Value-Based Reinforcement Learning

Shusen Wang

Action-Value Functions

Discounted Return

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

- $R_t = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \gamma^3\, r_{t+3} + \cdots$ (to infinity.)

- The rewards depend on actions $a_t, a_{t+1}, a_{t+2}, a_{t+3}, \cdots$ and states $s_{t+1}, s_{t+2}, s_{t+3}, \cdots$
- Observing s_t, the future rewards are random.
- Actions are randomly sampled: $a_t \sim \pi(\cdot\,|s_t)$. (Policy function.)
- States are randomly sampled: $s_{t+1} \sim p(\cdot\,|s_t, a_t)$. (State transition.)

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

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

- $R_t = r_t + \gamma \, r_{t+1} + \gamma^2 \, r_{t+2} + \gamma^3 \, r_{t+3} + \cdots$ (to infinity.)

Definition: Action-value function for policy π.

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

  - 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 a_t and s_t.

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

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

- $R_t = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \gamma^3\, r_{t+3} + \cdots$ (to infinity.)

Definition: Action-value function for policy π.

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

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?

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

$Q^\star$ 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 ($\approx$ maximize the discounted return.)

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

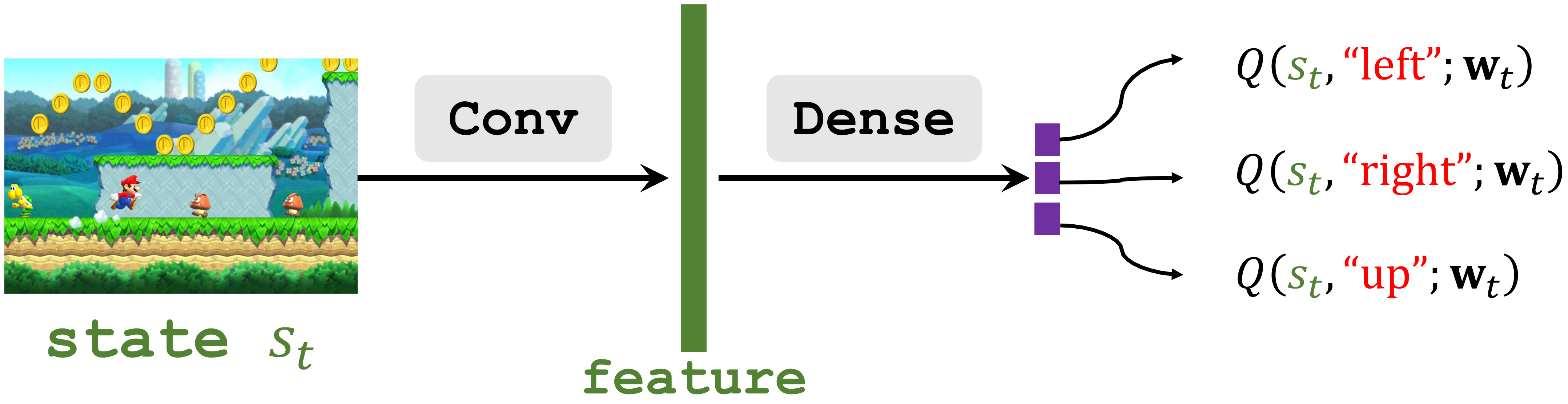
- Obviously, the best action is $a^\star = \underset{a}{\operatorname{argmax}}\, Q^\star(s, a)$.

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 output is the approximate $Q^\star$ for every a.

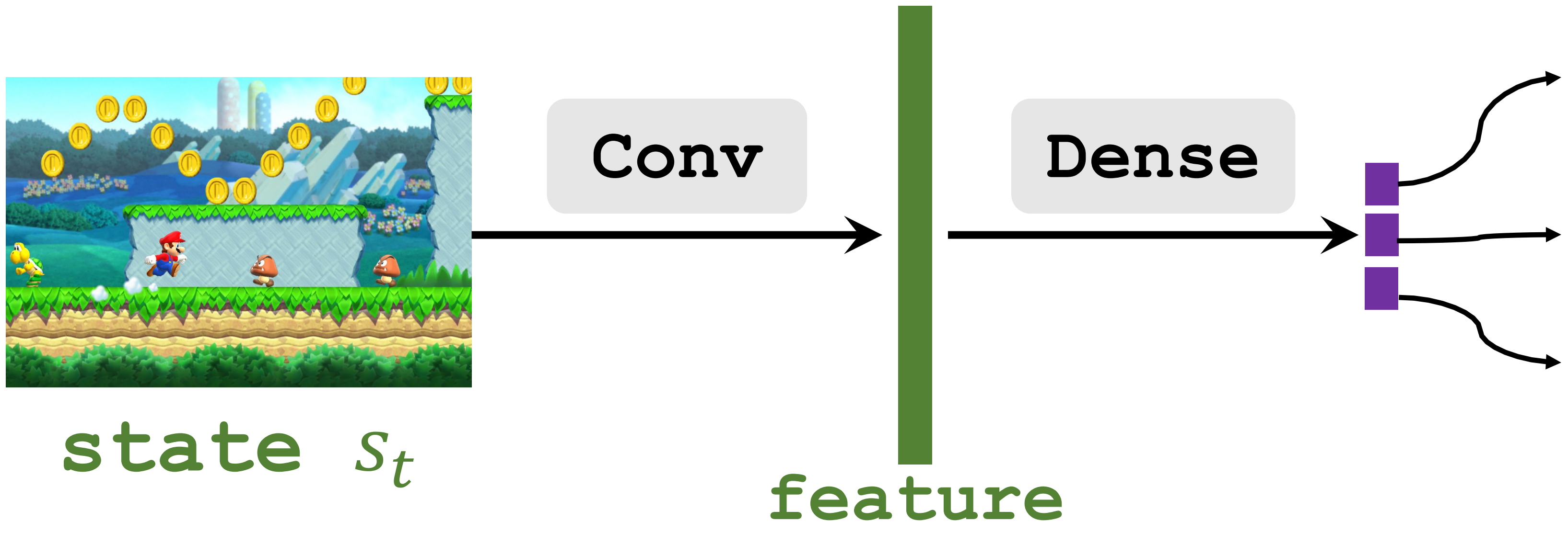
Deep Q Network (DQN)

- 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

$$s_t \longrightarrow a_t$$

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

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

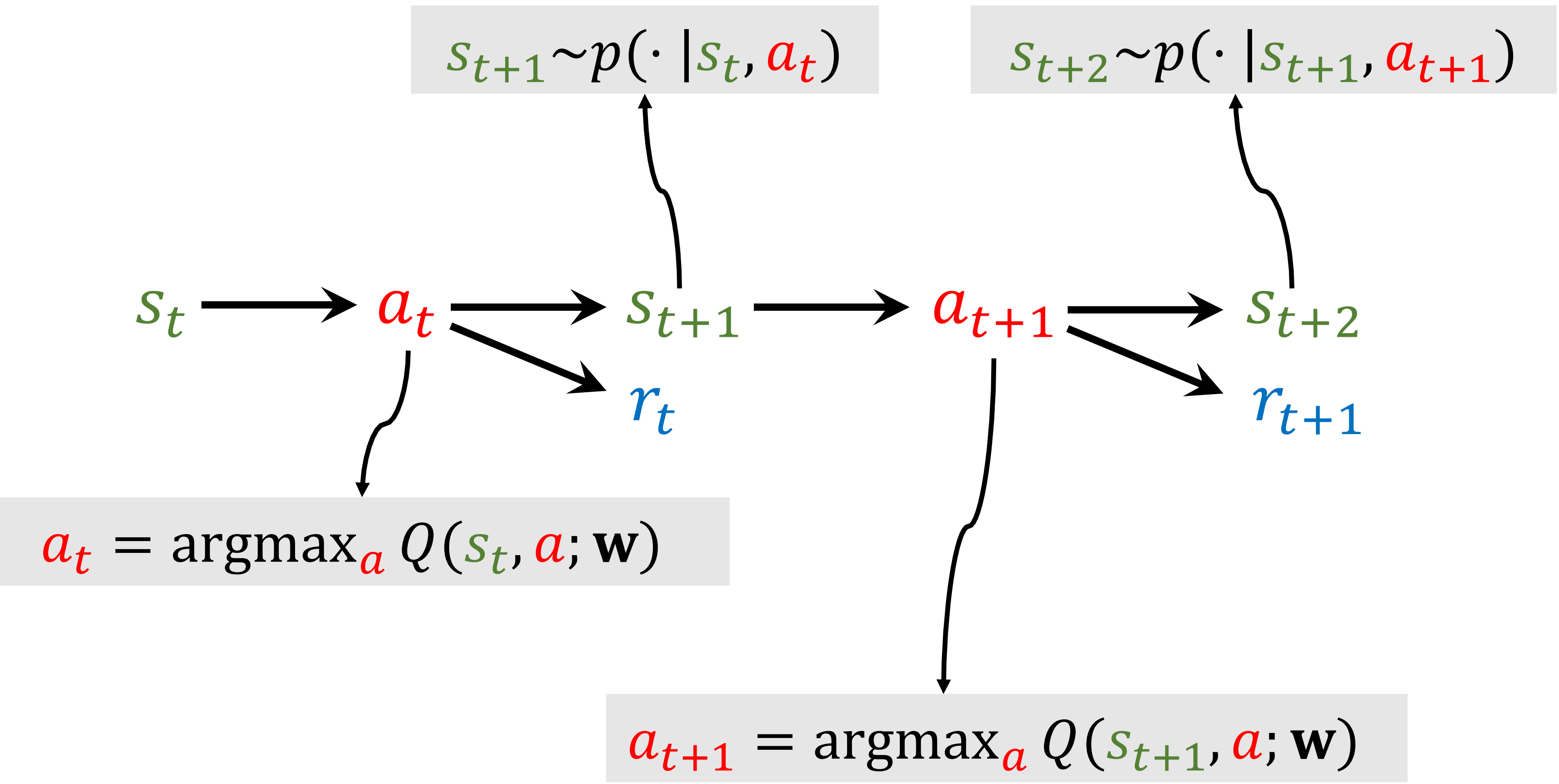
$$s_{t+1} \sim p(\cdot \mid s_t, a_t)$$

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

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

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

Apply DQN to Play Game



Apply DQN to Play Game

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

$$\begin{array}{ccccccccccc} s_t & \longrightarrow & a_t & \longrightarrow & s_{t+1} & \longrightarrow & a_{t+1} & \longrightarrow & s_{t+2} & \longrightarrow & a_{t+2} \\ & & & \searrow & r_t & & & \searrow & r_{t+1} & & \end{array}$$

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

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

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

Apply DQN to Play Game

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

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

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

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

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

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

How to train DQN?

Naïve approach:

1. Play a game to the end (totally T steps.)
2. Make predictions using DQN at every step. Get

$$Q(s_1, a_1; \mathbf{w}),\ Q(s_2, a_2; \mathbf{w}),\ \cdots,\ Q(s_T, a_T; \mathbf{w}).$$

3. Observe rewards: $r_1, r_2, \cdots, r_T$.
4. Compute returns: $R_1, R_2, \cdots, R_T$.

- $R_t = r_t + \gamma \cdot r_{t+1} + \cdots + \gamma^{T-t} \cdot r_T$.
- R_t is unknown until the game ends.

How to train DQN?

Naïve approach:

1. Play a game to the end (totally T steps.)
2. Make predictions using DQN at every step. Get
$$Q(s_1, a_1; \mathbf{w}),\ Q(s_2, a_2; \mathbf{w}),\ \cdots,\ Q(s_T, a_T; \mathbf{w})\ .$$
3. Observe rewards: $r_1, r_2, \cdots, r_T$.
4. Compute returns: $R_1, R_2, \cdots, R_T$.
5. Loss: $L(\mathbf{w}) = \sum_{t=1}^{T} [Q(s_t, a_t; \mathbf{w}) - R_t]^2$.
6. Use $\frac{\partial\, L(\mathbf{w})}{\partial\, \mathbf{w}}$ to update $\mathbf{w}$.

How to train DQN?

Naïve approach:

1. Play a game to the end (totally T steps.)
2. Make predictions using DQN at every step. Get
$$Q(s_1, a_1; \mathbf{w}),\ Q(s_2, a_2; \mathbf{w}),\ \cdots,\ Q(s_T, a_T; \mathbf{w}).$$
3. Observe rewards: $r_1, r_2, \cdots, r_T$.
4. Compute returns: $R_1, R_2, \cdots, R_T$.
5. Loss: $L(\mathbf{w}) = \sum_{t=1}^{T}[Q(s_t, a_t; \mathbf{w}) - R_t]^2$.
6. Use $\frac{\partial\, L(\mathbf{w})}{\partial\, \mathbf{w}}$ to update $\mathbf{w}$.

Problem: What if the game takes long to end or does not end at all?

Temporal Difference (TD) Learning

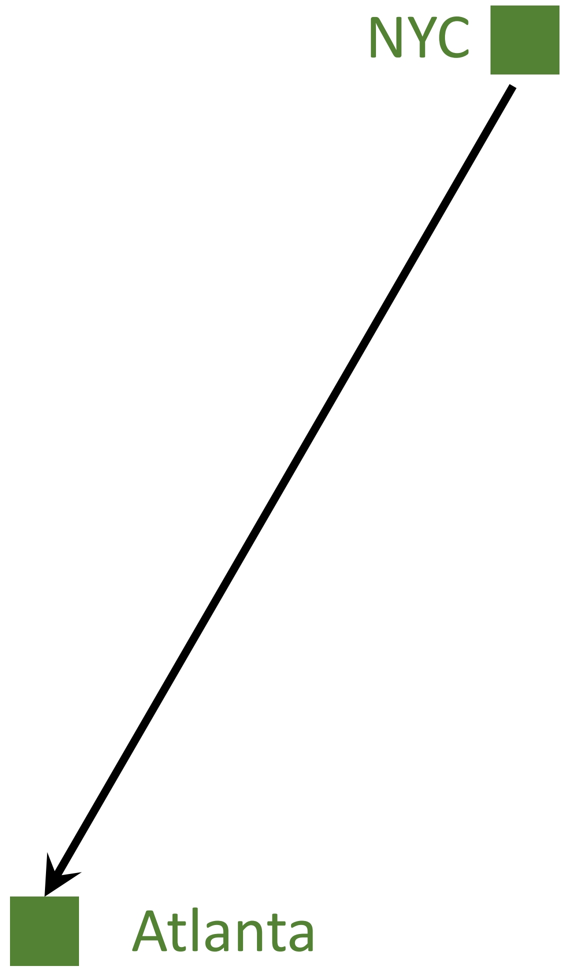
Reference

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?

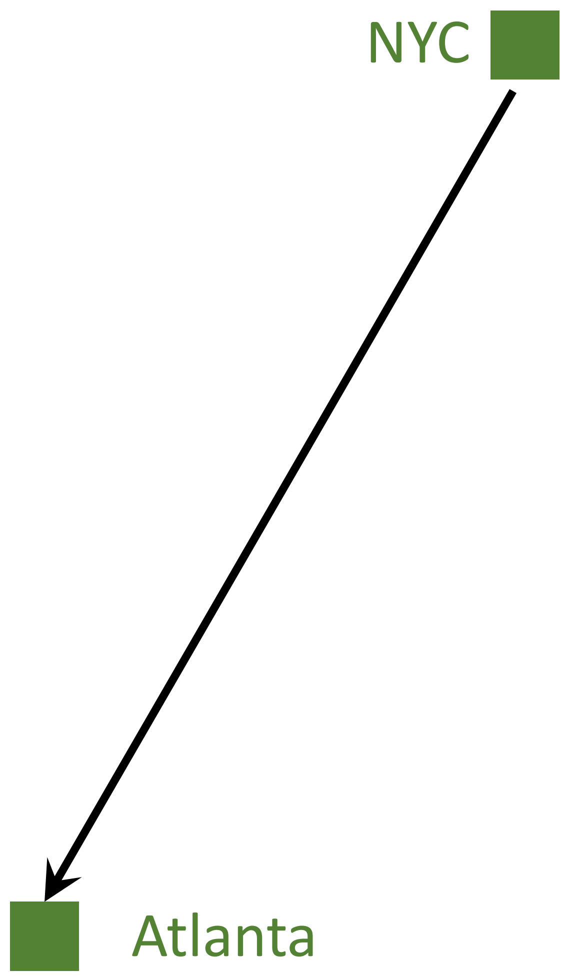


Example

- 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., $y = 860$.
- 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: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial\, L}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.

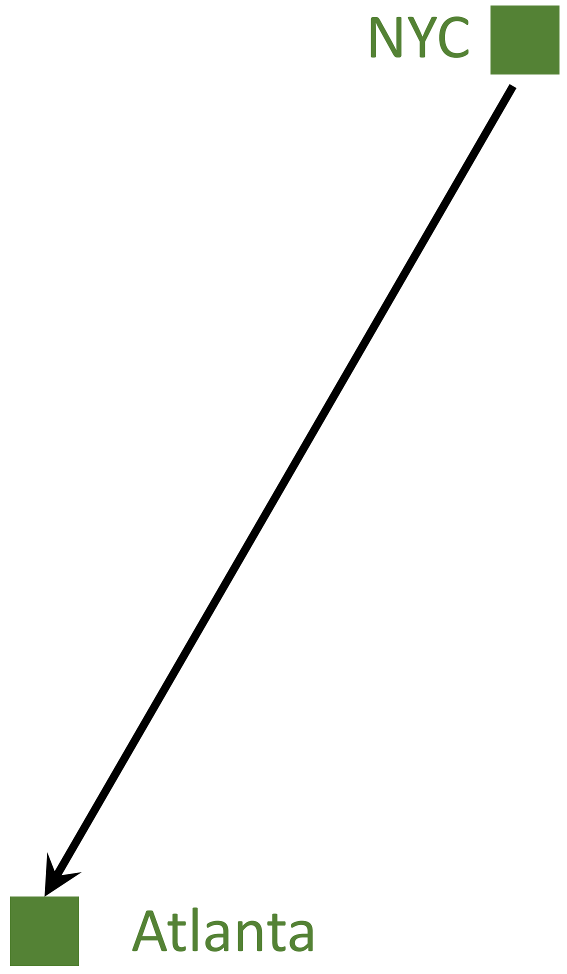


Example

- 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?

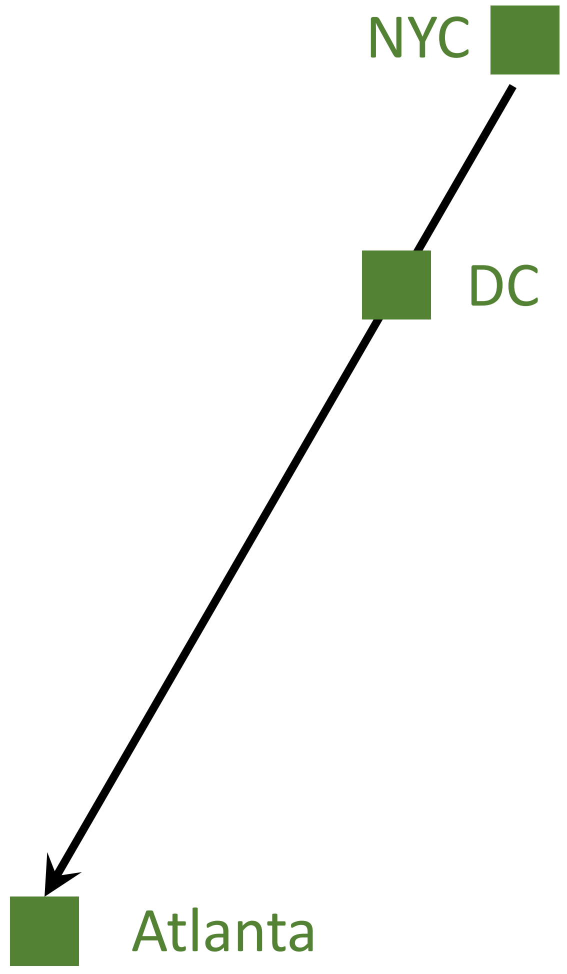


Example

- 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?

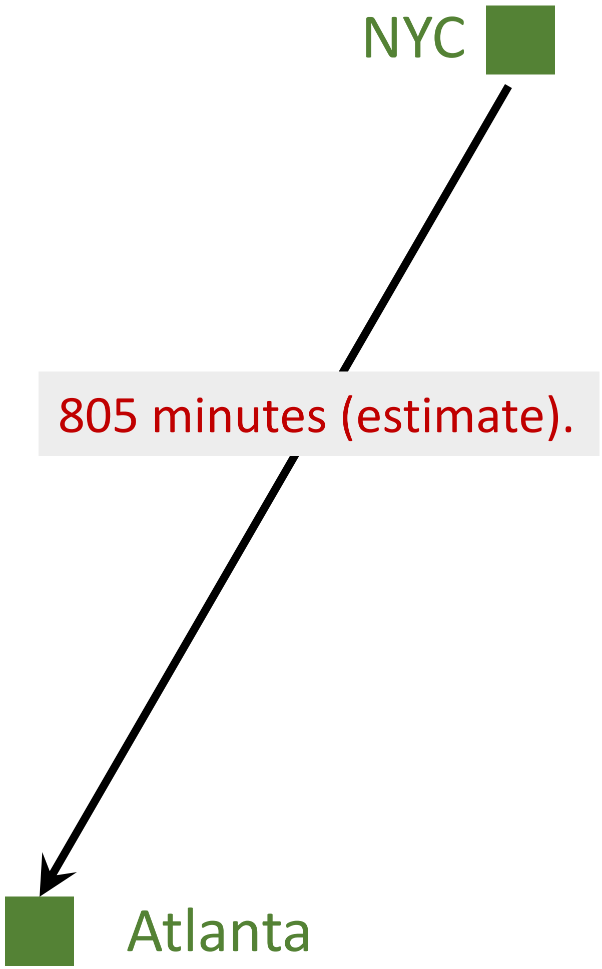
- Can I update the model before finishing the trip?
- Can I get a better $\mathbf{w}$ as soon as I arrived DC?



Temporal Difference (TD) Learning

- Model's estimate:

NYC to Atlanta: 805 minutes (estimate).



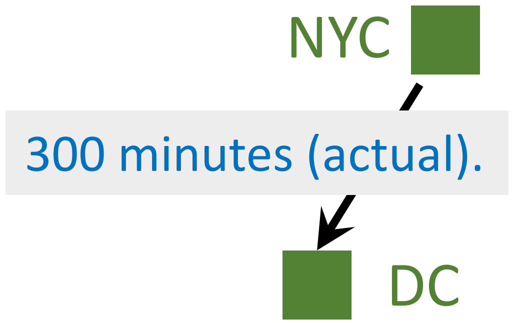
Temporal Difference (TD) Learning

- Model's estimate:

  NYC to Atlanta: 805 minutes (estimate).

- I arrived at DC; actual time cost:

  NYC to DC: 300 minutes (actual).



Temporal Difference (TD) Learning

- Model's estimate:

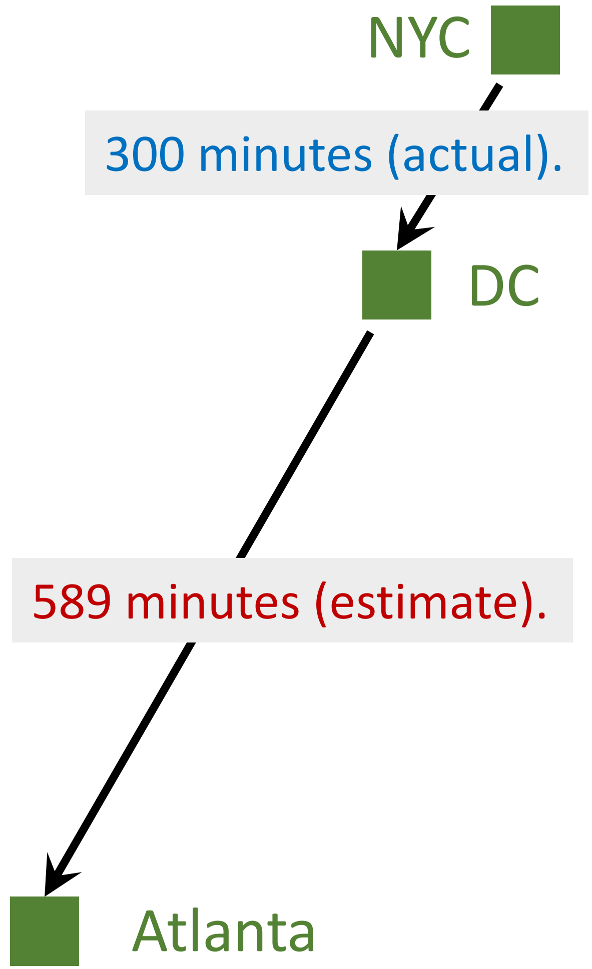
  NYC to Atlanta: 805 minutes (estimate).

- I arrived at DC; actual time cost:

  NYC to DC: 300 minutes (actual).

- Model now updates its estimate:

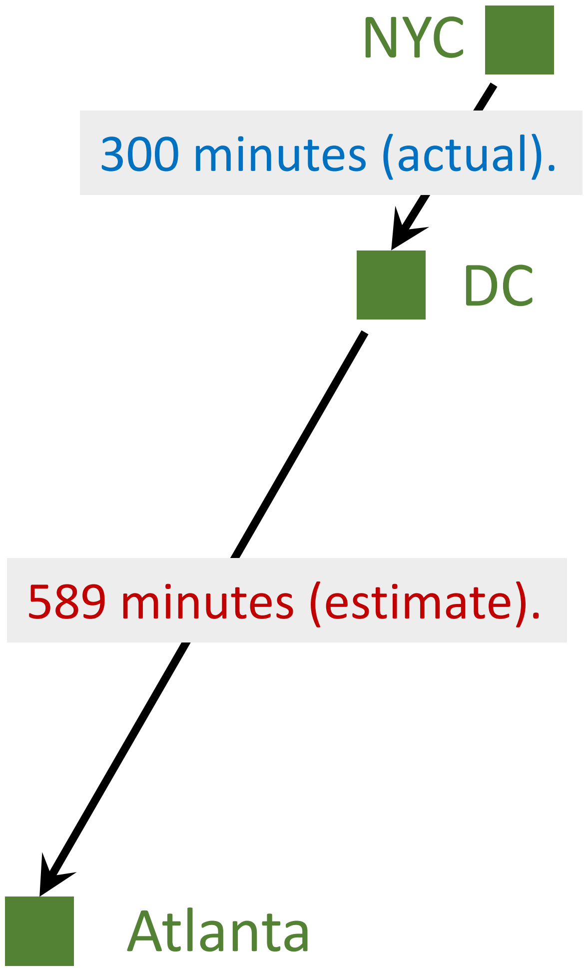
  DC to Atlanta: 589 minutes (estimate).



Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 805$ minutes .
- Updated estimate: $300 + 589 = 889$ minutes.

TD target.

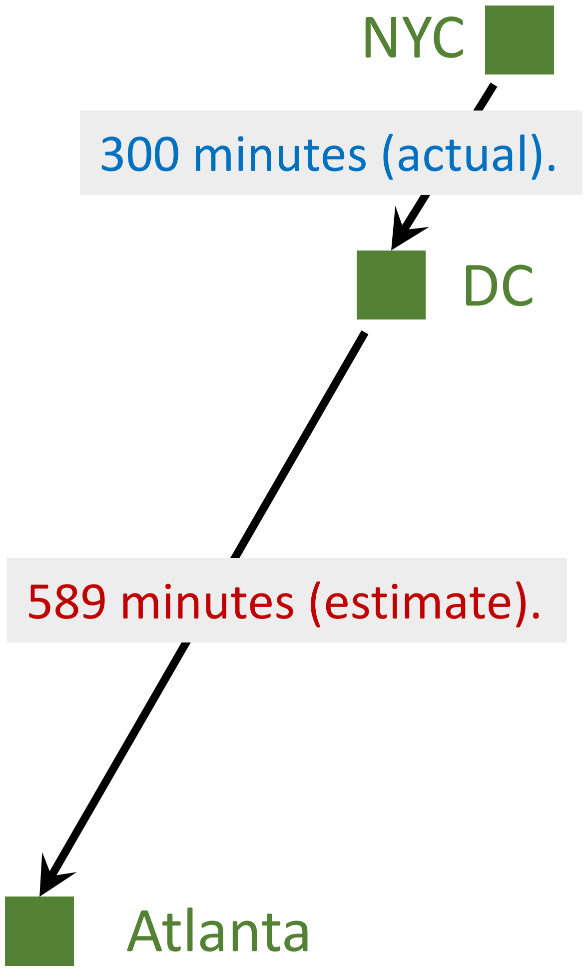


Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 805$ minutes .
- Updated estimate: $300 + 589 = 889$ minutes.

TD target.

- TD target $y = 889$ is a more reliable estimate than 805.

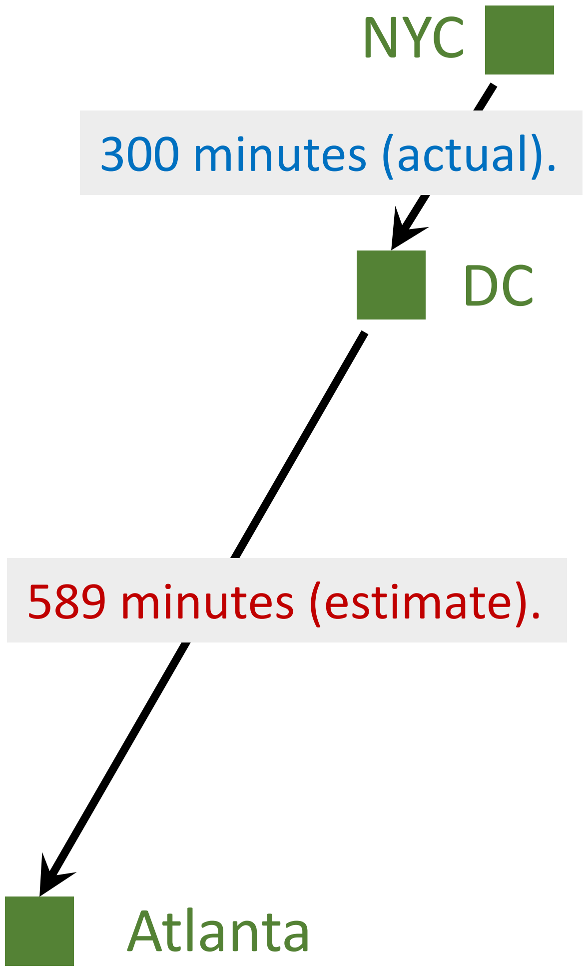


Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 805$ minutes .
- Updated estimate: $300 + 589 = 889$ minutes.

TD target.

- TD target $y = 889$ is a more reliable estimate than 805.
- Loss: $L = \frac{1}{2}\big(\underbrace{Q(\mathbf{w}) - y}_{\text{TD error}}\big)^2$.

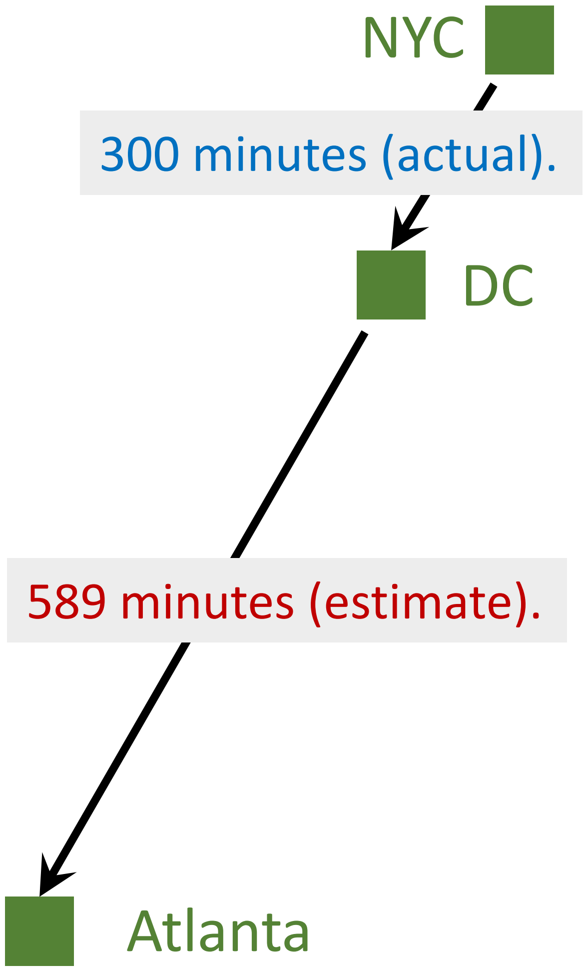


Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 805$ minutes .
- Updated estimate: $300 + 589 = 889$ minutes.

TD target.

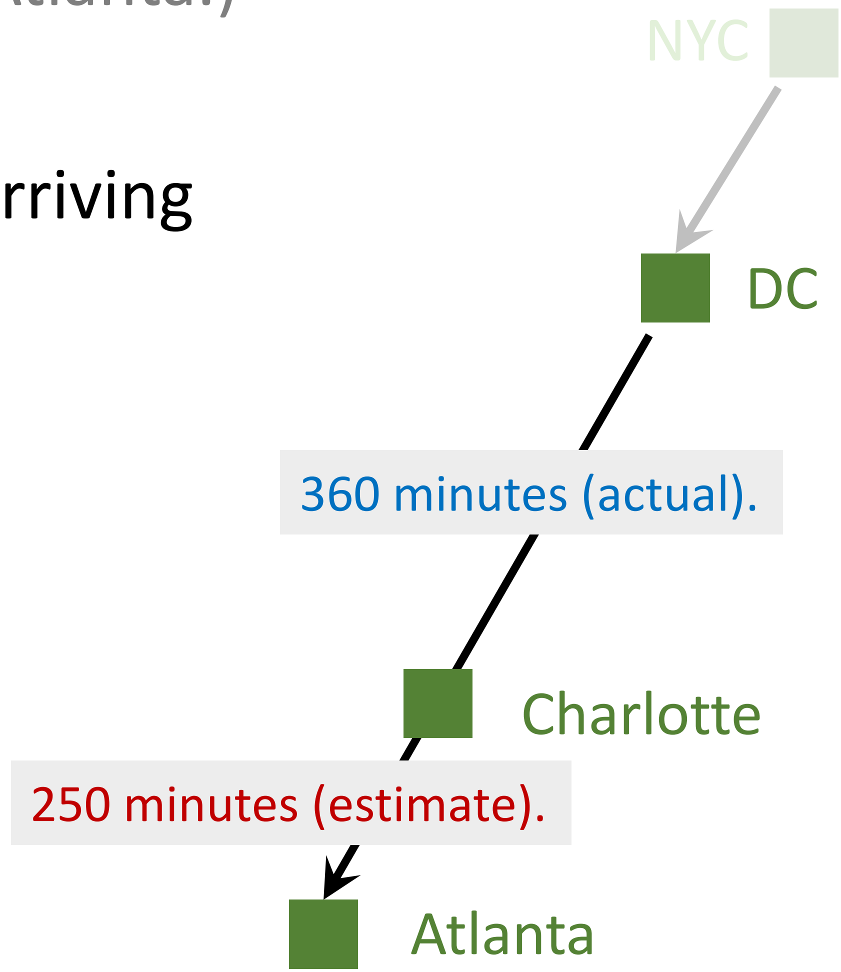
- TD target $y = 889$ is a more reliable estimate than 805.
- Loss: $L = \frac{1}{2}\left(Q(\mathbf{w}) - y\right)^2$.
- Gradient: $\frac{\partial\, L}{\partial\, \mathbf{w}} = (805 - 889) \cdot \frac{\partial\, Q(\mathbf{w})}{\partial\, \mathbf{w}}$.
- Gradient descent: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot \frac{\partial\, L}{\partial\, \mathbf{w}}\Big|_{\mathbf{w}=\mathbf{w}_t}$.



Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 589$ minutes (DC to Atlanta.)

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



Temporal Difference (TD) Learning

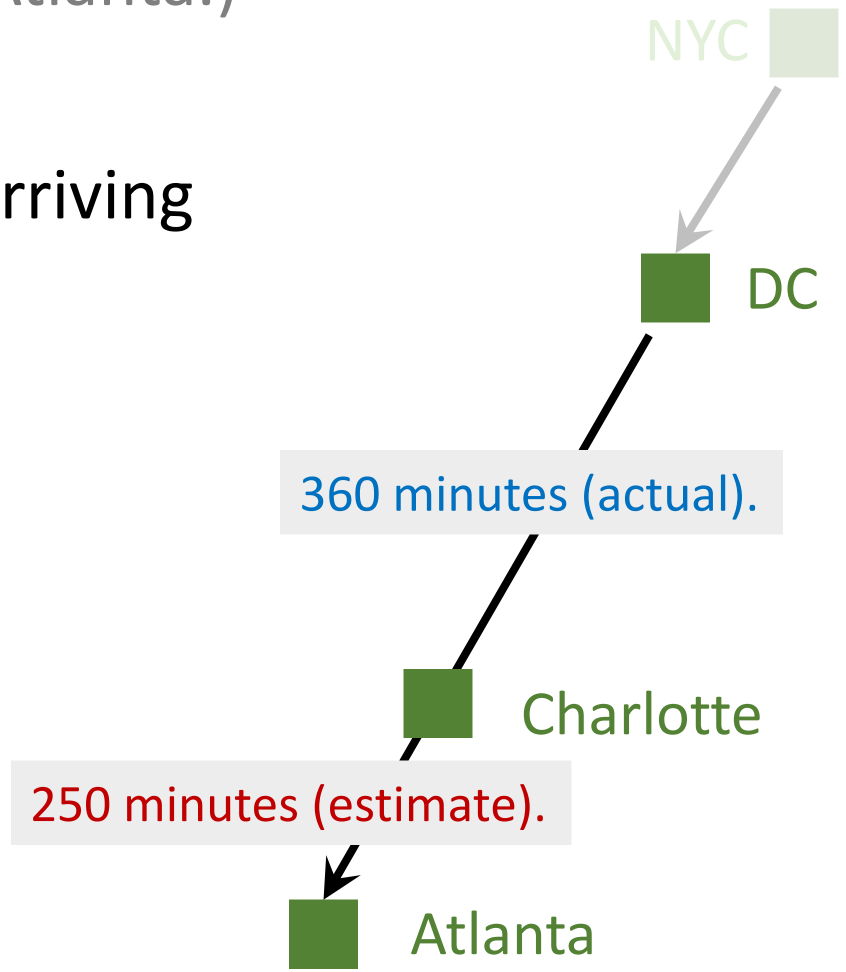
- Model's estimate: $Q(\mathbf{w}) = 589$ minutes (DC to Atlanta.)
- Continue driving. Update the model again upon arriving at Charlotte.
- I arrive at Charlotte, I have the new TD target:

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

Actual time

Updated estimate

TD target.

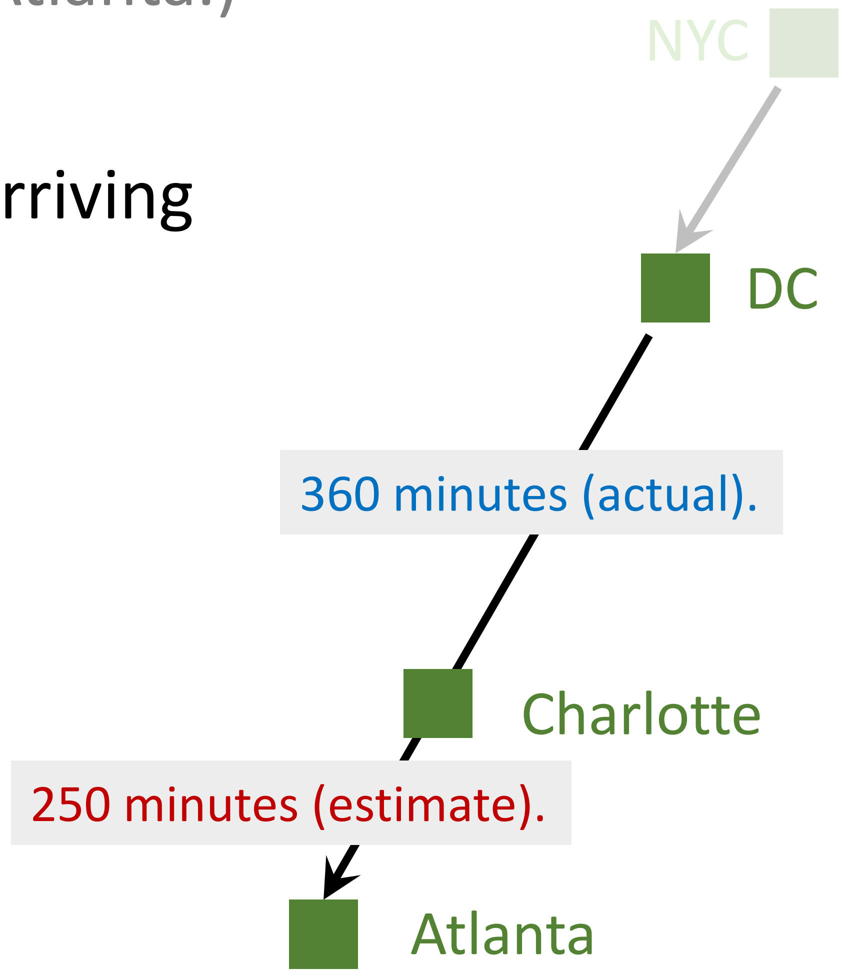


Temporal Difference (TD) Learning

- Model's estimate: $Q(\mathbf{w}) = 589$ 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

Apply DQN to Play Game

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

Apply DQN to Play Game

$s_t \longrightarrow a_t$ $\longrightarrow s_{t+1}$

r_t

$Q(s_t, a_t; \mathbf{w})$

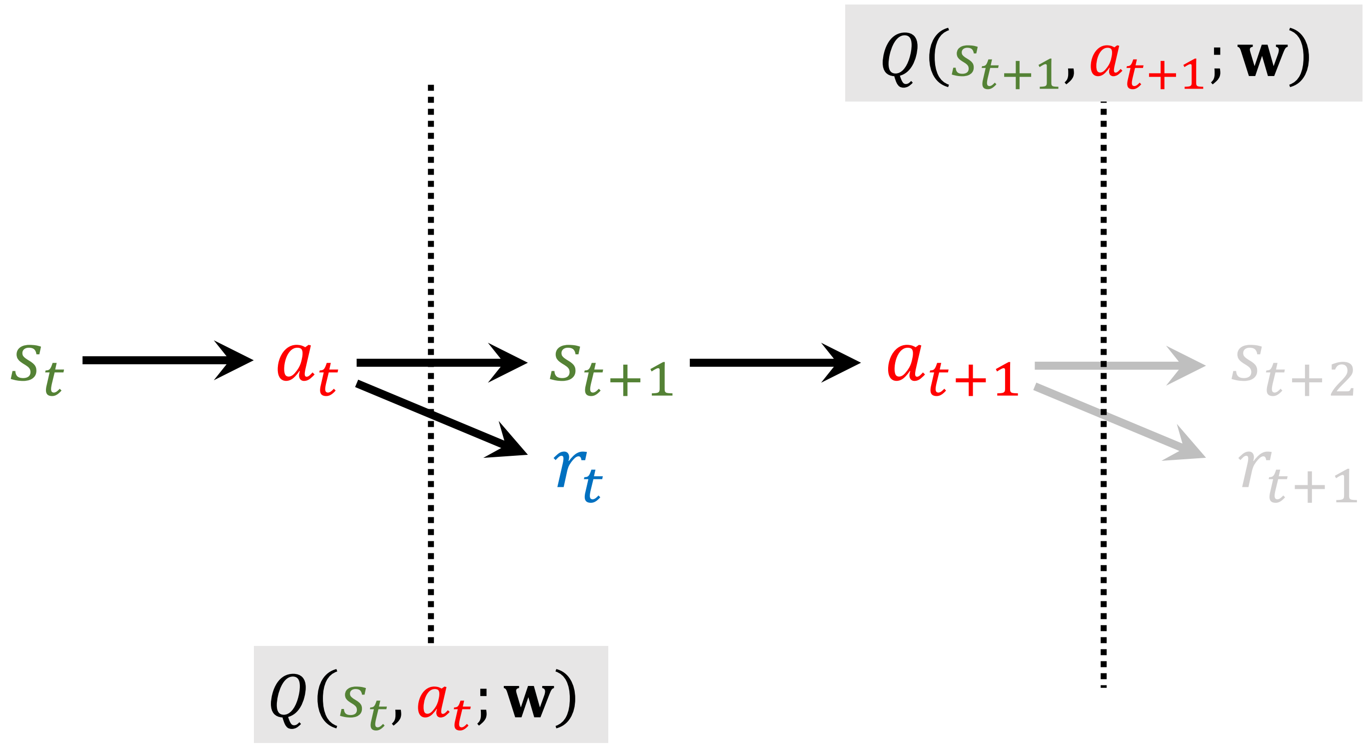
Apply DQN to Play Game

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

$Q(s_t, a_t; \mathbf{w})$

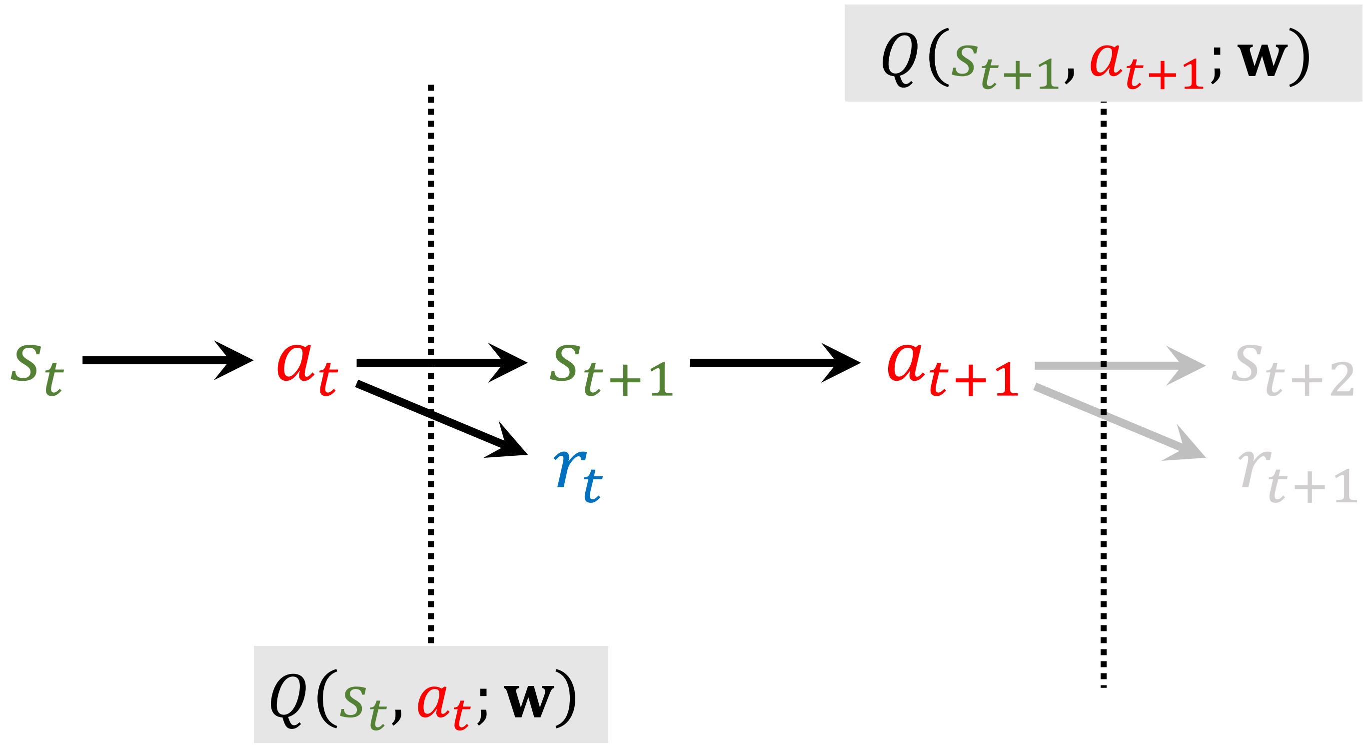
$a_{t+1} = \operatorname{argmax}_a Q(s_{t+1}, a; \mathbf{w})$

Apply DQN to Play Game



Train DQN using TD learning

If it is accurate estimate, then $Q(s_{t+1}, a_{t+1}; \mathbf{w}) = \mathbb{E}\underbrace{[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots]}_{\text{Discounted return: } R_{t+1}}$



If it is accurate estimate, then $Q(s_t, a_t; \mathbf{w}) = \mathbb{E}\underbrace{[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots]}_{\text{Discounted return: } R_t}$

Train DQN using TD learning

If it is accurate estimate, then $Q(s_{t+1}, a_{t+1}; \mathbf{w}) = \mathbb{E}[\underbrace{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots}_{\text{Discounted return: } R_{t+1}}]$

Fact: $R_t = r_t + \gamma \cdot R_{t+1}$.

If it is accurate estimate, then $Q(s_t, a_t; \mathbf{w}) = \mathbb{E}[\underbrace{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots}_{\text{Discounted return: } R_t}]$

Train DQN using TD learning

If it is accurate estimate, then $Q(s_{t+1}, a_{t+1}; \mathbf{w}) = \mathbb{E}[\underbrace{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots}_{\text{Discounted return: } R_{t+1}}]$

If DQN is accurate estimate, then

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

If it is accurate estimate, then $Q(s_t, a_t; \mathbf{w}) = \mathbb{E}[\underbrace{r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots}_{\text{Discounted return: } R_t}]$

Train DQN using TD learning

If it is accurate estimate, then $Q(s_{t+1}, a_{t+1}; \mathbf{w}) = \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots]$

If DQN is accurate estimate, then

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

If it is accurate estimate, then $Q(s_t, a_t; \mathbf{w}) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots]$

Train DQN using TD learning

If DQN is accurate estimate, then

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

Train DQN using TD learning

If DQN is accurate estimate, then

$$\underbrace{Q(s_t, a_t; \mathbf{w})}_{\text{Old estimate}} = \underbrace{r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})}_{\text{TD target}}$$

TD target (more reliable estimate of the value)

Old estimate (less reliable)

Train DQN using TD learning

If DQN is accurate estimate, then

$$Q(s_t, a_t; \mathbf{w}) = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w})$$

- TD target: $y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a; \mathbf{w}_t)$.
- 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.

- $Q^{\star}(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s, a, \pi\right].$

Value-Based Reinforcement Learning

Definition: Optimal action-value function.

- $Q^{\star}(s, a) = \max_{\pi} \mathbb{E}\left[R_t | s, a, \pi\right]$.

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 and action 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.

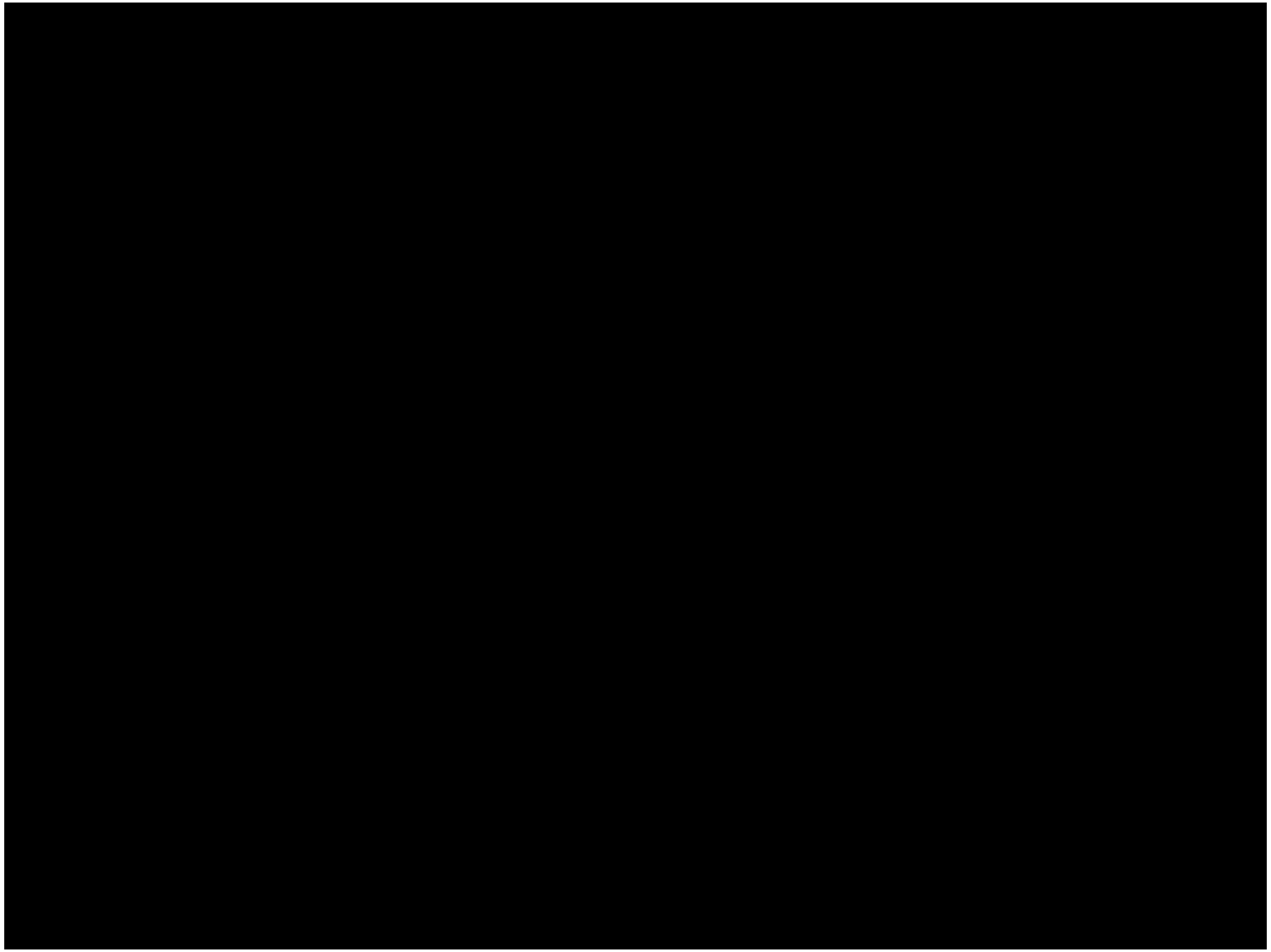
1. Observe state s_t and action a_t.
2. Predict the value: $q_t = Q(s_t, a_t; \mathbf{w}_t)$.
3. Differentiate the value network: $\mathbf{d}_t = \frac{\partial\, Q(s_t, a_t; \mathbf{w})}{\partial\, \mathbf{w}} \Big|_{\mathbf{w}=\mathbf{w}_t}$.
4. Environment provides new state s_{t+1} and reward r_t.
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 and action 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: $\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \cdot (q_t - y_t) \cdot \mathbf{d}_t$.

Play Breakout using DQN



(The video was posted on YouTube by DeepMind)