Day - 99 (March 9, 2025) Falgun 25, 2081

1. Example of Continous State - Space Applications (6min)
2. Ethical use of Recommender Systems (10min)
3. Tensorflow Implementation of Content-based filtering (4min)
4. Reinforcement Learning, Mars Rover Examples (14min)
5. The Return in Reinforcement Learning (10min)
6. Lunar Lander (5min)
7. Algorithm - Refinement Improved Neural Network Architecture. (3min)
8. Algorithm Reinforced ϵ-greedy Policy (8min)

Ethical Considerations

Recommender

widely used.

Ethics

depends upon the individual.

What is the goal of the recommender system?

Recommend:

- Movies most likely to be rated 5 stars by user
- Products most likely to be purchased
- Ads most likely to be clicked on + high bid
- Products generating the largest profit
- Video leading to maximum watch time

trying to show you things
that are most useful to you?

DeepLearning.AI Stanford ONLINE Andrew Ng

Payday loans → Bad Example

Squeeze Customers

Exploitation.

Bid higher for Ads → More profits

- Be transparent with users
- Powerful, lucrative and Handy Technology.

Ethical considerations with recommender systems

Travel industry

Good travel experience to more users

Bid higher for ads

More profitable

1) payday loans

Exploitation

Recommender

Amelioration: Don't accept ads from exploitative businesses

It can afford to pay

Other problematic cases:

- Maximizing user engagement (e.g. watch time) has led to large social media/video sharing sites to amplify conspiracy theories and hate/toxicity

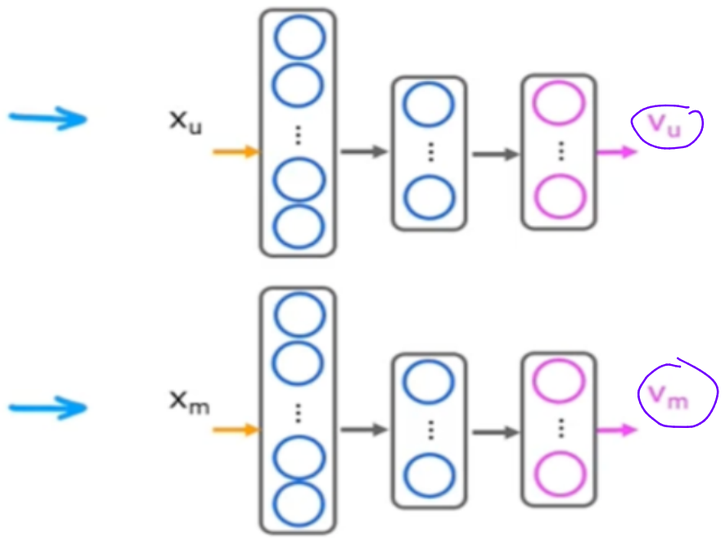
  → Amelioration : Filter out problematic content such as hate speech, fraud, scams and violent content

- Can a ranking system maximize your profit rather than users' welfare be presented in a transparent way?

  Amelioration : Be transparent with users

I would encourage you and

Tensorflow Implementation (key Concepts)




```
user_NN = tf.keras.models.Sequential([
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(32)
])

item_NN = tf.keras.models.Sequential([
  tf.keras.layers.Dense(256, activation='relu'),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(32 )
])
```

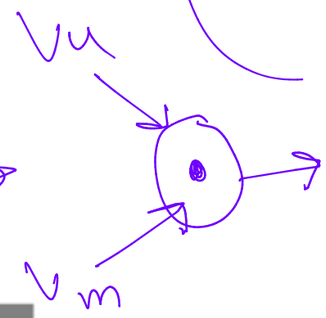
```
# create the user input and point to the base network
input_user = tf.keras.layers.Input(shape=(num_user_features))
vu = user_NN(input_user)
vu = tf.linalg.l2_normalize(vu, axis=1)

# create the item input and point to the base network
input_item = tf.keras.layers.Input(shape=(num_item_features))
vm = item_NN(input_item)
vm = tf.linalg.l2_normalize(vm, axis=1)

# measure the similarity of the two vector outputs
output = tf.keras.layers.Dot(axes=1)([vu, vm])

# specify the inputs and output of the model
model = Model([input_user, input_item], output)

# Specify the cost function
cost_fn = tf.keras.losses.MeanSquaredError()
```



it's very similar to how we have previously

Reinforcement Learning

Ambigious Action (left / Right?) Direction

Position of Helicopter → How to move Control Sticks

Application →

Autonomous Helicopter

Weight: 32 pounds

How to fly it Autonomously?

This is actually the Stanford autonomous helicopter, weighs 32 pounds and

DeepLearning.AI Stanford Andrew Ng

using RL!

State s → action 'a'

Supervised Learning

$\begin{bmatrix} x \\ \uparrow \end{bmatrix} \not\rightarrow \begin{bmatrix} y \\ \uparrow \end{bmatrix}$

Limitations: (Not possible) x, y

Reward function for Helicopter

positive reward: helicopter flying well +1

Negative Reward: Helicopter flying poorly −1000

Reinforcement Learning
Tell it what to do?
Rather than How to do it??

Supervised Sucks at Autonomous Learning :

What to do ? by RL..

Applications

- Controlling robots
- Factory optimization
- Financial (stock) trading
- Playing games (including video games)

Robotic Dog Example

[Thanks to Zico Kolter]

what's the best way to place its legs to get over a given obstacle.

t later this week in the
ou implement for yourself

↳ used in few Applications

↳ Backed by Researcher

Researchy with Rewards - ⊕ , ⊖

MARS ROVER EXAMPLE:

Rewards

Left ←

4 3 2 1

0 0 0 100

Terminal State.

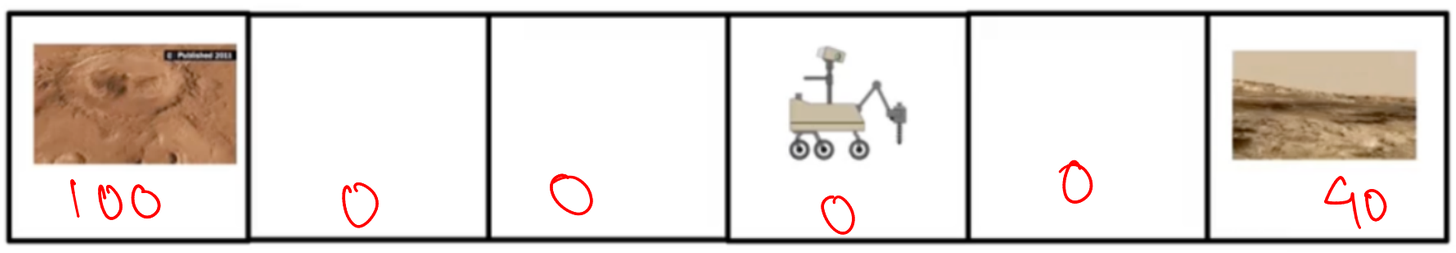
Reward pts → 100

Mars Rover Example

Right →

0 0 40

Terminal State.



state 1 2 3 4 5 6

?? Voluable Reward decides it.

← Left ? Retrun ? → Right

(S, a, R(S), S')

4 3 2 1

0 0 0 100

0 0 40

0 0 0 0 100

[Credit: Jagriti Agrawal, Emma Brunskill]

DeepLearning.AI Stanford ONLINE State 6 also has Andrew Ng

S → State

S' → New state.

(4, ←, 10, 3) ←

(S, a, R(S), S')

Example of Return

100	50 ←	25 ←	12.5 ←	6.25 ←	40
100	0	0	0	0	40
1	2	3	4	5	6

← return

← reward

$\gamma = 0.5$

The return depends on the actions you take.

100	2.5 →	5 →	10 →	20 →	40
100	0	0	0	0	40
1	2	3	4	5	6

$0+(0.5)0+(0.5)^2 40 = 10$

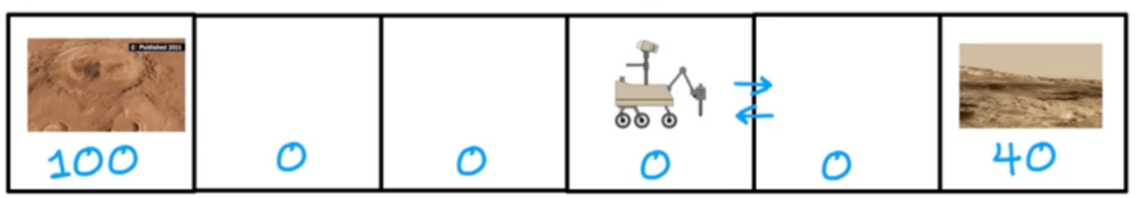
100	50 ←	25 ←	12.5 ←	20 →	40
100	0	0	0	0	40
1	2	3	4	5	6

0

DeepLearning.AI Stanford ONLINE Andrew Ng

Mars Rover Example

terminal state terminal state



state 1 2 3 4 5 6

left right

state 4 3 2 1

0 0 0 100

0 0 40

0 0 0 0 0 100

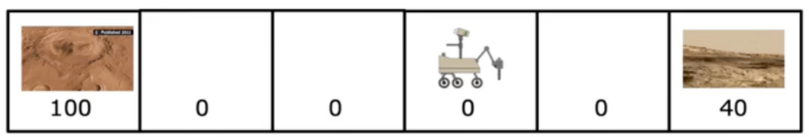
$\rightarrow (s, a, R(s), s')$

$(4, \leftarrow, 0, 3)$

l, Emma Brunskill]

.AI Stanford ONLINE Andrew Ng

Return



state 1 2 3 4 5 6

Return $= 0+(0.9)0+(0.9)^2 0+(0.9)^3 100 = 0.729 \times 100 = 72.9$

Return $= R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$ (until terminal state)

Discount Factor $\gamma = 0.9$ 0.99 0.999

$\gamma = 0.5$

Return $= 0+(0.5)0+(0.5)^2 0+(0.5)^3 100 = 12.5$

and this turns out to be a return of 12.5.

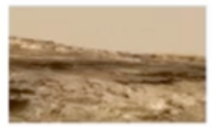
DeepLearning.AI Stanford ONLINE Andrew Ng

$0 < \gamma < 1$ (γ = discount factor)

$G = R_1 + \gamma R_2 + \gamma R_2 + \gamma^2 R_3 + \cdots$

The Return in Reinforcement Learning (Navigation Technique) | RL NT.

Return

					
100	0	0	0	0	40
state 1	2	3	4	5	6

Return = $0 + (0.9)0 + (0.9)^2 0 + (0.9)^3 100$ $\Rightarrow 0.729 \times 100 \Rightarrow 72.9$

Return $\Rightarrow R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$ (until terminal state)

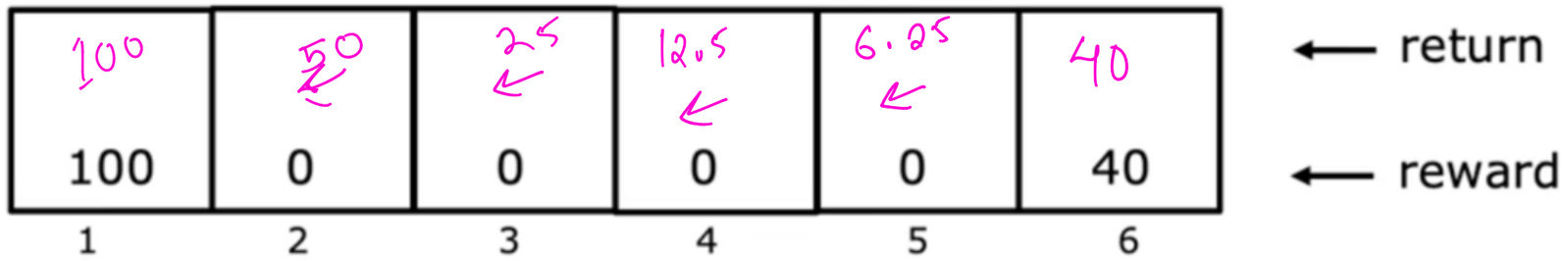
Discount factor $\Rightarrow$ $\gamma = 0.9$ $\gamma = 0.99$ 0.999

$\gamma \Rightarrow 0.5$

Return $\Rightarrow 0 + (0.5)0 + (0.5)^2 0 + (0.5)^3 100 \Rightarrow 12.5$

DeepLearning.AI Stanford | ONLINE Andrew Ng

Example of Return

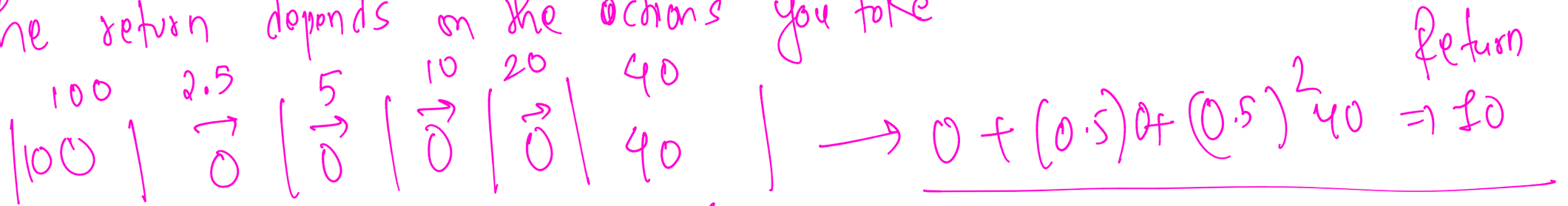


← return

← reward

$\gamma = 0.5$

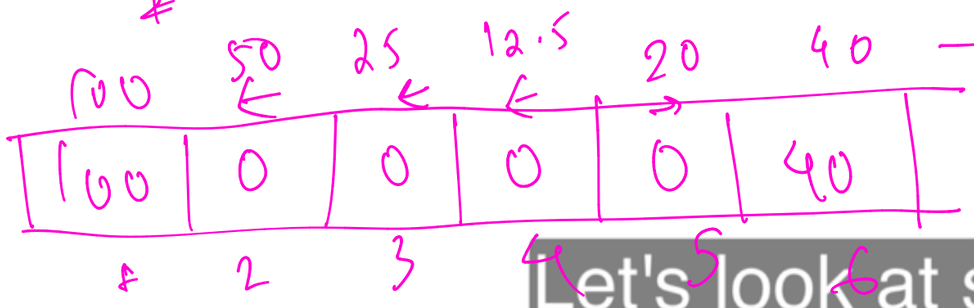
the return depends on the actions you take



→ $0 + (0.5)0 + (0.5)^2 40 = 10$

Returns

Actions Depends



$0 + (0.5)40 = 20$.

Let's look at some concrete examples of returns.

DeepLearning.AI Stanford ONLINE Andrew Ng

Explanation of the Lecture on Return in Reinforcement Learning

In reinforcement learning (RL), the **return** is a measure of the cumulative rewards an agent receives over time, considering the effect of **delayed rewards** and **discounting future rewards**.

1️⃣ Understanding Return with an Analogy

Imagine you have two options:

- Pick up a **$5 bill immediately** at your feet.
- Walk **30 minutes to get a $10 bill**.

Even though $10 is greater than $5, the extra effort (or delay) makes the decision less straightforward. **Return** in RL captures this trade-off by prioritizing earlier rewards more than later ones.

2️⃣ Discount Factor (γ, Gamma)

Since immediate rewards are generally more valuable, RL introduces a **discount factor** $0 < \gamma < 1$ to reduce the importance of future rewards. The return G is calculated as:

$$G = R_1 + \gamma R_2 + \gamma^2 R_3 + \gamma^3 R_4 + \dots$$

- **Immediate rewards are fully counted** (multiplied by 1).
- **Future rewards are discounted** (multiplied by $\gamma, \gamma^2, \gamma^3, \dots$).
- **Common values** for γ are **0.9, 0.99, 0.999**, which make future rewards still significant but slightly less valuable.

3️⃣ Example with a Mars Rover

Consider a Mars Rover moving between states and earning rewards:

- Moving **left** from state **4 → 3 → 2 → 1 (terminal state)** gets a reward **100 at state 1**.
- Using γ **= 0.9**, the return is:

$$G = 0 + 0.9(0) + 0.9^2(0) + 0.9^3(100) = 72.9$$

- If γ **= 0.5**, the same path gives:

$$G = 0 + 0.5(0) + 0.5^2(0) + 0.5^3(100) = 12.5$$

↓

Since **a smaller discount factor reduces the value of future rewards**, the return is much lower when $\gamma = 0.5$.

4️⃣ Comparison of Different Actions

- If the Rover **always goes left**, its return values are higher in most cases.
- If it **always goes right**, its return values are generally lower.
- **Smart strategy:** Going left most of the time but switching right when near a good reward (e.g., from state 5).

5️⃣ Financial Interpretation of Discount Factor

- Similar to **time value of money**, where **$1 today is worth more than $1 in the future** because of interest.
- RL uses the **discount factor** to encourage quicker rewards over delayed ones.

6️⃣ Handling Negative Rewards

- If rewards include **negative values** (e.g., a **-10 penalty**), the agent will try to **delay** them as much as possible.
- Example: In finance, if you have to pay **$10**, it's better to **postpone the payment**, as inflation or interest reduces its real value.

📌 Summary

- **Return** = Sum of rewards weighted by discount factor.
- **Discount factor (γ)** reduces the importance of future rewards.
- **Smaller γ makes the system more impatient** (future rewards lose value faster).
- **Optimizing return helps decide the best actions** in RL.
- **Financial analogy:** Similar to interest rates and time value of money.
- **Negative rewards are pushed into the future** when possible.

This concept is essential for understanding **how reinforcement learning agents evaluate long-term rewards** and **make optimal decisions over time.** 🚀

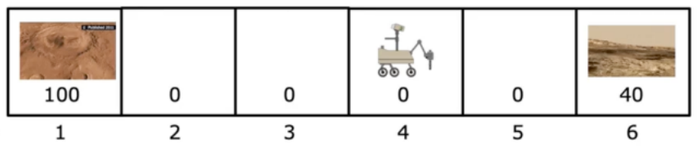
Example of Return

$\gamma = 0.5$

The return depends on the actions you take.

DeepLearning.AI Stanford ONLINE Andrew Ng

Return



Return $= 0+(0.9)0+(0.9)^2 0+(0.9)^3 100 = 0.729 \times 100 = 72.9$

Return $= R_1 + \gamma R_2 + \gamma^2 R_3 + \cdots$ (until terminal state)

Discount Factor $\gamma = 0.9$ 0.99 0.999

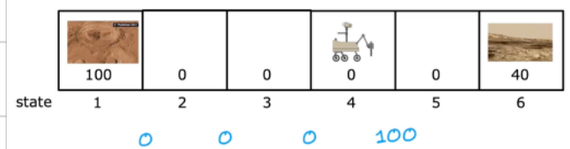
$\gamma = 0.5$

Return $= 0+(0.5)0+(0.5)^2 0+(0.5)^3 100 = 12.5$

and this turns out to be a return of 12.5.

DeepLearning.AI Stanford ONLINE Andrew Ng

Return



DeepLearning.AI Stanford ONLINE Andrew Ng

Return in Reinforcement Learning (RL) - Summary

1 What is Return?

The **return** is the cumulative reward an agent receives, considering future rewards with a **discount factor (γ)**.

2 Discount Factor (γ, Gamma)

- $G = R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$
- **Higher γ (e.g., 0.99)** → Future rewards are important.
- **Lower γ (e.g., 0.5)** → Future rewards lose value faster.

3 Mars Rover Example

- Moving **left** (State 4 → 3 → 2 → 1) gets **100 reward** at the end.
- With γ **= 0.9**, return = **72.9**; with γ **= 0.5**, return = **12.5**.
- Smart strategies mix **left and right** for optimal return.

4 Financial & Negative Rewards Interpretation

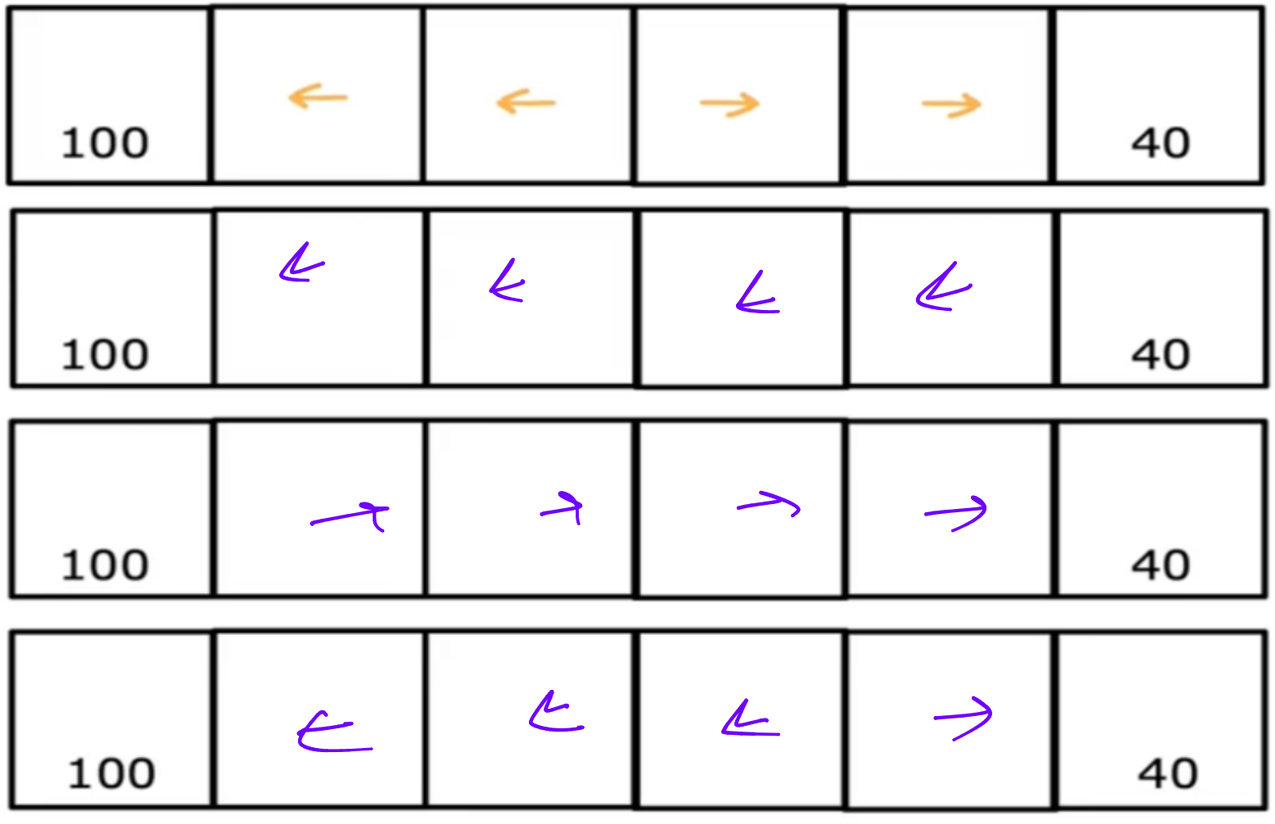
- **Similar to time value of money**: $1 today > $1 in the future.
- **Negative rewards (penalties) are delayed** to reduce impact.

Key Takeaways

- **Return = Total rewards, discounted over time.**
- **γ controls impatience**: High γ values prioritize long-term rewards.
- **RL agents optimize return to make the best decisions.** 🚀

Policies in RL: Making Decisions

Policy



State $\xrightarrow[\pi]{\text{policy}}$ action a

$\pi(s) = a$

$\pi(1) = \leftarrow$

$\pi(2) = \leftarrow$

$\pi(3) = \leftarrow$

$\pi(4) = \leftarrow$

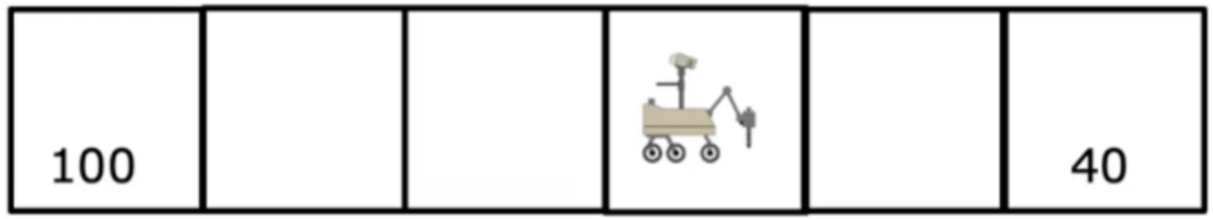
$\pi(5) = \rightarrow$

A policy is a function $\pi(s) =$ a mapping from states to actions that tells you what action 'a' to take in a given state 's'.

Another way we could choose actions is to always go for

DeepLearning.AI Stanford Andrew Ng

The goal of reinforcement learning



Find a policy π that tells you what action ($a = \pi(s)$) to take in every state (s) so as to maximize the return.

take in every state so as
to maximize the return.

DeepLearning.AI Stanford ONLINE Andrew Ng

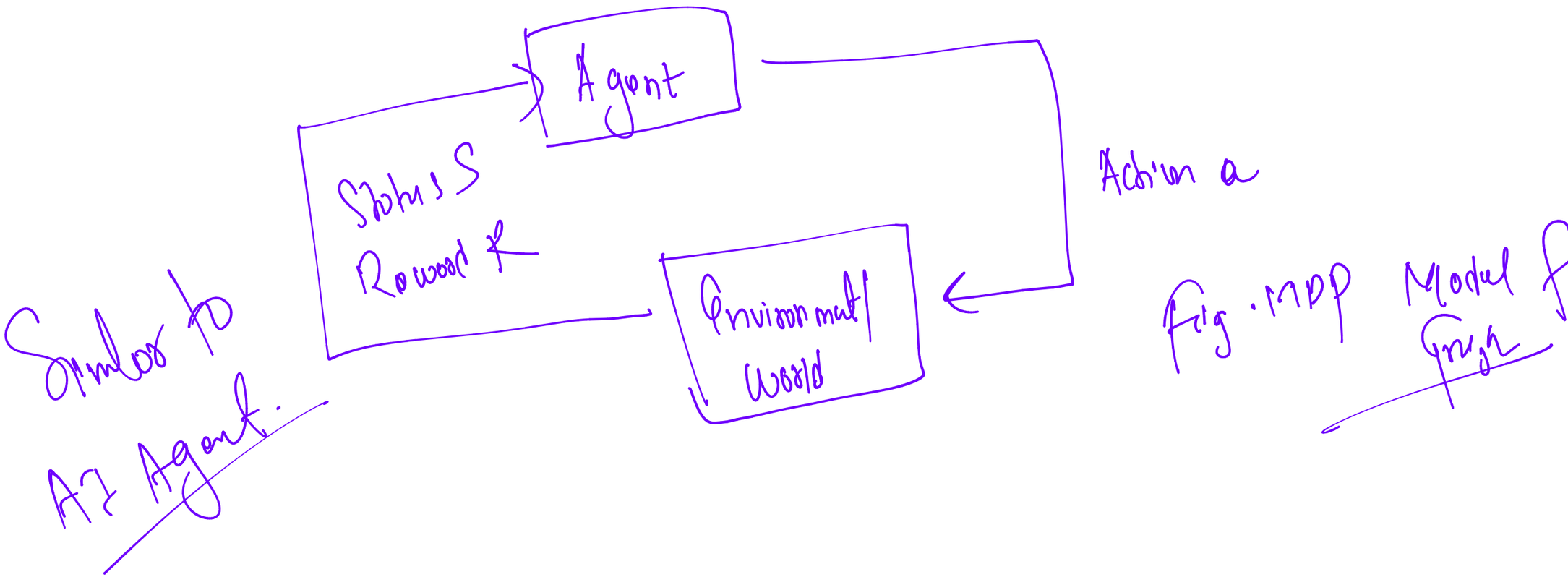
	Mars rover 	Helicopter 	Chess 
states	6 states	Position of Helicopter	pieces on board.
actions		How to Move Controlstick	Possible moves
rewards	100, 0, 40	+1, -1000, 0.99	+1, 0, -1.
discount factor γ	0.5		0.999, 0.995
return	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$	$R_1 + \gamma R_2 + \gamma^2 R_3 + \dots$
policy π	| 100 | ← | ← | ← | → | 40 |	find $\pi(s) = a$	find $\pi(s) = a$ Given reward pick action

and then finally, a policy whose job it

DeepLearning.AI Stanford ONLINE Andrew Ng

Markov Decision Process (MDP)

Focus only on current state.



that's just the formalism
that we've been

DeepLearning.AI Stanford ONLINE Andrew Ng

State - action value function Definition:

State action value function

$Q(s,a)$ = Return if you
- Start in state s.
- take action a (once).
- then behave optimally after that.

100	50 ←	25 ←	12.5 ←	20 →	40 ← Action
100	0	0	0	0	40 ← Reward

$Q(2,\leftarrow)$ $Q(2,\rightarrow)+2.5$

Go ← $Q(2,\leftarrow)$

100	0	0	0	0	40
1	50 | 2	12.5 | 3	4	5	6

$Q(4,\leftarrow)$ $\Rightarrow 12.5$

$Q(s,a)$

$Q(2,\rightarrow)$

$0 \quad 0 \quad 0 \quad 100$

$Q + (0.5)0 + (0.5)^2 0 + (0.5)^3 400$

$\Rightarrow 12.5$

$Q(2,\rightarrow)$

$Q(2,\leftarrow) \Rightarrow 50$

$0 + (0.5)100$

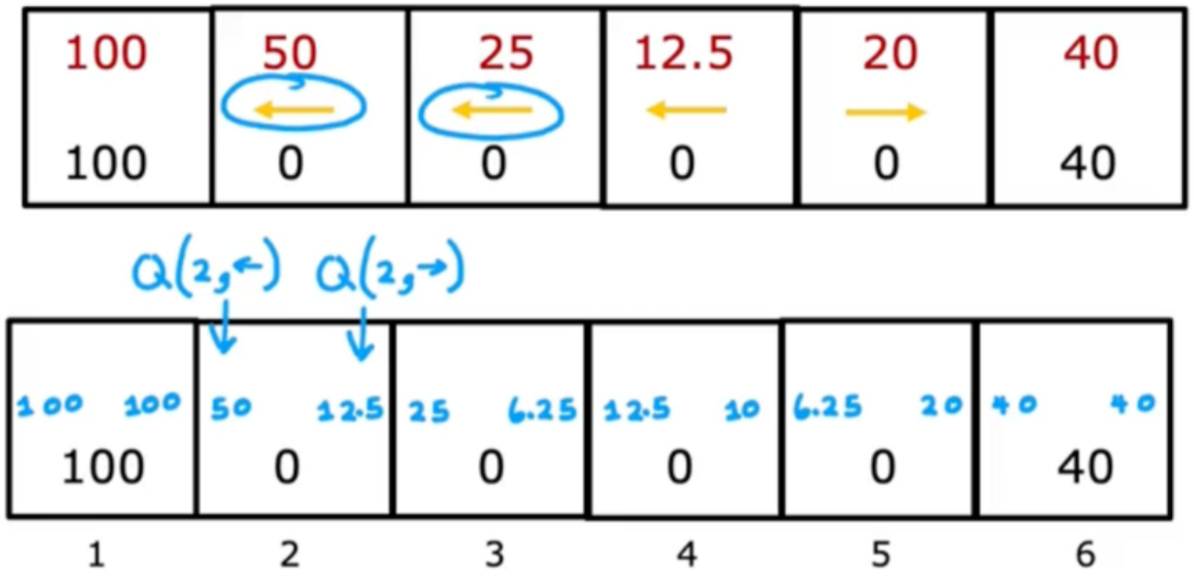
$\Rightarrow 50$

Will give a number that equals the return.

State action value function (Q- function)

$Q(s,a)$ = Return if you
- start in state s.
- take action a (once).
- then behave optimally after that.

$Q(s,a)$



← return
← action
← reward

$Q(2,\rightarrow) = 12.5$

$0+(0.5)0+(0.5)^2 0+(0.5)^3 100$

$Q(2,\leftarrow) = 50$

$0+(0.5)100$

$Q(4,\leftarrow) = 12.5$

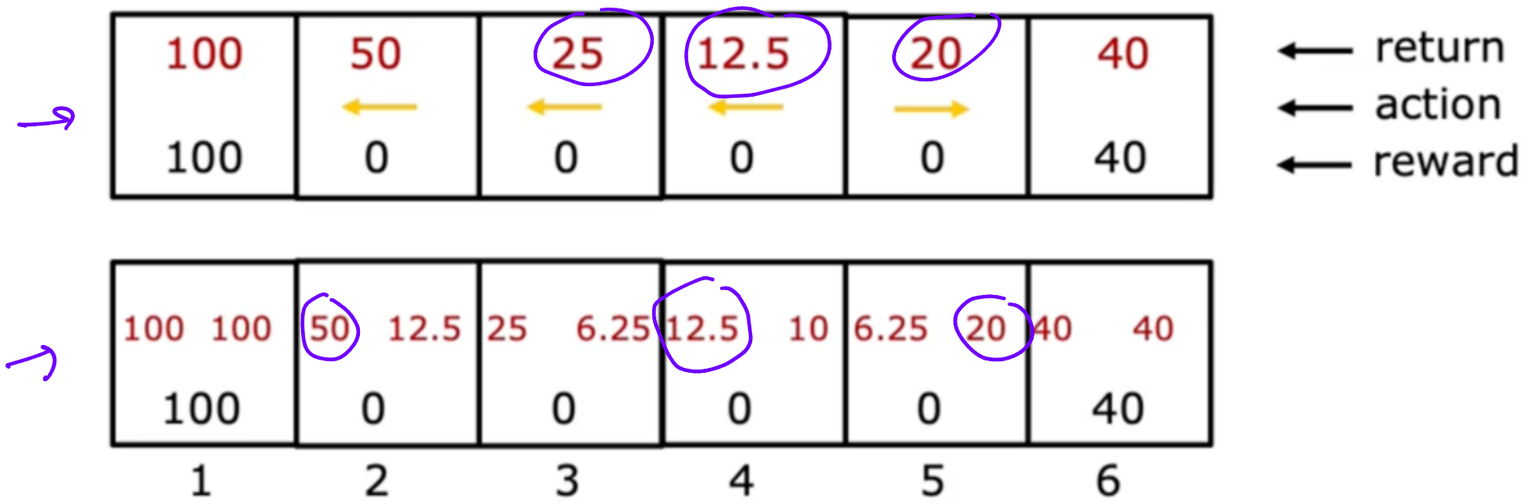
$0+(0.5)0+(0.5)^2 0+(0.5)^3 100$

So this is Q of s,a.

DeepLearning.AI Stanford ONLINE Andrew Ng

[Q^* (optimal Q function) .]

Picking actions



← return
← action
← reward

$\max_a Q(s,a)$

$\pi(s) = a$

The best possible return from state s is $\max_a Q(s,a)$.

$Q(s,a)$ = Return if you
- Start in state s
- take action a
- then behave optimally.

$Q(4, \leftarrow)$ 12.5

$Q(4, \rightarrow)$ 10

The best possible action in state is that action gives $\max_a Q(s,a)$

give you a way to pick actions as well.

DeepLearning.AI Stanford ONLINE Andrew Ng

Sure! This lecture is explaining the **State-Action Value Function**, commonly known as the **Q-function**, which is a fundamental concept in **Reinforcement Learning (RL)**. Let's break it down step by step.

---

1. What is the Q-function $Q(s, a)$?

The Q-function **estimates the return** (total future reward) if:

- You start in a given state s.
- You take a specific action a.
- Then, **you behave optimally** after that (i.e., always take the best actions to maximize future rewards).

So, $Q(s, a)$ tells us **how good it is** to take action a in state s, assuming we act optimally afterward.

---

2. Why does the definition feel circular?

The lecture acknowledges that the definition seems circular because:

- $Q(s, a)$ is based on **optimal behavior**.
- But if we already knew the **optimal policy** (the best action to take in every state), then we wouldn't need to compute $Q(s, a)$!

However, **RL algorithms solve this issue** by learning $Q(s, a)$ **without knowing the optimal policy upfront**. This will be made clear when we study Q-learning later.

---

3. Example Calculation of $Q(s, a)$

Scenario

- There's a **Mars Rover** navigating a series of states (locations).
- Each action (moving left or right) leads to different rewards.
- The **discount factor** γ (gamma) is **0.5**, meaning future rewards are worth **less** than immediate rewards.

Example 1: $Q(2, \text{right})$

- From **state 2**, if you go **right**, you reach **state 3**.
- From **state 3**, you behave **optimally**, which means moving left to eventually reach **state 1** (which gives 100 reward).
- The rewards along the way:
  - 0 at state 2
  - 0 at state 3
  - 0 at state 2 (looping back)
  - 100 at state 1

So, using the discount factor:

$$Q(2, \text{right}) = 0 + 0.5(0) + 0.5^2(0) + 0.5^3(100) = 12.5$$

Example 2: $Q(2, \text{left})$

- From **state 2**, if you go **left**, you reach **state 1**, which gives **100 reward**.
- The return calculation:

$$Q(2, \text{left}) = 0 + 0.5(100) = 50$$

So, **going left from state 2 is better** than going right because it gives a higher expected return.

---

4. General Pattern: How to Compute $Q(s, a)$

For any state s and action a:

$$Q(s, a) = R + \gamma \cdot Q(\text{next state}, \text{best next action})$$

where:

- R is the immediate reward.
- γ is the discount factor.
- $Q(\text{next state}, \text{best next action})$ is the highest Q-value from the next state.

This is a **recursive relationship** that helps us compute the Q-values step by step.

---

5. How to Choose the Best Action?

Once we compute $Q(s, a)$ for all possible actions in a state, we choose:

$$\pi(s) = \arg\max_a Q(s, a)$$

which means we **pick the action a that gives the highest Q-value**.

For example, in **state 4**:

- $Q(4, \text{left}) = 12.5$
- $Q(4, \text{right}) = 10$

Since **12.5 is greater than 10**, the optimal action in **state 4** is **going left**.

---

6. Why is the Q-function Important?

If we can compute $Q(s, a)$, then we can **always take the best action** in any state. This means:

1. **We can define the optimal policy $\pi(s)$ easily** (just take the action with the highest $Q(s, a)$).

6. Why is the Q-function Important?

If we can compute $Q(s, a)$, then we can **always take the best action** in any state. This means:

1. **We can define the optimal policy $\pi(s)$ easily** (just take the action with the highest $Q(s, a)$).
2. **Reinforcement learning algorithms** (like Q-learning) focus on estimating $Q(s, a)$ to find the best actions.

---

7. Alternative Notations

- The Q-function is often denoted as $Q^*(s, a)$ or **"optimal Q-function"** in RL literature.
- But for this lecture, just remember that $Q(s, a)$ **is the same as the "state-action value function"**.

Summary

- $Q(s, a)$ tells us how good it is to take action a in state s.
- It assumes **optimal behavior after the first action**.
- We compute it using a **recursive formula**.
- The **optimal policy** simply picks the action that maximizes $Q(s, a)$.
- The next step in RL is learning **how to estimate $Q(s, a)$ efficiently** without knowing the optimal policy in advance.

State Action Value Example

Run Code

State Action Value Function Example

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

In [1]:
```
import numpy as np
from utils import *
```

In [2]:
```
# Do not modify
num_states = 6
num_actions = 2
```

In [3]:
```
terminal_left_reward = 100
terminal_right_reward = 0
each_step_reward = 0

# Discount factor
gamma = 0.5

# Probability of going in the wrong direction
misstep_prob = 0
```

In [4]:
```
generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	50.0	25.0	12.5	20.0	40.0
100	← 0	← 0	← 0	→ 0	40

Q(s,a)

100.0 100.0	50.0 12.5	25.0 6.25	12.5 10.0	6.25 20.0	40.0 40.0
100	0	0	0	0	40

In [ ]:

smaller value says only 10.

Run Code

State Action Value Function Example

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

In [1]:
```
import numpy as np
from utils import *
```

In [2]:
```
# Do not modify
num_states = 6
num_actions = 2
```

In [7]:
```
terminal_left_reward = 100
terminal_right_reward = 40
each_step_reward = 0

# Discount factor
gamma = 0.9

# Probability of going in the wrong direction
misstep_prob = 0
```

In [8]:
```
generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	90.0	81.0	72.9	65.61	40.0
	←	←	←	←	
100	0	0	0	0	40

Q(s,a)

100.0 100.0	90.0 72.9	81.0 65.61	72.9 59.05	65.61 36.0	40.0
100	0	0	0	0	40

Implemented Q* Algorithm in this Code

Source: Reinforcement learning, Recommendations and Unsupervised learning Coursera Course.

Change the parameters like γ, states, gamma, discount factor.

Run Code

State Action Value Function Example

In this Jupyter notebook, you can modify the mars rover example to see how the values of Q(s,a) will change depending on the rewards and discount factor changing.

In [1]:
```
import numpy as np
from utils import *
```

In [2]:
```
# Do not modify
num_states = 6
num_actions = 2
```


In [9]:
```
terminal_left_reward = 100
terminal_right_reward = 40
each_step_reward = 0

# Discount factor
gamma = 0.3

# Probability of going in the wrong direction
misstep_prob = 0
```

In [10]:
```
generate_visualization(terminal_left_reward, terminal_right_reward, each_step_reward, gamma, misstep_prob)
```

Optimal policy

100.0	30.0	9.0	3.6	12.0	40.0
	←	←	→	→	
100	0	0	0	0	40

Q(s,a)

100.0 100.0	30.0 2.7	9.0 1.08	2.7 3.6	1.08 12.0	40.0 40.0
100	0	0	0	0	40

In [ ]: