Lecture 5: More on Humanoid Project, TD Learning, and Policy Gradient

GEARS Program
Summer 2024

Junxi Zhu

From previous lecture…

Value-Based vs Policy-Based Learning

- **Question**: What is the difference between Value-Based Learning and Policy-Based Learning?

Value-Based Learning

- Return (discounted)

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

- Action-value function for policy π

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

- Optimal action-value function

$$Q^*(s_t, a_t) = \max_\pi Q_\pi(s_t, a_t)$$

- Goal of **value-based learning** is to learn or approximate $Q^*(s_t, a_t)$

Policy-Based Learning

- Return (discounted)

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

- Action-value function for policy π

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

- State-value function

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$$

- Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t; \theta)$
- Approximate value function $V_\pi(s_t)$ by

$$V(s; \theta) = \sum_a \pi(a|s; \theta) \cdot Q_\pi(s, a)$$

- Goal of **policy-based learning** is to learn θ that maximizes $J(\theta) = \mathbb{E}_S[V(S; \theta)]$

Value-Based vs Policy-Based Learning

- Value-based learning: such as Q-learning, Deep Q Network (DQN)
- Learns a value function that estimates the expected return of taking a particular action in a given state and following the optimal policy thereafter
- It tries to derive an optimal policy indirectly by maximizing the value function
- Advantages:
  - Optimal action selection: Directly provides a way to select the best action based on the value function
  - Well-studied: Many theoretical guarantees and well-established algorithms exist for value-based methods
- Limitations:
  - Discrete actions: Typically more suited to environments with discrete action spaces
  - Exploration challenges: Often requires careful balance of exploration and exploitation
  - Function approximation issues: May suffer from instability and divergence issues when using function approximation, such as neural networks

Value-Based vs Policy-Based Learning

- Policy-based learning: such as REINFORCE and Proximal Policy Optimization (PPO)
- Directly parameterize the policy and optimize it using gradient-based methods
- The policy determines the action to take in each state
- Advantages:
  - Continuous actions: Naturally handle environments with continuous action spaces
  - Stochastic policies: Can represent stochastic policies, which are beneficial for exploration and robustness
  - Stable update: Often more stable and easier to tune than value-based methods when using function approximation
  - Policy improvement: Directly optimize the policy, potentially leading to better performance in certain environments
- Limitations:
  - Sample efficiency: Typically less sample-efficient compared to value-based methods
  - Variance: Gradient estimates can have high variance, requiring techniques like baseline subtraction to reduce it

Policy-Based Learning

- **Question**: How does the policy network output continuous action? For example, we want the policy network to generate a desired torque within the range of 0 to 10 Nm.

Policy-Based Learning

- **Question**: How does the policy network output continuous action? For example, we want the policy network to generate a desired torque within the range of 0 to 10 Nm.
- We can let the policy network output two values: mean and standard deviation of the Gaussian distribution for output torque
- To choose an action, the policy samples from the Gaussian distribution using the mean and standard deviation produced by the network

Value-Based + Policy-Based Learning

- Combing policy-based and value-based methods can leverage the strengths of both approaches. Actor-Critic methods, such as Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C), use a policy network and a value network to provide better stability and performance
- **Question**: which one below is correct and why? Hint: what is the output of each network?
  - Policy network = Critic, Value network = Actor
  - Policy network = Actor, Value network = Critic

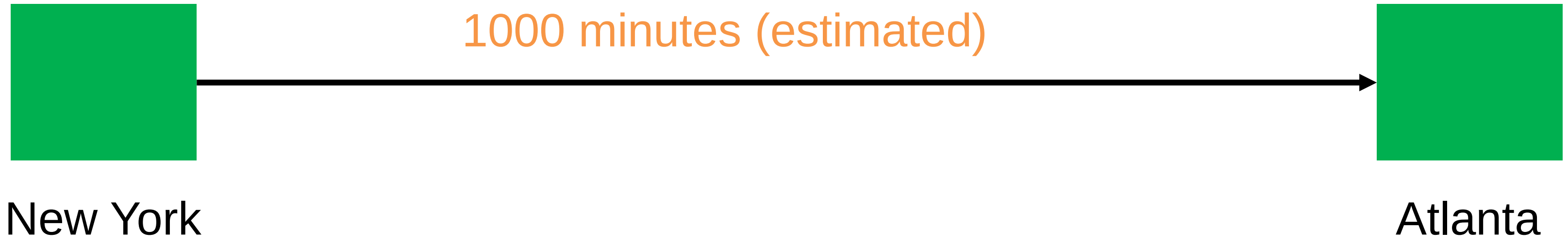
Value-Based + Policy-Based Learning

- Combing policy-based and value-based methods can leverage the strengths of both approaches. Actor-Critic methods, such as Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C), use a policy network (actor) and a value network (critic) to provide better stability and performance
- Question: Why actor is policy network and critic is value network? Why not the opposite?
- Example (Actor-Critic Methods):
    - The critic helps reduce the variance of the policy gradient estimates by providing a baseline, while the actor updates the policy in a direction suggested by the critic
    - This combination often leads to faster and more stable learning

This lecture…

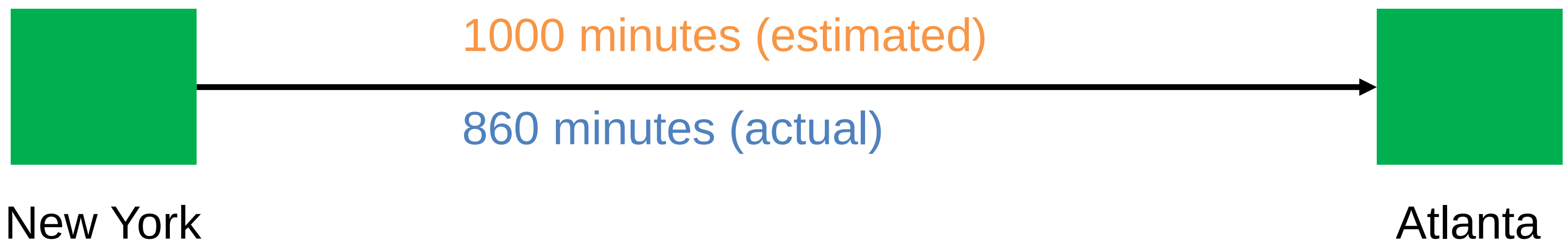
Temporal Difference (TD) Learning

- I want to drive from New York to Atlanta
- Model $Q(w)$ estimates the time cost, e.g., 1000 minutes
- **Question**: How do I update the model?



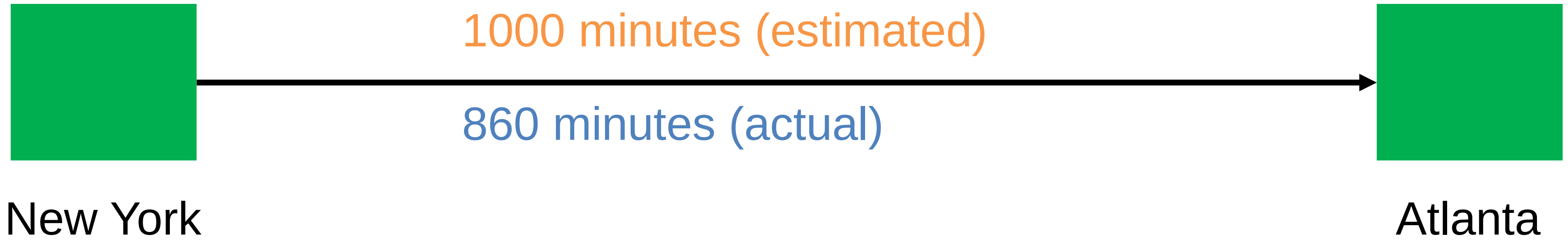
Temporal Difference (TD) Learning

- I want to drive from New York to Atlanta
- Model $Q(w)$ estimates the time cost, e.g., 1000 minutes
- **Question**: How do I update the model?
- Make a prediction: $q = Q(w)$, e.g., $q = 1000$
- Finish the trip and get the target y, e.g., $y = 860$



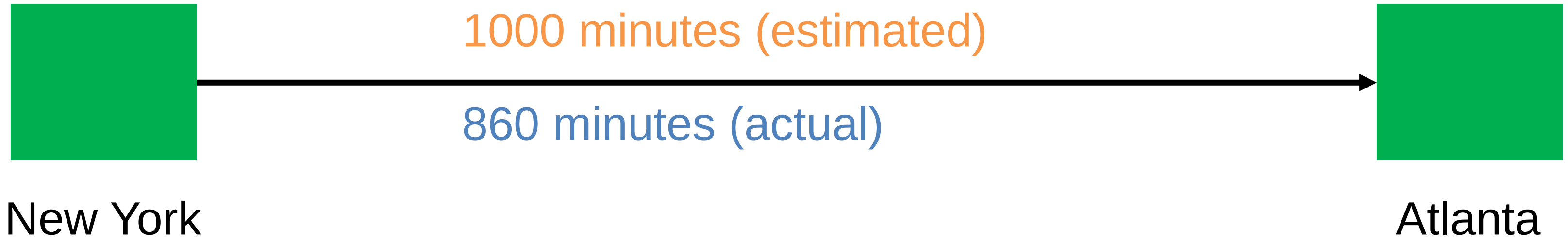
Temporal Difference (TD) Learning

- I want to drive from New York to Atlanta
- Model $Q(w)$ estimates the time cost, e.g., 1000 minutes
- **Question**: How do I update the model?
- Make a prediction: $q = Q(w)$, 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: $\frac{\partial L}{\partial w} = \frac{\partial q}{\partial w} \cdot \frac{\partial L}{\partial q} = \frac{\partial Q(w)}{\partial w} \cdot (q - y)$
- Gradient descent: $w_{t+1} = w_t - \alpha \cdot \left.\frac{\partial L}{\partial w}\right|_{w=w_t}$



Temporal Difference (TD) Learning

- I want to drive from New York to Atlanta
- Model $Q(w)$ estimates the time cost, e.g., 1000 minutes
- **Question**: How do I update the model?
- Can I update the model before finishing the trip?

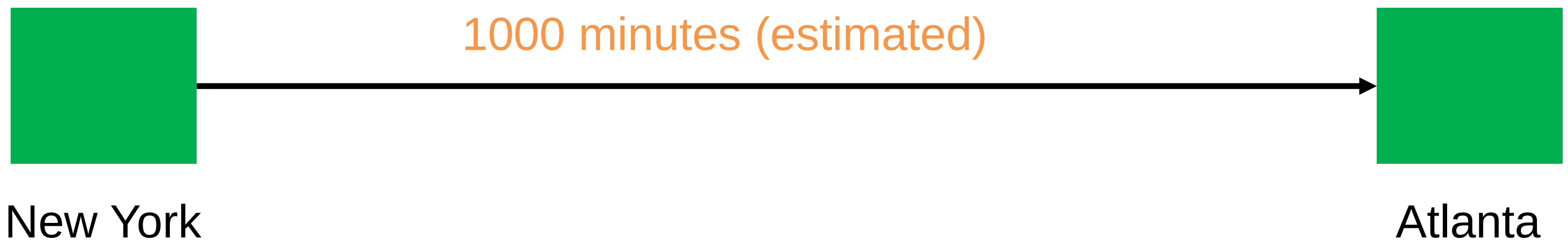


Temporal Difference (TD) Learning

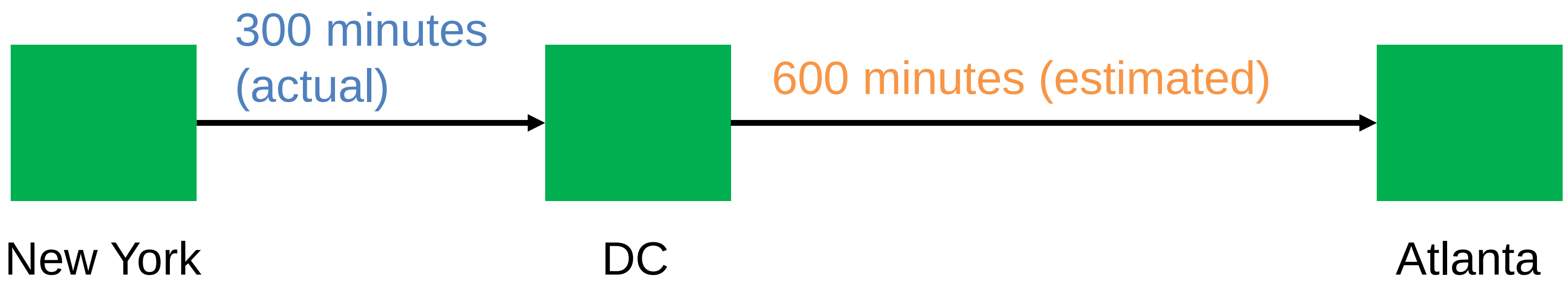
- I want to drive from New York to Atlanta
- Model $Q(w)$ estimates the time cost, e.g., 1000 minutes
- **Question**: How do I update the model?
- Can I update the model before finishing the trip?
- Can I get better w as soon as I arrived at DC?

Temporal Difference (TD) Learning

- Model's estimate: New York to Atlanta: 1000 minutes (estimated)

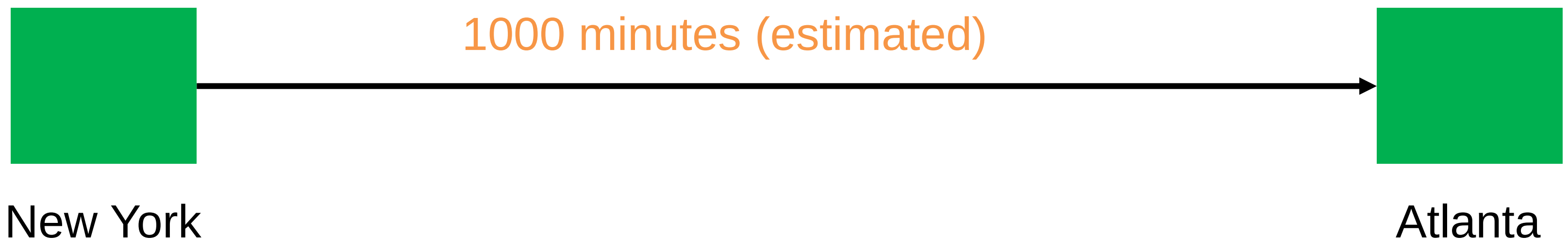


- I arrived at DC; actual time cost is 300 minutes
- Model now updates its estimate: DC to Atlanta: 600 minutes (estimated)



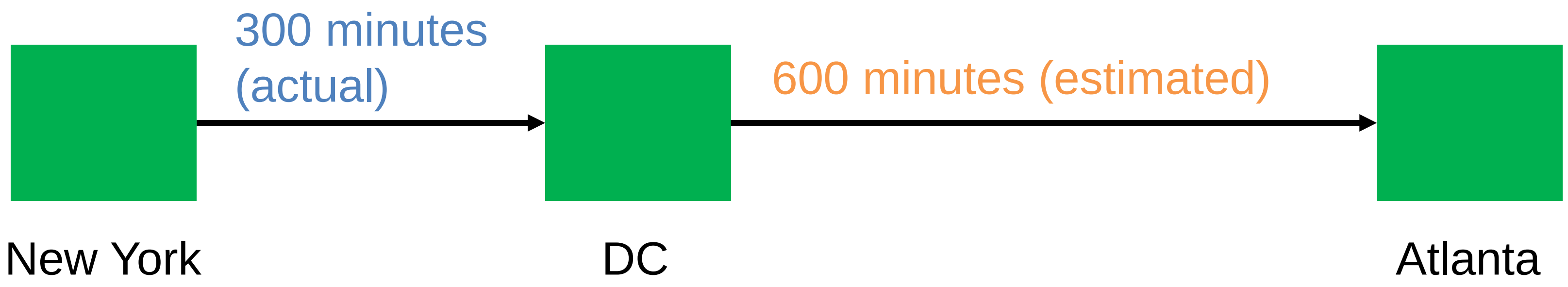
Temporal Difference (TD) Learning

- Model's estimate: $Q(w) = 1000$ minutes



- Updated estimate: $300 + 600 = 900$ minutes

TD target

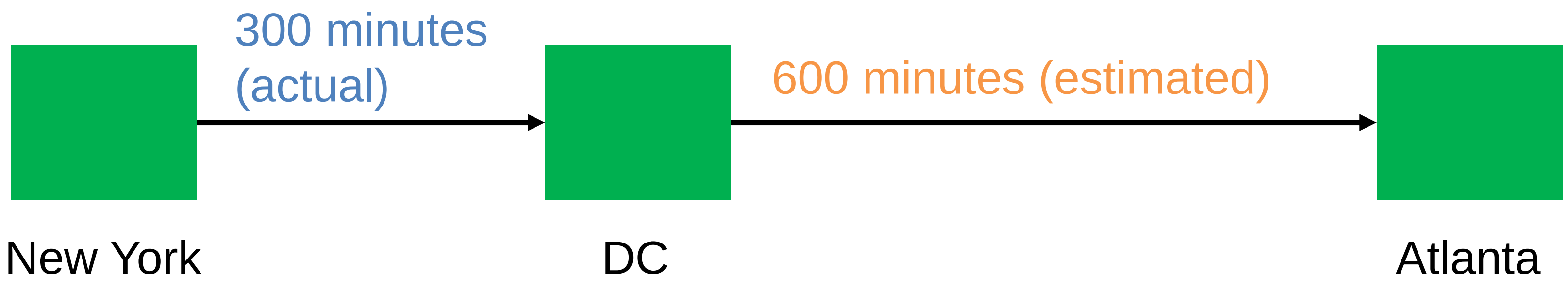


Temporal Difference (TD) Learning

- Model's estimate: $Q(w) = 1000$ minutes
- Updated estimate: $300 + 600 = 900$ minutes

TD target

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

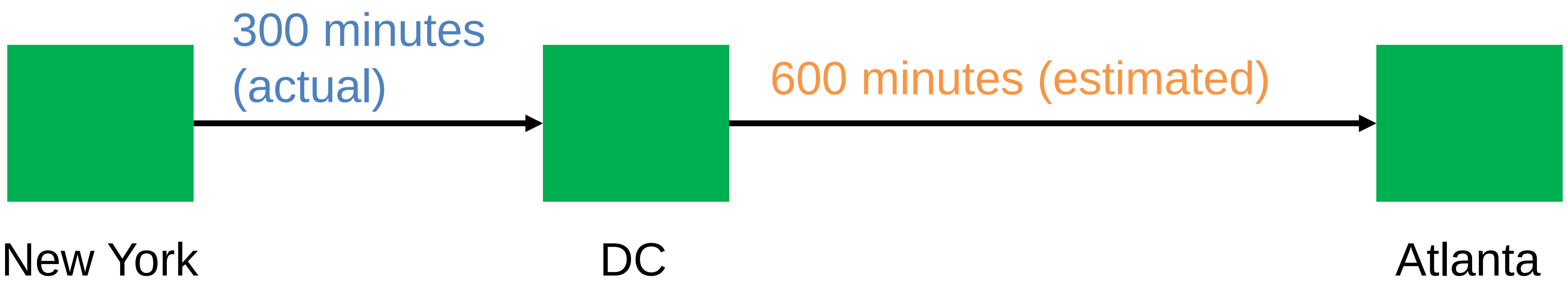


Temporal Difference (TD) Learning

- Model's estimate: $Q(w) = 1000$ minutes
- Updated estimate: $300 + 600 = 900$ minutes

TD target

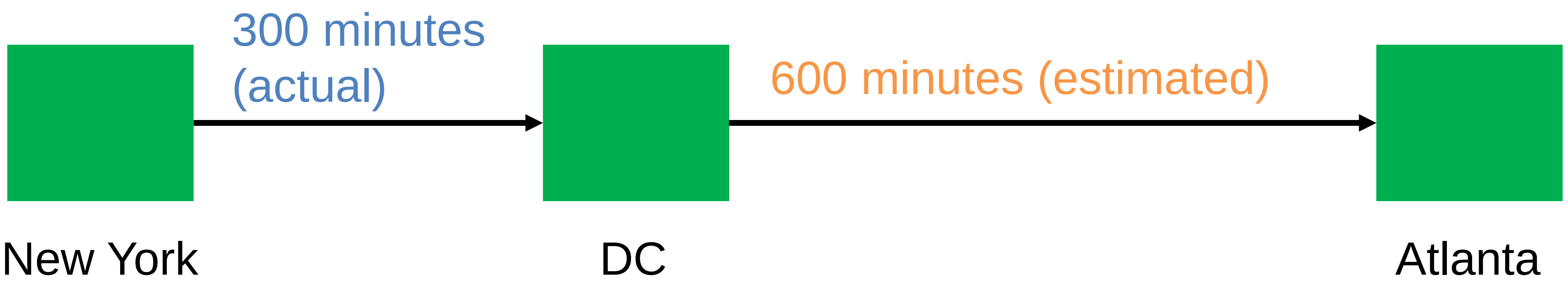
- TD target $y = 900$ is a more reliable estimate than 1000
- Loss: $L = \frac{1}{2}(\underbrace{Q(w) - y}_{\text{TD error}})^2$



Temporal Difference (TD) Learning

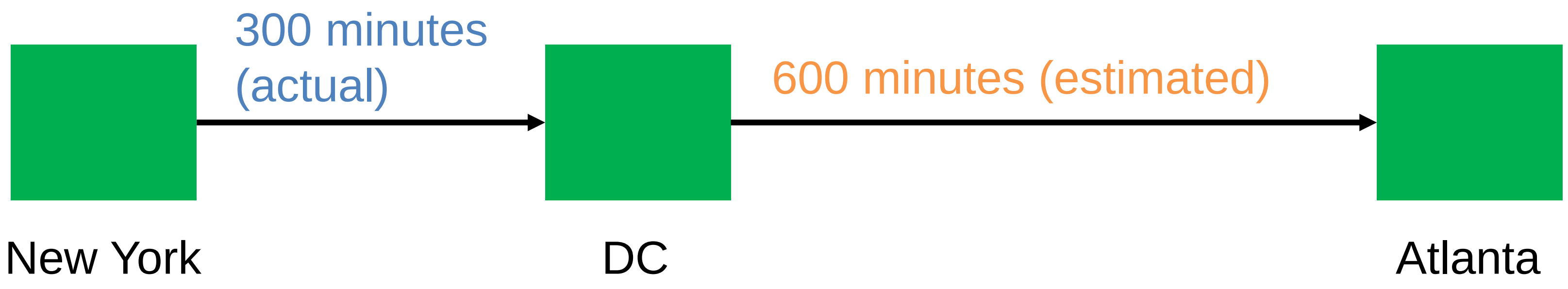
- Model's estimate: $Q(w) = 1000$ minutes
- Updated estimate: $300 + 600 = 900$ minutes

  TD target
- TD target $y = 900$ is a more reliable estimate than 1000
- Loss: $L = \frac{1}{2}(Q(w) - y)^2$
- Gradient: $\frac{\partial L}{\partial w} = \underbrace{(1000 - 900)}_{\text{TD error}} \cdot \frac{\partial Q(w)}{\partial w}$
- Gradient descent: $w_{t+1} = w_t - \alpha \cdot \left.\frac{\partial L}{\partial w}\right|_{w=w_t}$



Why does TD Learning Work?

- Model's estimate:
  - NY to Atlanta: 1000 minutes
  - DC to Atlanta: 600 minutes
  - Thus, NY to DC: 400 minutes
- Ground truth: NY to DC: 300 minutes
- TD error: $\delta = 400 - 300 = 100$



How to Apply TD Learning to DQN?

- In the "driving time" example, we have the equation

$$T_{NYC \to ATL} \approx T_{NYC \to DC} + T_{DC \to ATL}$$

Model's estimate　　Actual time　　Model's estimate

- In deep reinforcement learning:

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

Monte Carlo vs Bootstrapping

(Recall) How to Train DQN?

- Train DQN using TD learning
- Prediction: $Q(s_t, a_t; w)$
- TD Target

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

- Loss:

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

- Gradient descent:

$$w_{t+1} = w_t - \alpha \left.\frac{\partial L_t}{\partial w}\right|_{w=w_t}$$

Monte Carlo

- We could alternatively define loss function as

$$L_t = \frac{1}{2}[Q(s_t, a_t; w) - u_t]^2$$

- Where $u_t = \sum_{i=0}^{n-t} \gamma^i r_{t+i}$. Essentially, we play the game till the end and obtain all the rewards $r_1, r_2, \cdots, r_n$
- This is called Monte Carlo method
- **Advantage**: Since u_t is an unbiased estimate of $Q_\pi(s_t, a_t)$, the estimated value network is also unbiased
- **Disadvantage**: There is large uncertainty associated with the random variable U_t, thus the estimated value has high variance and will lead to slow convergence of the training of the value network

Bootstrapping

- If we still use the original loss function

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

- Where $y_t = r_t + \gamma Q(s_{t+1}, a_{t+1}; w_t)$
- This is called Bootstrapping method
- **Advantage**: Single-step TD has small uncertainty because it only depends on S_{t+1}, A_{t+1}, the estimated value has smaller variance and leads to faster convergence of the training of the value network
- **Disadvantage**: Since $q(s, a; w)$ is an approximation of $Q_\pi(s, a)$, it will lead to a biased estimate (why?)

Bootstrapping

- Recall that

$$q(s_t, a_t; w) \leftarrow r_t + \gamma q(s_{t+1}, a_{t+1}; w_t)$$

- Then

$$\begin{aligned}
&q(s_{t+1}, a_{t+1}; w_t) \;\; \text{underestimates(overestimates)} \;\; Q_\pi(s_{t+1}, a_{t+1}) \\
&\Rightarrow \; r_t + \gamma q(s_{t+1}, a_{t+1}; w_t) \;\; \text{underestimates(overestimates)} \;\; Q_\pi(s_t, a_t) \\
&\Rightarrow q(s_t, a_t; w) \;\; \text{underestimates(overestimates)} \;\; Q_\pi(s_t, a_t)
\end{aligned}$$

- Thus, bootstrapping causes the bias to propagate from (s_{t+1}, a_{t+1}) to (s_t, a_t)

Policy Gradient

Recall from Previous Lecture…

- Return (discounted)

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

- Action-value function for policy π

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

- State-value function

$$V_\pi(s_t) = \mathbb{E}_A[Q_\pi(s_t, A)] = \sum_a \pi(a|s_t) \cdot Q_\pi(s_t, a)$$

- Approximate policy function $\pi(a|s_t)$ by policy network $\pi(a|s_t; \theta)$
- Approximate value function $V_\pi(s_t)$ by

$$V(s; \theta) = \sum_a \pi(a|s; \theta) \cdot Q_\pi(s, a)$$

- Goal of **policy-based learning** is to learn θ that maximizes $J(\theta) = \mathbb{E}_S[V(S; \theta)]$

Policy-Based Reinforcement Learning

- Approximate state-value function

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy-based learning: Learn θ that maximizes $J(\theta) = \mathbb{E}_S[V(S;\theta)]$
- How to improve θ? Policy gradient ascent!
- Observe state s
- Update policy by: $\theta \leftarrow \theta + \beta \cdot \frac{\partial V(s;\theta)}{\partial \theta}$

Policy gradient

Policy Gradient (Not Rigorous Derivation)

- Approximate state-value function

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy gradient: Derivative of $V(s;\theta)$ w.r.t θ

$$\frac{\partial V(s;\theta)}{\partial \theta} = \frac{\partial \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)}{\partial \theta}$$

$$= \sum_a \frac{\partial \pi(a|s;\theta) \cdot Q_\pi(s,a)}{\partial \theta}$$

Push derivative inside the summation

$$= \sum_a \frac{\partial \pi(a|s;\theta)}{\partial \theta} \cdot Q_\pi(s,a)$$

Pretend Q_π is independent of θ (It may not be true!)

Policy Gradient (Not Rigorous Derivation)

- Approximate state-value function

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy gradient: Derivative of $V(s;\theta)$ w.r.t θ

$$\frac{\partial V(s;\theta)}{\partial \theta} = \sum_a \frac{\partial \pi(a|s;\theta)}{\partial \theta} \cdot Q_\pi(s,a)$$

$$= \sum_a \pi(a|s;\theta) \cdot \boxed{\frac{1}{\pi(a|s;\theta)} \cdot \frac{\partial \pi(a|s;\theta)}{\partial \theta}} \cdot Q_\pi(s,a)$$

$$= \sum_a \pi(a|s;\theta) \cdot \frac{\partial \log \pi(a|s;\theta)}{\partial \theta} \cdot Q_\pi(s,a)$$

Chain rule: $\frac{\partial \log \pi(\theta)}{\partial \theta} = \frac{1}{\pi(\theta)} \cdot \frac{\partial \pi(\theta)}{\partial \theta}$

Policy Gradient (Not Rigorous Derivation)

- Approximate state-value function

$$V(s;\theta) = \sum_a \pi(a|s;\theta) \cdot Q_\pi(s,a)$$

- Policy gradient: Derivative of $V(s;\theta)$ w.r.t θ

$$\frac{\partial V(s;\theta)}{\partial\theta} = \sum_a \frac{\partial \pi(a|s;\theta)}{\partial\theta} \cdot Q_\pi(s,a)$$

$$= \sum_a \pi(a|s;\theta) \cdot \frac{\partial \log \pi(a|s;\theta)}{\partial\theta} \cdot Q_\pi(s,a)$$

$$= \mathbb{E}_{A\sim\pi(\cdot|s;\theta)}\left[\frac{\partial \log \pi(A|s;\theta)}{\partial\theta} \cdot Q_\pi(s,A)\right]$$

This expectation is taken w.r.t the random variable $A\sim\pi(\cdot\,|s;\theta)$

Calculate Policy Gradient

- Policy gradient:

$$\frac{\partial V(s;\theta)}{\partial \theta} = \mathbb{E}_{A\sim\pi(\cdot|s;\theta)}\left[\frac{\partial \log \pi(A|s;\theta)}{\partial \theta} \cdot Q_\pi(s,A)\right]$$

- Step 1: Randomly sample an action $\hat{a}$ according to $\pi(\cdot\,|s;\theta)$
- Step 2: Calculate $g(\hat{a},\theta) = \frac{\partial \log \pi(\hat{a}|s;\theta)}{\partial \theta} \cdot Q_\pi(s,\hat{a})$
- Step 3: Use $g(\hat{a},\theta)$ as an unbiased approximation to the policy gradient $\frac{\partial V(s;\theta)}{\partial \theta}$

Update Policy Network using Policy Gradient

1. Observe the state s_t
2. Randomly sample action a_t according to $\pi(\cdot\,|s;\theta)$
3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate)
4. Differentiate policy network $d_{\theta,t} = \frac{\partial \log \pi(a_t|s_t;\theta)}{\partial \theta}\Big|_{\theta=\theta_t}$
5. Approximate policy gradient: $g(a_t, \theta_t) = q_t \cdot d_{\theta,t}$
6. Update policy network: $\theta_{t+1} = \theta_t + \beta \cdot g(a_t, \theta_t)$

Update Policy Network using Policy Gradient

1. Observe the state s_t
2. Randomly sample action a_t according to $\pi(\cdot\,|s;\theta)$
3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate) How?
4. Differentiate policy network $d_{\theta,t} = \frac{\partial \log \pi(a_t|s_t;\theta)}{\partial \theta}\Big|_{\theta=\theta_t}$
5. Approximate policy gradient: $g(a_t, \theta_t) = q_t \cdot d_{\theta,t}$
6. Update policy network: $\theta_{t+1} = \theta_t + \beta \cdot g(a_t, \theta_t)$

Update Policy Network using Policy Gradient

1. Observe the state s_t
2. Randomly sample action a_t according to $\pi(\cdot\,|s;\theta)$
3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate)

How?

Option 1: REINFORCE

- Play the game to the end and generate the trajectory

$$s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_T, a_T, r_T$$

- Compute the discounted return $u_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$, for all t
- Since $Q_\pi(s_t, a_t) = \mathbb{E}[U_t]$, we can use u_t to approximate $Q_\pi(s_t, a_t)$
- Thus, use $q_t = u_t$

Update Policy Network using Policy Gradient

1. Observe the state s_t
2. Randomly sample action a_t according to $\pi(\cdot\,|s;\theta)$
3. Compute $q_t \approx Q_\pi(s_t, a_t)$ (some estimate)

How?

Option 2: Approximate Q_π using a neural network

- This leads to the actor-critic method (next lecture!)

Let’s check the code…