} \rightarrow a_{t+2}$$

$$r_t \qquad r_{t+1}$$

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

$$a_{t+1} = \text{argmax}_a \, Q(s_{t+1}, a; \mathbf{w})$$

$$a_{t+2} = \text{argmax}_a \, Q(s_{t+2}, a; \mathbf{w})$$

Apply DQN to Play Game

$$s_{t+1} \sim p(\cdot \mid s_t, a_t) \qquad s_{t+2} \sim p(\cdot \mid s_{t+1}, a_{t+1}) \qquad s_{t+3} \sim p(\cdot \mid s_{t+2}, a_{t+2})$$

$$s_t \rightarrow a_t \rightarrow s_{t+1} \rightarrow a_{t+1} \rightarrow s_{t+2} \rightarrow a_{t+2} \rightarrow s_{t+3} \quad \cdots$$

$$r_t \qquad r_{t+1} \qquad r_{t+2}$$

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

$$a_{t+1} = \mathrm{argmax}_a \, Q(s_{t+1}, a; \mathbf{w})$$

$$a_{t+2} = \mathrm{argmax}_a \, Q(s_{t+2}, a; \mathbf{w})$$

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.

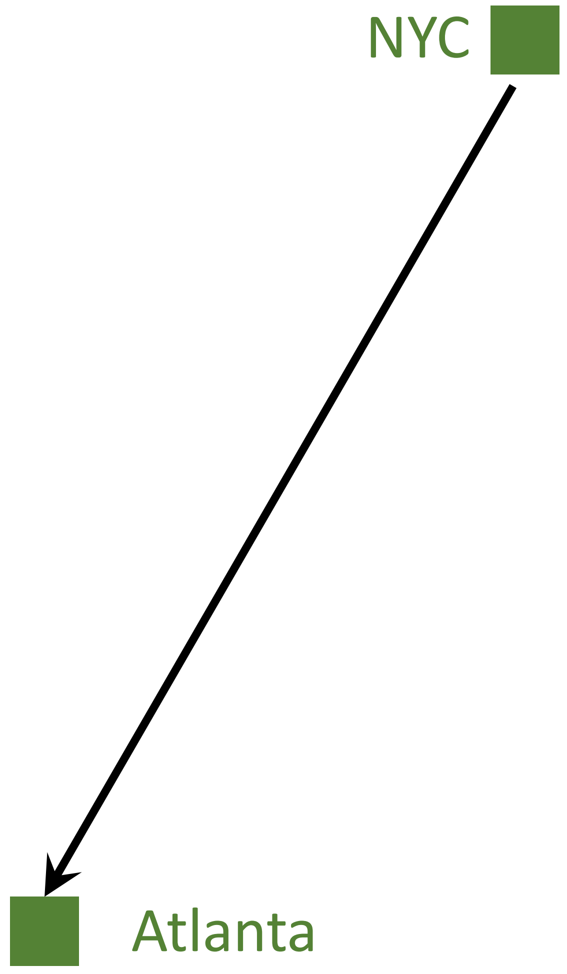
Question: How do I update the model?

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., $y = 860$.

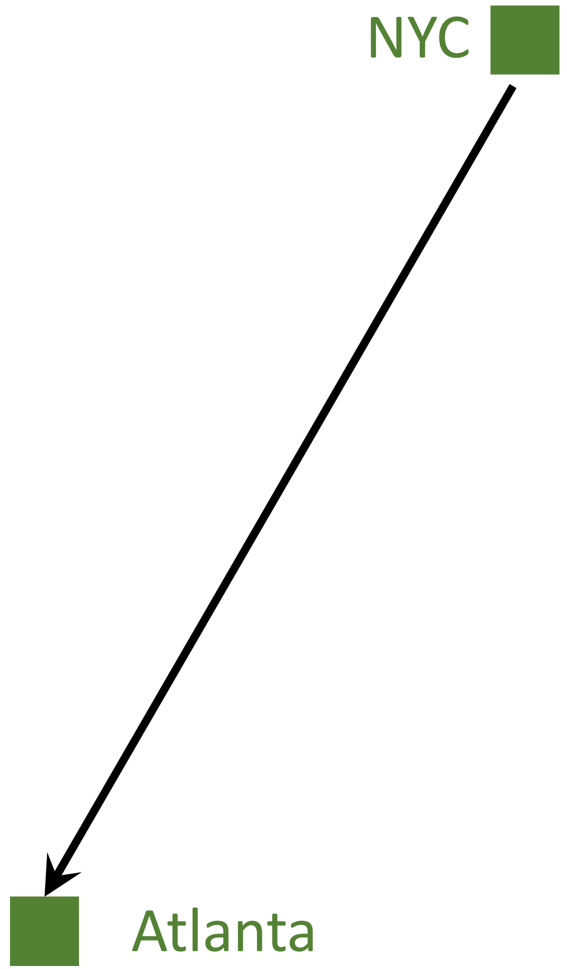


Example

- 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: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial L}{\partial \mathbf{w}} \big|_{\mathbf{w}=\mathbf{w}_t}$.

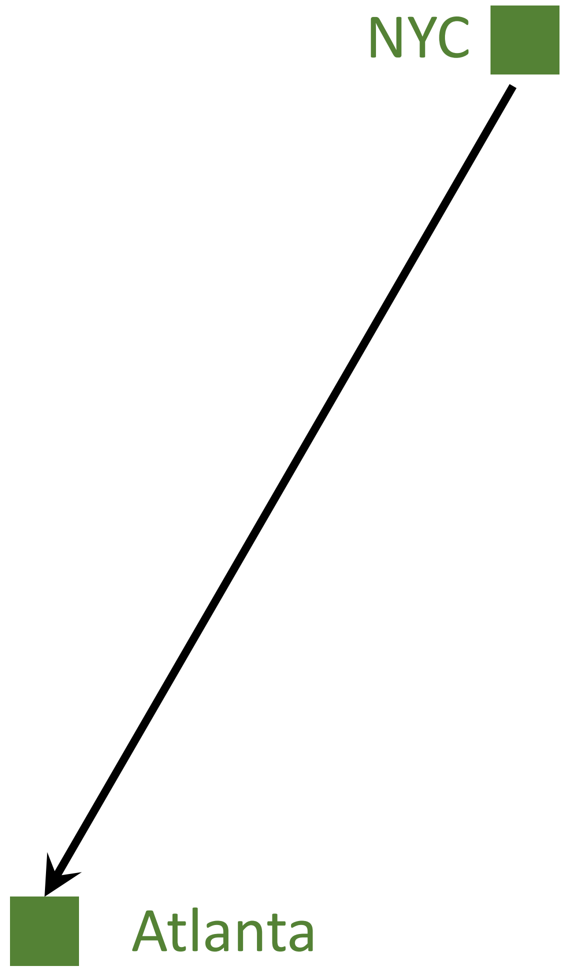


Example

- 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?

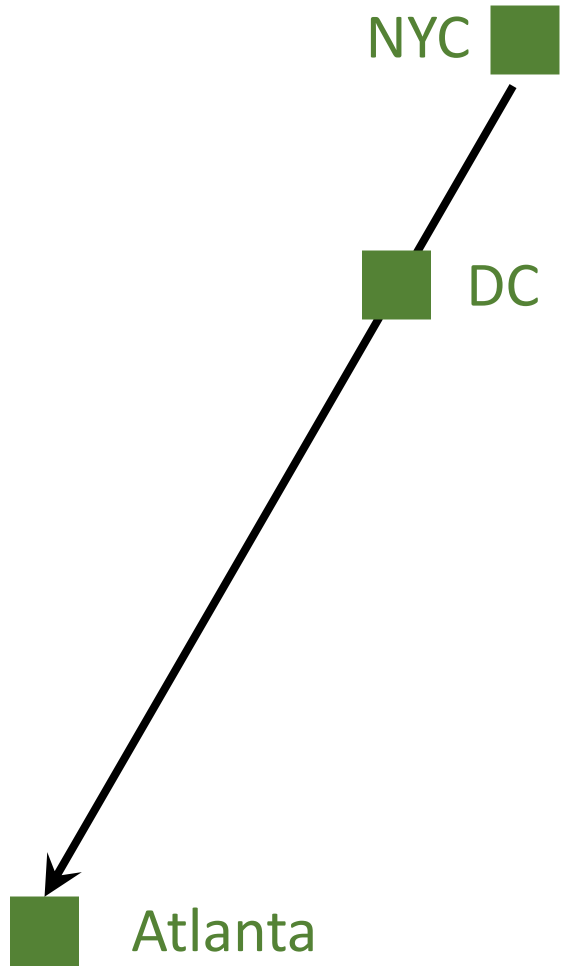


Example

- 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?

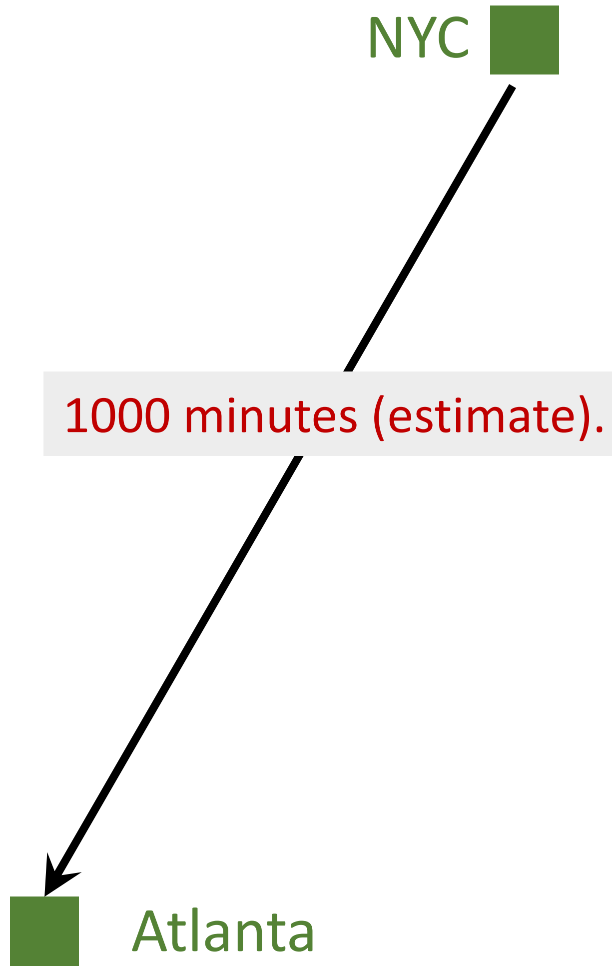
- Can I update the model before finishing the trip?
- Can I get a better $\mathbf{w}$ as soon as I arrived DC?



Temporal Difference (TD) Learning

- Model's estimate:

  NYC to Atlanta: 1000 minutes (estimate).



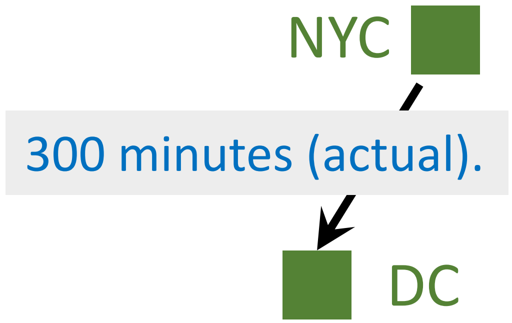
Temporal Difference (TD) Learning

- 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:

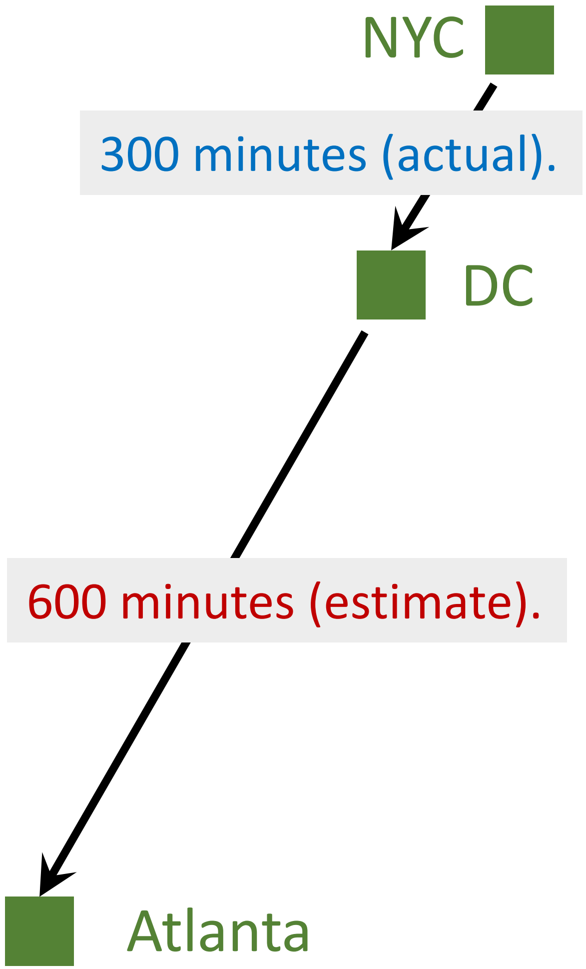
  NYC to Atlanta: 1000 minutes (estimate).

- I arrived at DC; actual time cost:

  NYC to DC: 300 minutes (actual).

- Model now updates its estimate:

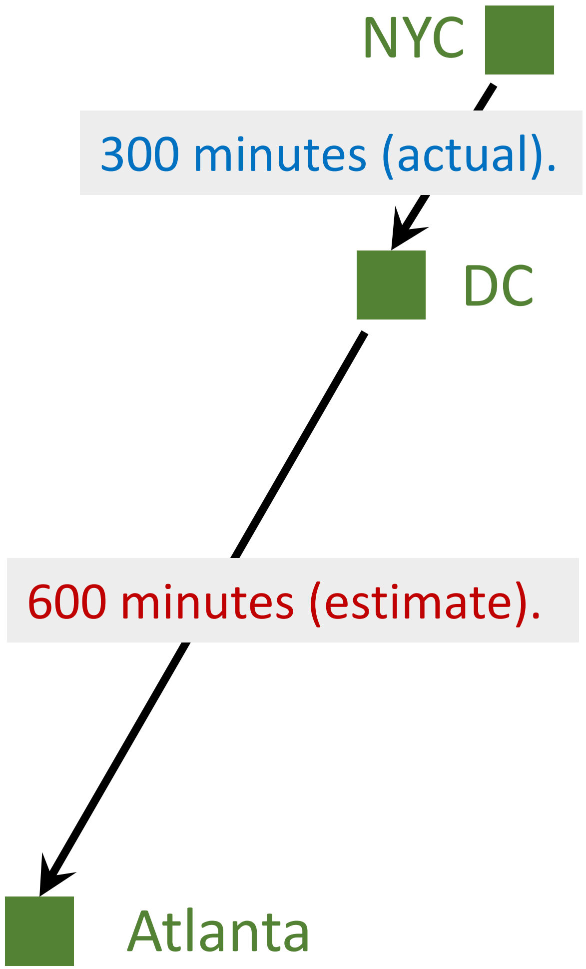
  DC to Atlanta: 600 minutes (estimate).



Temporal Difference (TD) Learning

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

TD target.

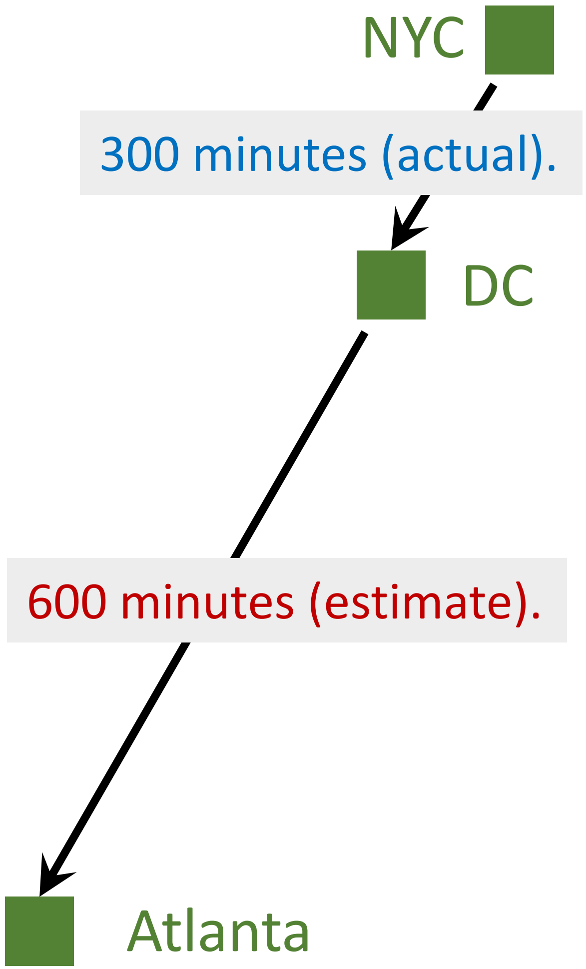


Temporal Difference (TD) Learning

- 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.

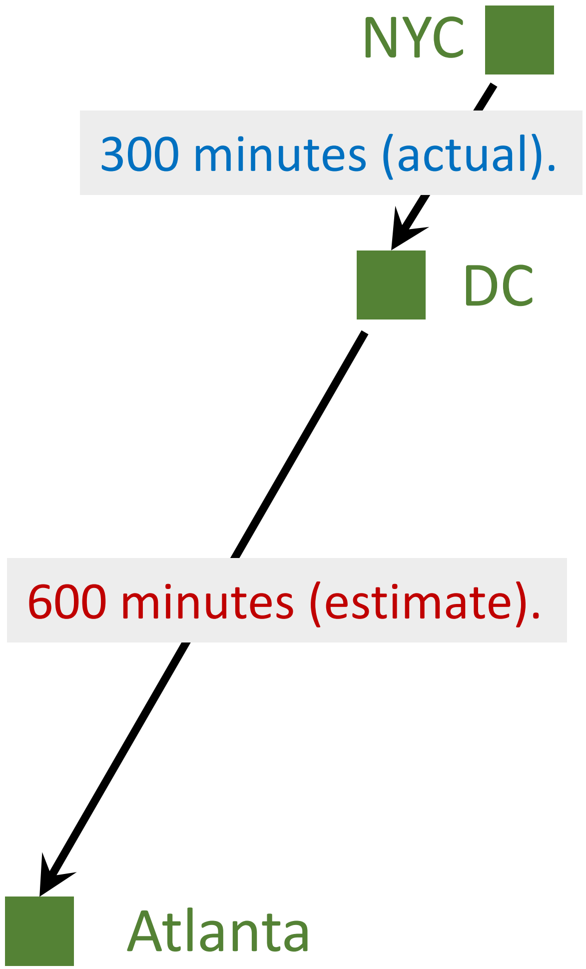


Temporal Difference (TD) Learning

- 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: $L = \frac{1}{2}\underbrace{(Q(\mathbf{w}) - y)}_{\text{TD error}}{}^2$.

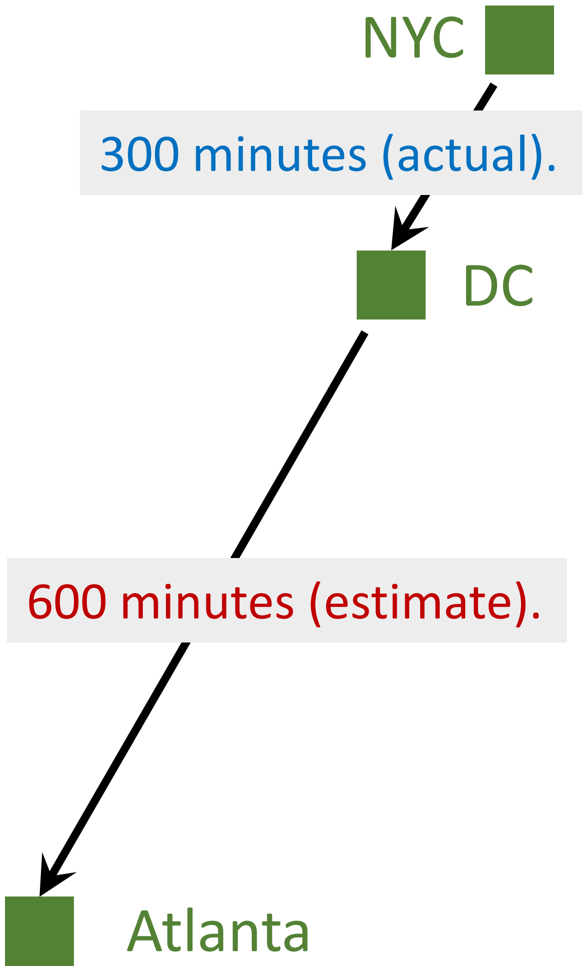


Temporal Difference (TD) Learning

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

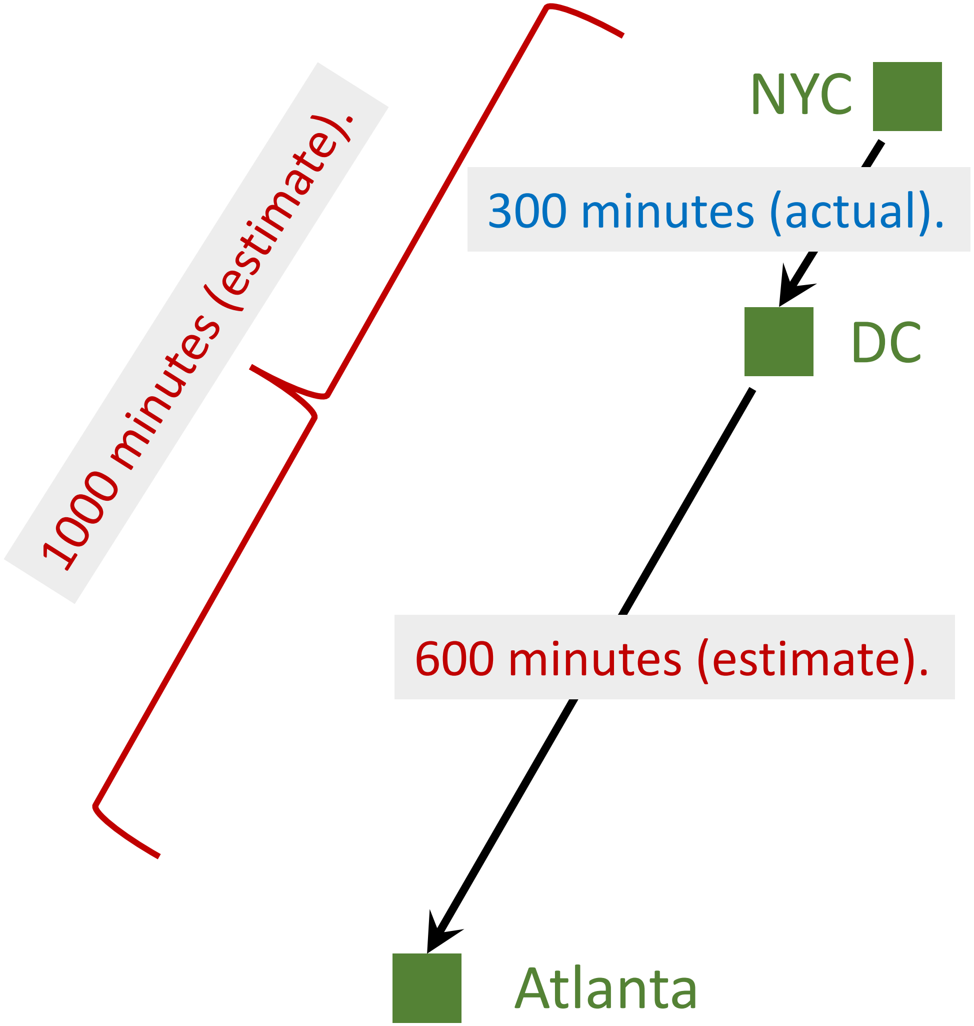
TD target.

- 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?

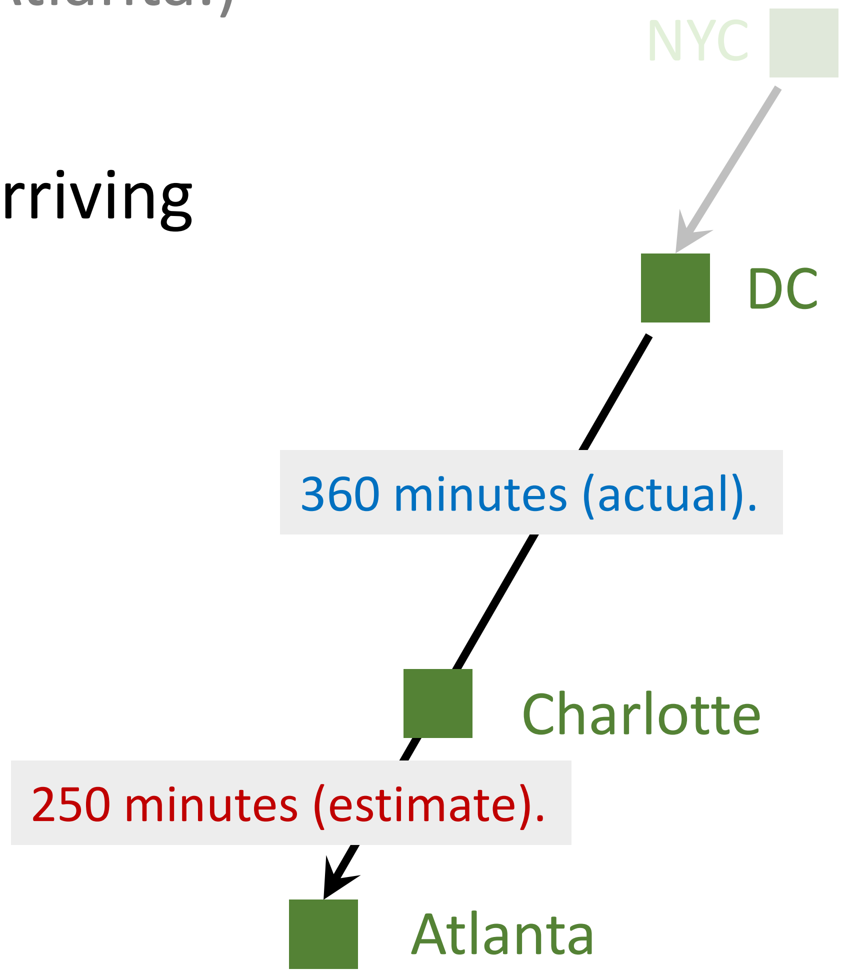
- Model's estimates:
  - NYC to Atlanta: 1000 minutes.
  - DC to Atlanta: 600 minutes.
  - ➔ NYC to DC: 400 minutes.
- Ground truth:
  - NYC to DC: 300 minutes.



Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 600$ minutes (DC to Atlanta.)

- Continue driving. Update the model again upon arriving at Charlotte.

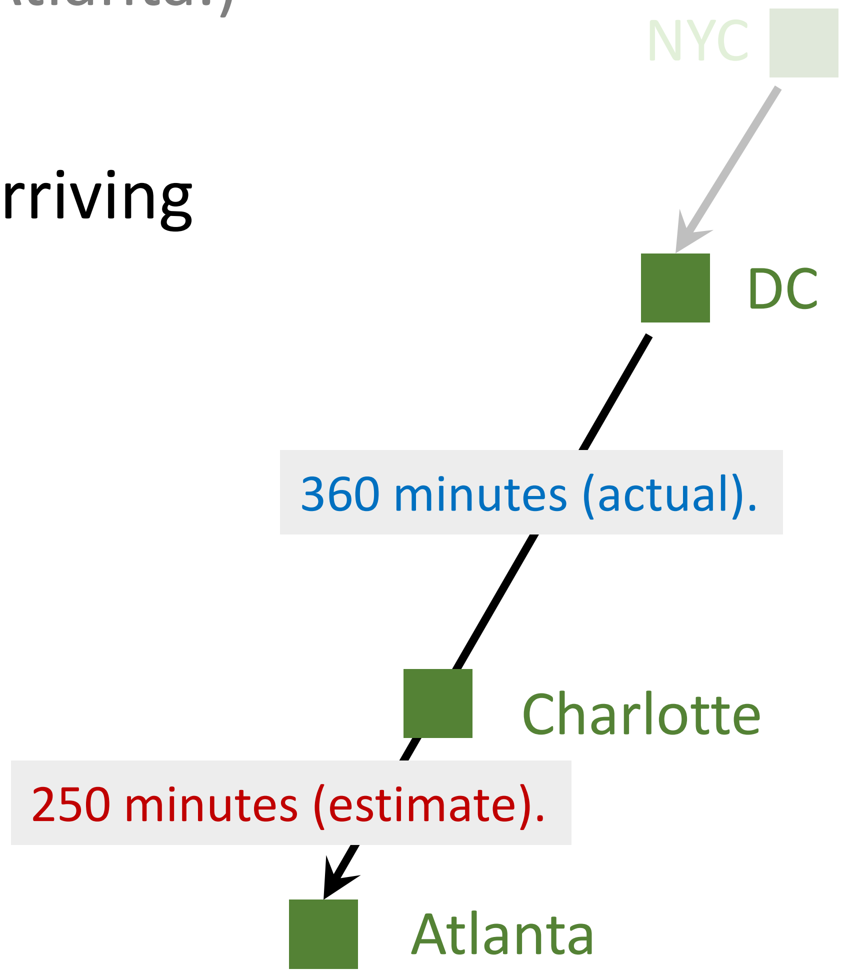


Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 600$ minutes (DC to Atlanta.)

- Continue driving. Update the model again upon arriving at Charlotte.
- I arrive at Charlotte, I have the new TD target:

$$\underbrace{360}_{\text{Actual time}} + \underbrace{250}_{\text{Updated estimate}} = \underbrace{610}_{\text{TD target.}} \text{ minutes.}$$



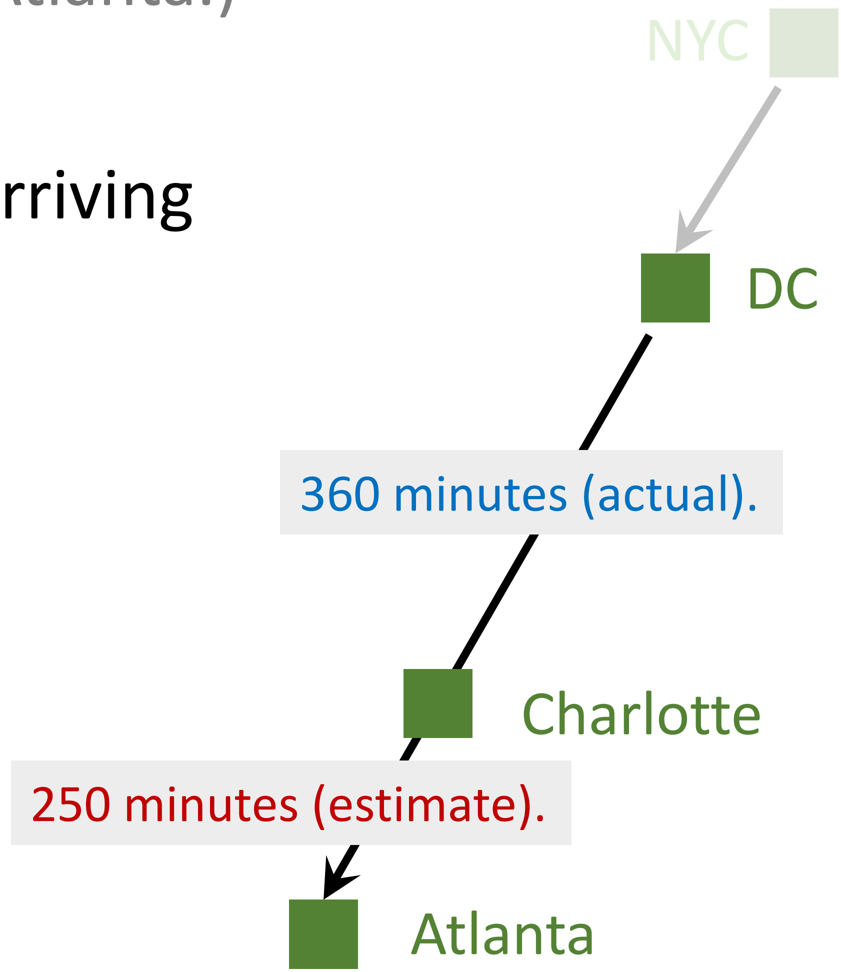
Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 600$ minutes (DC to Atlanta.)

- Continue driving. Update the model again upon arriving at Charlotte.
- I arrive at Charlotte, I have the new TD target:

$$360 + 250 = 610 \text{ minutes.}$$

- Loss: $L = \frac{1}{2}(Q(\mathbf{w}) - 610)^2$.
- Update the model again using gradient descent.



TD Learning for DQN

How to apply TD learning to DQN?

Definition of discounted return:

- $U_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + \gamma^4 \cdot r_{t+4} + \cdots$

How to apply TD learning to DQN?

- $U_t = r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + \gamma^4 \cdot r_{t+4} + \cdots$

$$= r_t + \gamma \cdot \underbrace{\left(r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \cdots\right)}_{= U_{t+1}}$$

How to apply TD learning to DQN?

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

- $$\begin{aligned} U_t &= r_t + \gamma \cdot r_{t+1} + \gamma^2 \cdot r_{t+2} + \gamma^3 \cdot r_{t+3} + \gamma^4 \cdot r_{t+4} + \cdots \\ &= r_t + \gamma \cdot \underbrace{\left(r_{t+1} + \gamma \cdot r_{t+2} + \gamma^2 \cdot r_{t+3} + \gamma^3 \cdot r_{t+4} + \cdots\right)}_{= U_{t+1}} \end{aligned}$$

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 estimate of $\mathbb{E}[U_t]$.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is estimate of $\mathbb{E}[U_{t+1}]$.

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 estimate of $\mathbb{E}[U_t]$.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is estimate of $\mathbb{E}[U_{t+1}]$.
- Thus, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\approx \mathbb{E}[U_t]} \approx \mathbb{E}[r_t + \gamma \cdot \underbrace{Q(s_{t+1}, a_{t+1}; \mathbf{w})}_{\approx \mathbb{E}[U_{t+1}]}]$.

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 estimate of $\mathbb{E}[U_t]$.
- DQN's output, $Q(s_{t+1}, a_{t+1}; \mathbf{w})$, is estimate of $\mathbb{E}[U_{t+1}]$.
- Thus, $\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Prediction}} \approx \underbrace{\mathbb{E}[r_t + \gamma \cdot Q(s_{t+1}, a_{t+1}; \mathbf{w})]}_{\text{TD target}}.$

Train DQN using TD learning

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

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

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

Train DQN using TD learning

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

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

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

- 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.

- $Q^{\star}(s_t, a_t) = \max_{\pi} \mathbb{E}\left[U_t | s_t, a_t\right].$

Value-Based Reinforcement Learning

Definition: Optimal action-value function.

- $Q^{\star}(s_t, a_t) = \max_{\pi} \mathbb{E}\left[U_t | s_t, a_t\right]$.

DQN: Approximate $Q^{\star}(s, a)$ using a neural network (DQN).

- $Q(s, a; \mathbf{w})$ is a neural network parameterized by $\mathbf{w}$.
- Input: observed state s (e.g., a screenshot of game.)
- Output: a vector, each entry of which corresponds to an action a.

Temporal Difference (TD) Learning

Algorithm: One iteration of TD learning.

1. Observe state $S_t = s_t$ and 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 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 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: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$.
6. Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

Thank you!