Dqlab Reinforcement Learning

Reinforcement learning adalah tipe algoritma machine learning yang bisa membuat agent software dan mesin bekerja secara otomatis untuk menentukan perilaku yang ideal sehingga dapat memaksimalkan kinerja algoritmanya. DAlam beberapa tahun terakhir, penggunaan reinforcement learning terus meningkat, contohnya DeepMind and the Deep Q learning pada tahun 2014, AlphaGo di tahun 2016, dan OpenAI dan PPO di tahun 2017. Algoritma reinforcement learning didefinisikan sebagai metode machine learning yang berkaitan dengan cara software agent mengambil tindakan di environmentnya. Algoritma ini merupakan bagian dari metode deep learning yang akan memaksimalkan sebagian reward kumulatif.

Dalam algoritma reinforcement learning, ada beberapa istilah penting yang akan sering digunakan, yaitu agent, environment (e), reward (r), state (s), policy (Ï€), value (V), value function, model of the environment, model based methods, dan Q value atau action value (Q). Pada artikel kali ini, DQLab akan mengupas reinforcement learning mulai dari istilah penting, algoritmanya, hingga penggunaan reinforcement learning. Penasaran? Yuk simak artikelnya!

### **1. Istilah Penting Dalam Reinforcement Learning**

Seperti yang sudah dijelaskan di awal, algoritma reinforcement learning memiliki beberapa istilah penting yang akan selalu dipakai saat kita bekerja dengan algoritma ini. Agent adalah entitas yang diasumsikan melakukan tindakan di environment untuk mendapatkan beberapa reward, environment (e) adalah skenario yang harus dihadapi agen, reward (r) adalah pengembalian langsung yang diberikan kepada agen ketika dia melakukan tindakan atau tugas tertentu, state adalah keadaan yang mengacu pada situasi saat ini yang dikembalikan oleh environment, policy (Ï€) adalah strategi yang diterapkan oleh agent untuk memutuskan tindakan selanjutnya berdasarkan keadaan saat ini, value(V) adalah pengembalian jangka panjang, value function adalah fungsi yang menentukan nilai state yang merupakan jumlah total reward, model of environment merupakan model yang bertugas untuk menirukan keadaan lingkungan. Fungsi ini akan membuat kesimpulan dan menentukan bagaimana environment bekerja. Model based method merupakan metode pemecahan masalah reinforcement learning yang menggunakan metode berbasis model. Q value atau action value (Q) sangat mirip dengan value, satu-satunya perbedaan antara keduanya adalah Q value membutuhkan parameter tambahan untuk actionnya.

### **2. Algoritma, Karakter, dan Tipe dari Reinforcement Learning**

Ada tiga pendekatan yang bisa kita gunakan untuk mengimplementasikan algoritma Reinforcement Learning, yaitu value-based, policy-based, dan model-based. Pendekatan berdasarkan value-based kita harus mencoba memaksimalkan value function. Dalam metode ini, agent mengharapkan long-term return dari policy Ï€. Dalam metode reinforcement learning policy-based, kita mencoba menghasilkan policy sedemikian rupa sehingga tindakan yang dilakukan di setiap state dapat membantu mendapatkan reward maksimum. Sedangkan dalam algoritma reinforcement learning model-based, kita perlu membuat model virtual untuk setiap environment dan agent belajar untuk bekerja di environment  tersebut. Algoritma reinforcement learning memiliki beberapa karakteristik penting, antara lain algoritma ini tidak memerlukan supervisor, jadi hanya menggunakan bilangan real atau reward, pengambilan keputusan dengan algoritma ini dilakukan secara beruntun. Selain itu, dalam algoritma reinforcement, waktu sangat berperan penting dan feedback yang diterima selalu delay (tertunda).

Algoritma reinforcement learning memiliki dua tipe yaitu positif dan negatif. Reinforcement learning positif didefinisikan sebagai peristiwa yang akan terjadi karena perilaku tertentu. Algoritma ini akan meningkatkan kekuatan dan frekuensi yang akan berdampak positif pada tindakan yang akan diambil oleh agent. Algoritma reinforcement learning positif ini akan membantu memaksimalkan kinerja dan mempertahankan perubahan waktu. Namun, terlalu banyak reinforcement juga dapat menyebabkan pengoptimalan state yang berlebih sehingga dapat mempengaruhi hasil. Tipe algoritma reinforcement yang kedua adalah algoritma reinforcement negatif. Algoritma ini diartikan sebagai penguatan perilaku yang terjadi karena adanya kondisi negatif yang seharusnya dihentikan atau dihindari. Algoritma tipe ini membantu kita untuk menentukan standar kerja minimum.

### **3. Contoh Pengaplikasian Reinforcement Learning di Beberapa Sektor**

Contoh pertama penggunaan reinforcement learning adalah di sektor manufaktur. Beberapa perusahaan manufaktur menggunakan robot dengan reinforcement learning untuk mengambil barang dari satu tempat ke tempat lain. Robot ini akan dilatih untuk menghafal objek dan melakukan pekerjaan dengan kecepatan dan presisi yang tinggi. Selain itu robot-robot ini juga bisa digunakan untuk menyortir berjuta-juta produk di gudang supermarket atau e-commerce. Tujuan pemanfaatan robot ini adalah untuk menghindari human error sehingga produk tersebut dapat dikirim ke konsumen yang tepat. Pabrik tesla pun menggunakan lebih dari 160 robot yang bekerja untuk merakit mobil sehingga dapat mengurangi resiko cacat saat proses produksi.

Algoritma reinforcement learning juga dapat diaplikasikan pada power system. Reinforcement learning dan teknik pengoptimalan digunakan untuk menilai keamanan sistem tenaga listrik dan meningkatkan kinerja Microgrid. Metode adaptive learning digunakan untuk mengembangkan sistem pengontrol dan pelindung. Teknologi transmisi dengan perangkat High-Voltage Direct Current (HVDC) dan Flexible Alternating Current Transmission System devices (FACTS) berdasarkan reinforcement learning dapat membantu mengurangi transmisi dan emisi CO2 secara efektif. Reinforcement learning digunakan untuk mengembangkan struktur kontrol yang terdistribusi untuk satu set sumber pembangkit.

Selain di sektor manufaktur dan power system, reinforcement learning juga dapat digunakan di sektor keuangan. Perusahaan Pit.AI merupakan perusahaan pertama yang memanfaatkan reinforcement learning untuk mengevaluasi strategi perdagangan. Algoritma ini ternyata menjadi tool yang kuat pada sistem pelatihan untuk mengoptimalkan tujuan keuangan. Algoritma reinforcement learning ini memiliki peran yang sangat besar dalam perdagangan pasar saham karena algoritma Q-Learning (salah satu tipe reinforcement learning) dapat mempelajari strategi perdagangan yang optimal melalui satu instruksi sederhana dengan memaksimalkan nilai portofolio.

REINFORCEMENT LEARING

# Video : What is Reinforcement Learning

Welcome to this final week of the machine learning specialization.

It's a little bit bittersweet for me that we're approaching the end of this

specialization, but I'm looking forward to this week,

sharing with you some exciting ideas about reinforcement learning.

In machine learning, reinforcement learning is one of those ideas that while

not very widely applied in commercial applications yet today,

is one of the pillars of machine learning.

And has lots of exciting research backing it up and improving it every single day.

So let's start by taking a look at what is reinforcement learning.

Let's start with an example.

Here's a picture of an autonomous helicopter.

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

it's actually sitting in my office right now.

Like many other autonomous helicopters, it's instrumented with an onboard

computer, GPS, accelerometers, and gyroscopes and

the magnetic compass so it knows where it is at all times quite accurately.

And if I were to give you the keys to this helicopter and

ask you to write a program to fly it, how would you do so?

Radio controlled helicopters are controlled with joysticks like these and

so the task is ten times per second you're given the position and orientation and

speed and so on of the helicopter.

And you have to decide how to move these two control sticks in order to keep

the helicopter balanced in the air.

By the way,

I've flown radio controlled helicopters as well as quad rotor drones myself.

And radio controlled helicopters are actually quite a bit harder to fly,

quite a bit harder to keep balanced in the air.

So how do you write a program to do this automatically?

Let me show you a fun video of something we got

a Stanford autonomous helicopter to do.

Here's a video of it flying under the control of a reinforcement learning

algorithm.

And let me play the video.

I was actually the cameraman that day and

this is the helicopter flying on the computer control and

if I zoom out the video, you see the trees planted in the sky.

So using reinforcement learning,

we actually got this helicopter to learn to fly upside down.

We told it to fly upside down.

And so reinforced learning has been used to get helicopters to fly a wide

range of stunts or we call them aerobatic maneuvers.

By the way, if you're interested in seeing other videos,

you can also check them out at this URL.

So how do you get a helicopter to fly itself using reinforcement learning?

The task is given the position of the helicopter to decide how to

move the control sticks.

In reinforcement learning, we call the position and orientation and

speed and so on of the helicopter the state s.

And so the task is to find a function that maps from the state of the helicopter

to an action a, meaning how far to push the two control sticks in order

to keep the helicopter balanced in the air and flying and without crashing.

One way you could attempt this problem is to use supervised learning.

It turns out this is not a great approach for autonomous helicopter flying.

But you could say, well if we could get a bunch of observations of states and

maybe have an expert human pilot tell us what's the best action y to take.

You could then train a neural network using supervised learning to

directly learn the mapping from the states s which I'm calling x here,

to an action a which I'm calling the label y here.

But it turns out that when the helicopter is moving through the air is

actually very ambiguous, what is the exact one right action to take.

Do you tilt a bit to the left or a lot more to the left or

increase the helicopter stress a little bit or a lot?

It's actually very difficult to get a data set of x and the ideal action y.

So that's why for a lot of task of controlling a robot like a helicopter and

other robots, the supervised learning approach doesn't work well and

we instead use reinforcement learning.Now a key input to a reinforcement learning is something called the reward or

the reward function which tells the helicopter when it's doing well and

when it's doing poorly.

So the way I like to think of the reward function is a bit like training a dog.

When I was growing up, my family had a dog and

it was my job to train the dog or the puppy to behave.

So how do you get a puppy to behave well?

Well, you can't demonstrate that much to the puppy.

Instead you let it do its thing and whenever it does something good,

you go, good dog.

And whenever they did something bad, you go, bad dog.

And then hopefully it learns by itself how to do more of the good dog and

fewer of the bad dog things.

So training with the reinforcement learning algorithm is like that.

When the helicopter's flying well, you go, good helicopter and

if it does something bad like crash, you go, bad helicopter.

And then it's the reinforcement learning algorithm's job to figure out how to get

more of the good helicopter and fewer of the bad helicopter outcomes.

One way to think of why reinforcement learning is so

powerful is you have to tell it what to do rather than how to do it.

And specifying the reward function rather than the optimal action gives you a lot

So the way I like to think of the reward function is a bit like training a dog.

When I was growing up, my family had a dog and

it was my job to train the dog or the puppy to behave.

So how do you get a puppy to behave well?

Well, you can't demonstrate that much to the puppy.

Instead you let it do its thing and whenever it does something good,

you go, good dog.

And whenever they did something bad, you go, bad dog.

And then hopefully it learns by itself how to do more of the good dog and

fewer of the bad dog things.

So training with the reinforcement learning algorithm is like that.

When the helicopter's flying well, you go, good helicopter and

if it does something bad like crash, you go, bad helicopter.

And then it's the reinforcement learning algorithm's job to figure out how to get

more of the good helicopter and fewer of the bad helicopter outcomes.

One way to think of why reinforcement learning is so

powerful is you have to tell it what to do rather than how to do it.

And specifying the reward function rather than the optimal action gives you a lot

more flexibility in how you design the system.

Concretely for flying the helicopter, whenever it is flying well,

you may give it a reward of plus one every second it is flying well.

And maybe whenever it's flying poorly you may give it a negative reward or

if it ever crashes, you may give it a very large negative reward like negative 1,000.

And so this would incentivize the helicopter to spend a lot more

time flying well and hopefully to never crash.

But here's another fun video.

I was using the good dog bad dog analogy for reinforcement learning for many years.

And then one day I actually managed to get my hands on a robotic dog and

could actually use this reinforcement learning good dog bad dog

methodology to train a robot dog to get over obstacles.

So this is a video of a robot dog that using reinforcement learning,

which rewards it, moving toward the left of the screen has learned

how to place its feet carefully or climb over a variety of obstacles.

And if you think about what it takes to program a dog like this,

I have no idea, I really don't know how to tell it

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

All of these things were figured out automatically by the robot just by giving

it rewards that incentivizes it,

making progress toward the goal on the left of the screen.

Today, reinforcement learning has been successfully applied to a variety of

applications ranging from controlling robots.

And in fact later this week in the practice lab, you implement for yourself

a reinforcement learning algorithm to land a lunar lander in simulation.

It's also been used for factory optimization.

How do you rearrange things in the factory to maximize throughput and

efficiency as well as financial stock trading.

For example, one of my friends was working on efficient stock execution.

So if you decided to sell a million shares over the next several days, well,

you may not want to dump a million shares on the stock market suddenly because

that will move prices against you.

So what's the best way to sequence out your trades over time so that you can sell

the shares you want to sell and hopefully get the best possible price for them?

Finally, there have also been many applications of reinforcement

learning to playing games, everything from checkers to chess to the card

game of bridge to go as well as for playing many video games.

So that's reinforcement learning.

Even though reinforcement learning is not used nearly as much

as supervised learning, it is still used in a few applications today.

And the key idea is rather than you needing to tell the algorithm what is

the right output y for every single input, all you have to do instead is specify

a reward function that tells it when it's doing well and when it's doing poorly.

And it's the job of the algorithm to automatically figure out how to choose

good actions.

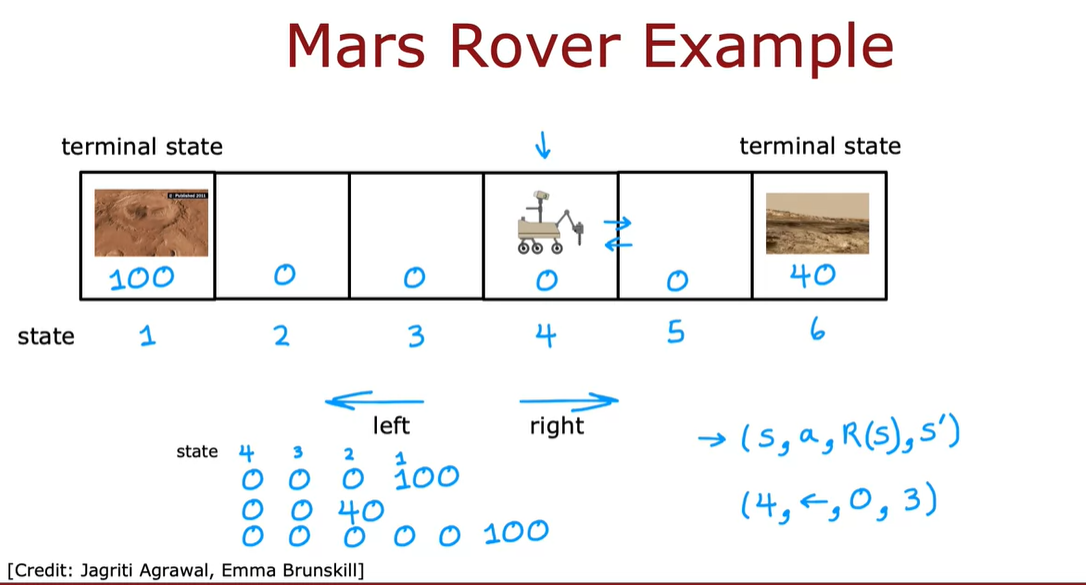
With that, let's now go into the next video where we'll formalize

the reinforcement learning problem and also start to develop algorithms for

automatically picking good actions



# Vide : Mars Rover Example



Let's take a look. We'll develop reinforcement learning

using a simplified example inspired by the Mars rover.

In this application, the rover can

be in any of six positions,

as shown by the six boxes here.

The rover, it might start off, say,

in disposition into fourth box shown here.

The position of the Mars rover is

called the state in reinforcement learning,

and I'm going to call these six states,

state 1, state 2,

state 3, state 4,

state 5, and state 6,

and so the rover is starting off in state 4.

Now the rover was sent to Mars to

try to carry out different science missions.

It can go to different places to

use its sensors such as a drill, or a radar,

or a spectrometer to

analyze the rock at different places on the planet,

or go to different places to take

interesting pictures for scientists on earth to look at.

In this example, state 1 here on the left has

a very interesting surface that

scientists would love for the rover to sample.

State 6 also has

a pretty interesting surface that

scientists would quite like the rover to sample,

but not as interesting as state 1.

We would more likely to carry out

the science mission ant state 1 than at state 6,

but state 1 is further away.

The way we will reflect state 1 being

potentially more valuable is through the reward function.

The reward at state 1 is a 100,

and the reward at stage 6 is 40,

and the rewards at all of the other states in-between,

I'm going to write as a reward of zero because there's

not as much interesting science to

be done at these states 2,

3, 4, and 5.

On each step, the rover gets

to choose one of two actions.

It can either go to the left or it can go to the right.

The question is, what should the rover do?

In reinforcement learning, we pay

a lot of attention to the rewards

because that's how we know if

the robot is doing well or poorly.

Let's look at some examples of what might

happen if the robot was to go left,

starting from state 4.

Then initially starting from state 4,

it will receive a reward of zero,

and after going left,

it gets to state 3,

where it receives again a reward of zero.

Then it gets to state 2,

receives the reward is 0,

and finally just to state 1,

where it receives a reward of 100.

For this application, I'm going to assume that

when it gets either state 1 or state 6,

that the day ends.

In reinforcement learning, we sometimes call this

a terminal state,

and what that means is that,

after it gets to one of these terminals states,

gets a reward at that state,

but then nothing happens after that.

Maybe the robots run out of

fuel or ran out of time for the day,

which is why it only gets to either enjoy

the 100 or the 40 reward,

but then that's it for the day.

It doesn't get to earn additional rewards after that.

Now instead of going left,

the robot could also choose to go to the right,

in which case from state 4,

it would first have a reward of zero,

and then it'll move right and get to state 5,

have another reward of zero,

and then it will get to this

other terminal state on the right,

state 6 and get a reward of 40.

But going left and going right are the only options.

One thing the robot could do is it can start from

state 4 and decide to move to the right.

It goes from state 4-5,

gets a reward of zero in state

4 and a reward of zero in state 5,

and then maybe it changes its mind and decides to start

going to the left as follows, in which case,

it will get a reward of zero at state 4, at state 3,

at state 2, and then the reward

of 100 when it gets to state 1.

In this sequence of actions and states,

the robot is wasting better time.

So this maybe isn't such a great way to take actions,

but it is one choice that the algorithm could pick,

but hopefully you won't pick this one.

To summarize, at every time step,

the robot is in some state,

which I'll call S,

and it gets to choose an action,

and it also enjoys some rewards,

R of S that it gets from that state.

As a result of this action,

it to some new state S prime.

As a concrete example,

when the robot was in state 4 and it took the action,

go left, maybe didn't enjoy the reward of

zero associated with that state

4 and it won't have any new state 3.

When you learn about

specific reinforcement learning algorithms,

you see that these four things,

the state, action, the reward and next state,

which is what happens basically every

time you take an action that just

be a core elements of

what reinforcement learning algorithms will

look at when deciding how to take actions.

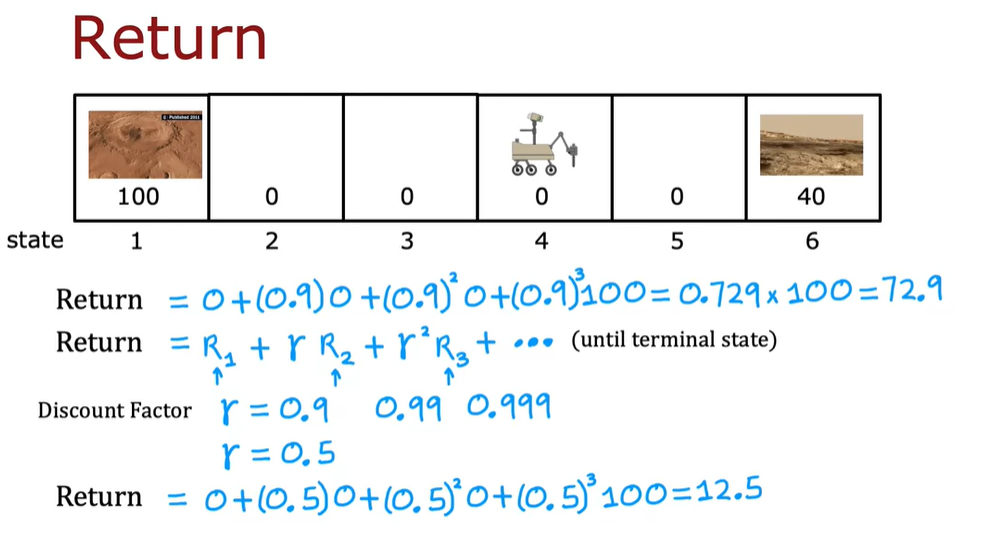
Just for clarity, the reward here,

R of S, this is the reward associated with this state.

This reward of zero is associated

with state 4 rather than with state 3.

# Video The Return in Reinforcement Learning



You saw in the last video,

what are the states of

reinforcement learning application,

as well as how depending on the actions

you take you go through different states,

and also get to enjoy different rewards.

But how do you know if a particular set of rewards is

better or worse than a different set of rewards?

The return in reinforcement learning,

which we'll define in this video,

allows us to capture that.

As we go through this,

one analogy that you might find helpful

is if you imagine

you have a five-dollar bill at your feet,

you can reach down and pick up,

or half an hour across town,

you can walk half an hour and pick up a 10-dollar bill.

Which one would you rather go after?

Ten dollars is much better than five dollars,

but if you need to walk for half an hour

to go and get that 10-dollar bill,

then maybe it'd be more

convenient to just pick up the five-dollar bill instead.

The concept of a return captures that rewards you can get

quicker are maybe more attractive than

rewards that take you a long time to get to.

Let's take a look at exactly how that works.

Here's a Mars Rover example.

If starting from state 4 you go to the left,

we saw that the rewards you get would

be zero on the first step from state 4,

zero from state 3,

zero from state 2,

and then 100 at state 1, the terminal state.

The return is defined as the sum

of these rewards but weighted by one additional factor,

which is called the discount factor.

The discount factor is a number a little bit less than 1.

Let me pick 0.9 as the discount factor.

I'm going to weight the reward

in the first step is just zero,

the reward in the second step is a discount factor,

0.9 times that reward,

and then plus the discount factor^2 times that reward,

and then plus the discount factor^3 times that reward.

If you calculate this out,

this turns out to be 0.729 times 100, which is 72.9.

The more general formula for the return is that if

your robot goes through some sequence of

states and gets reward R\_1 on the first step,

and R\_2 on the second step,

and R\_3 on the third step,

and so on,

then the return is R\_1 plus the discount factor Gamma,

this Greek alphabet Gamma

which I've set to 0.9 in this example,

the Gamma times R\_2

plus Gamma^2 times R\_3 plus Gamma^3 times R\_4,

and so on, until you get to the terminal state.

What the discount factor Gamma does is it has

the effect of making

the reinforcement learning algorithm

a little bit impatient.

Because the return gives full credit to

the first reward is 100 percent is 1 times R\_1,

but then it gives a little bit less credit to

the reward you get at the second step

is multiplied by 0.9,

and then even less credit to the reward you

get at the next time step R\_3,

and so on, and so getting rewards

sooner results in a higher value for the total return.

In many reinforcement learning algorithms,

a common choice for

the discount factor will be a number pretty close to 1,

like 0.9, or 0.99, or even 0.999.

But for illustrative purposes

in the running example I'm going to use,

I'm actually going to use a discount factor of 0.5.

This very heavily down weights or

very heavily we say discounts rewards in the future,

because with every additional parsing timestamp,

you get only half as much credit as

rewards that you would have gotten one step earlier.

If Gamma were equal to 0.5,

the return under the example above would have been

0 plus 0.5 times 0,

replacing this equation on top,

plus 0.5^2 0 plus 0.5^3 times 100.

That's lost reward because state 1 to terminal state,

and this turns out to be a return of 12.5.

In financial applications, the discount factor also has

a very natural interpretation as

the interest rate or the time value of money.

If you can have a dollar today,

that may be worth a little bit more

than if you could only get a dollar in the future.

Because even a dollar today you can put in the bank,

earn some interest, and end up with

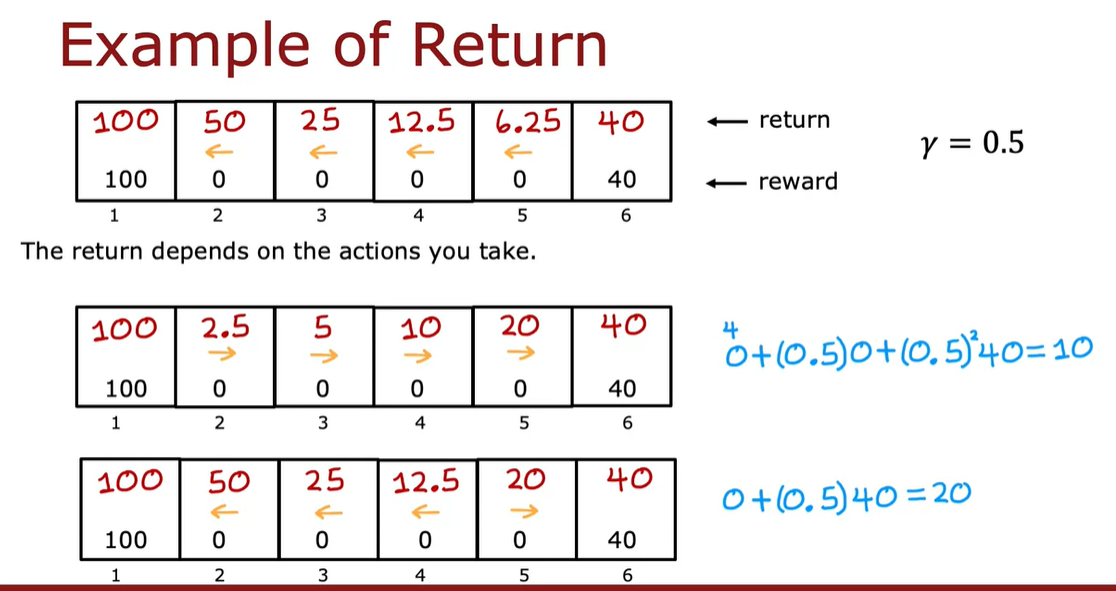
a little bit more money a year from now.

For financial applications, often,

that discount factor represents how much less

is a dollar in the future where

I've compared to a dollar today.



Let's look at some concrete examples of returns.

The return you get depends on the rewards,

and the rewards depends on the actions you take,

and so the return depends on the actions you take.

Let's use our usual example and say for this example,

I'm going to always go to the left.

We already saw previously that if

the robot were to start off in state 4,

the return is 12.5 as

we worked out on the previous slide.

It turns out that if it were to start off in say three,

the return would be 25 because it gets

to the 100 reward one step sooner,

and so it's discounted less.

If it were to start off in state 2,

the return would be 50.

If it were to just start off and state 1, well,

it gets the reward of 100 right away,

so it's not discounted low.

The return if we were to start out in

state 1 will be 100,

and then the return in these two states are 6.25.

It turns out if you start off in state 6,

which is terminal state,

you just get the reward and thus the return of 40.

Now, if you were to take a different set of actions,

the returns would actually be different.

For example, if we were to always go to the right,

if those were our actions,

then if you were to start in state 4, get a reward of 0.

Then you get to state 5, get a reward of 0,

and it gets to state 6,

and get a reward of 40.

In this case, the return would be 0 plus 0.5,

the discount factor times 0 plus 0.5 squared times 40,

and that turns out to be equal to 0.5 squared is 1/4,

so 1/4 of 40 is 10.

The return from this state,

from state 4 is 10.

If you were to take actions,

always go to the right.

Through similar reasoning,

the return from this state is 20,

the return from this state is five,

the return from this state is 2.5,

and then the return,

the determinant state is is 140.

By the way,

if these numbers don't fully make sense,

feel free to pause the video and

double-check the math and see if you

can convince yourself that these

are the appropriate values for the return.

For if you start from different states,

and if you were to always go to the right.

We see that it would always go to the right.

The return you expect to get is lower for most states.

Maybe always going to the right isn't

as good an idea as always going to the left.

But it turns out that we

don't have to always go to the left,

always go to the right.

We could also decide if you're in state 2, go left.

If your in state 3, go left.

If you're in state 4, go left.

But if you're in state 5,

then you're so close to this reward.

Let's go right.

This will be a different way of choosing

actions to take based on what state you're in.

It turns out that the return

you get from the different states will be 100, 50, 25,

12.5, 20, and 40.

Just to illustrate one case.

If you were to start off in state 5,

here you would go to the right,

and so the rewards you get would be

zero first in state 5, and then 4.

The return is zero, the first reward,

plus the discount factor is 0.5 times 40, which is 20,

which is why the return from this status

20 if you take actions shown here.

To summarize, the return in

reinforcement learning is the sum

of the rewards that the system gets,

weighted by the discount factor,

where rewards in the far future are weighted

by the discount factor raised to a higher power.

Now, this actually has

an interesting effect when you have

systems with negative rewards.

In the example we went through,

all the rewards were zero or positive.

But if there are any rewards are negative,

then the discount factor actually

incentivizes the system to

push out the negative rewards

as far into the future as possible.

Taking a financial example,

if you had to pay someone $10,

maybe that's a negative reward of minus 10.

But if you could postpone payment by a few years,

then you're actually better off

because $10 a few years from now,

because of the interest rate is actually worth

less than $10 that you had to pay today.

For systems with negative rewards,

it causes the algorithm to try to push

out the make the rewards

as far into the future as possible.

For financial applications and for other applications,

that actually turns out

to be right thing for the system to do.

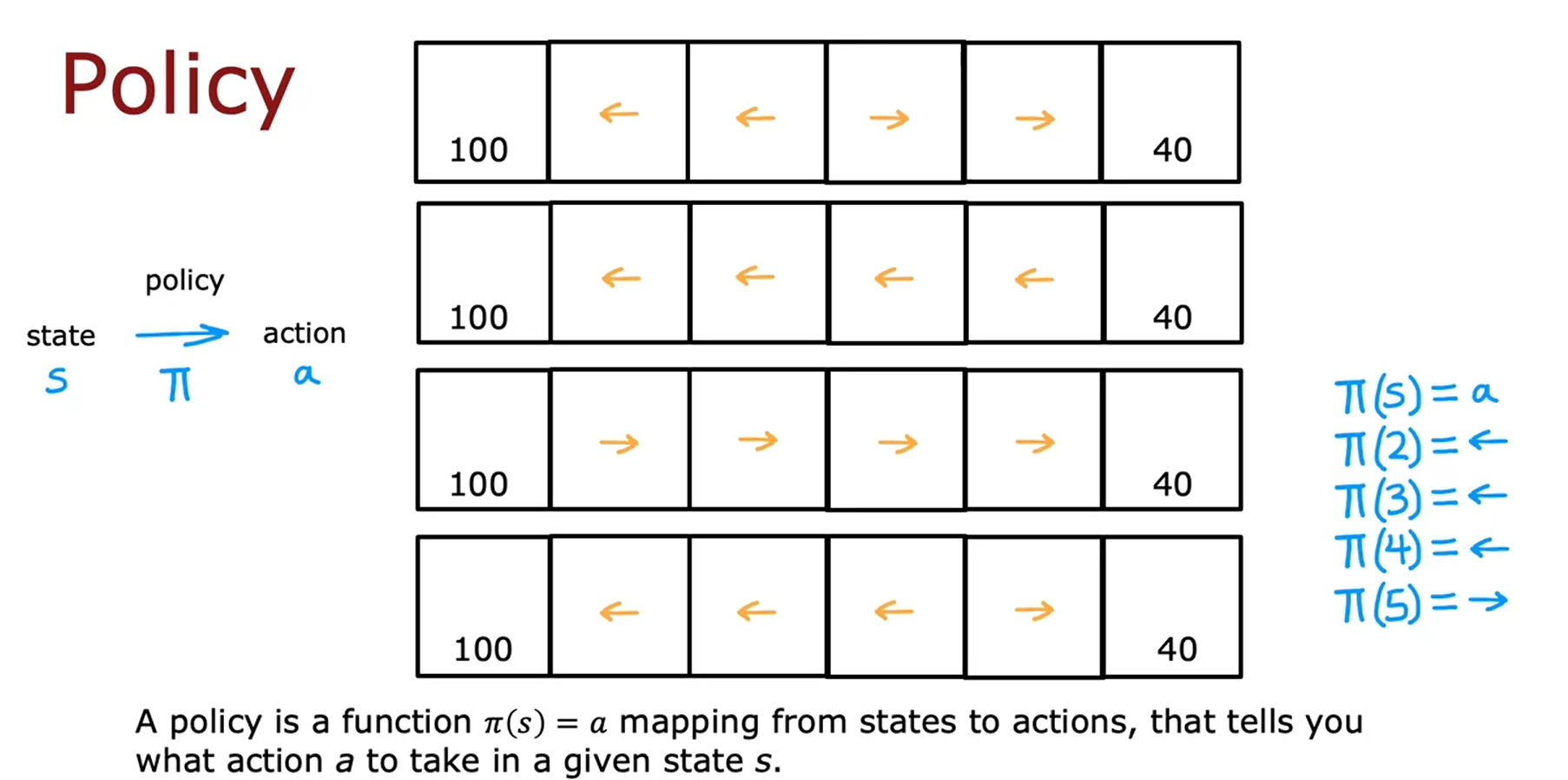
You now know what

is the return in reinforcement learning,

let's go on to the next video to

formalize the goal of reinforcement learning algorithm.

# Video : Making decisions: Policies in reinforcement learning



Let's formalize how

a reinforcement learning algorithm picks actions.

In this video, you'll learn about what is

a policy in reinforcement learning algorithm.

Let's take a look.

As we've seen, there are many different ways that you

can take actions in the reinforcement learning problem.

For example, we could decide

to always go for the nearer reward,

so you go left if this leftmost reward is

nearer or go right if this rightmost reward is nearer.

Another way we could choose actions is to always go for

the larger reward or

we could always go for smaller reward,

doesn't seem like a good idea,

but it is another option,

or you could choose to go left

unless you're just one step away from the lesser reward,

in which case, you go for that one.

In reinforcement learning, our goal is to come up with

a function which is called a policy Pi,

whose job it is to take as input

any state s and map it

to some action a that it wants us to take.

For example, for this policy here at the bottom,

this policy would say that if you're in state 2,

then it maps us to the left action.

If you're in state 3,

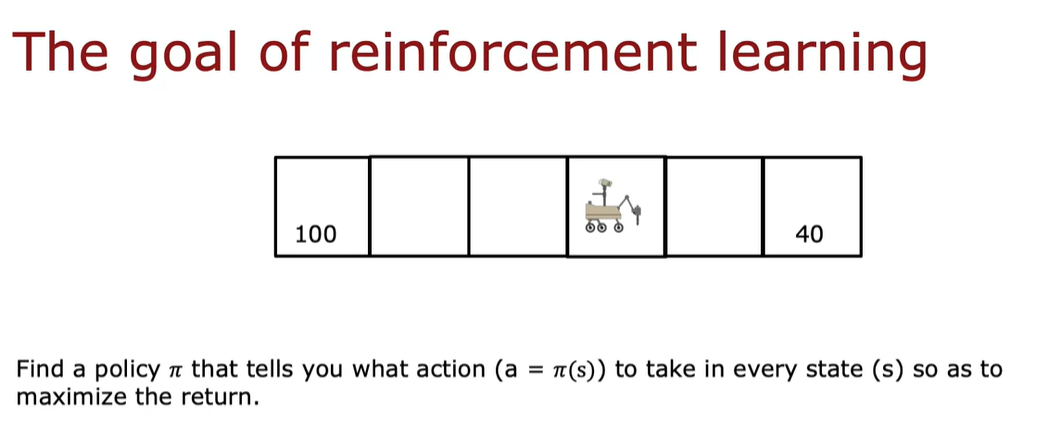
the policy says go left.

If you are in state 4 also go

left and if you're in state 5, go right.

Pi applied to state S,

tells us what action it wants us to take in that state.



The goal of reinforcement learning is to find a policy Pi

or Pi of S that tells you what action to

take in every state so as to maximize the return.

By the way, I don't know if policy is

the most descriptive term of what pi is,

but it's one of those terms that's

become standard in reinforcement learning.

Maybe calling Pi a

controller rather than a policy would be

more natural terminology but policy

is what everyone in

reinforcement learning now calls this.

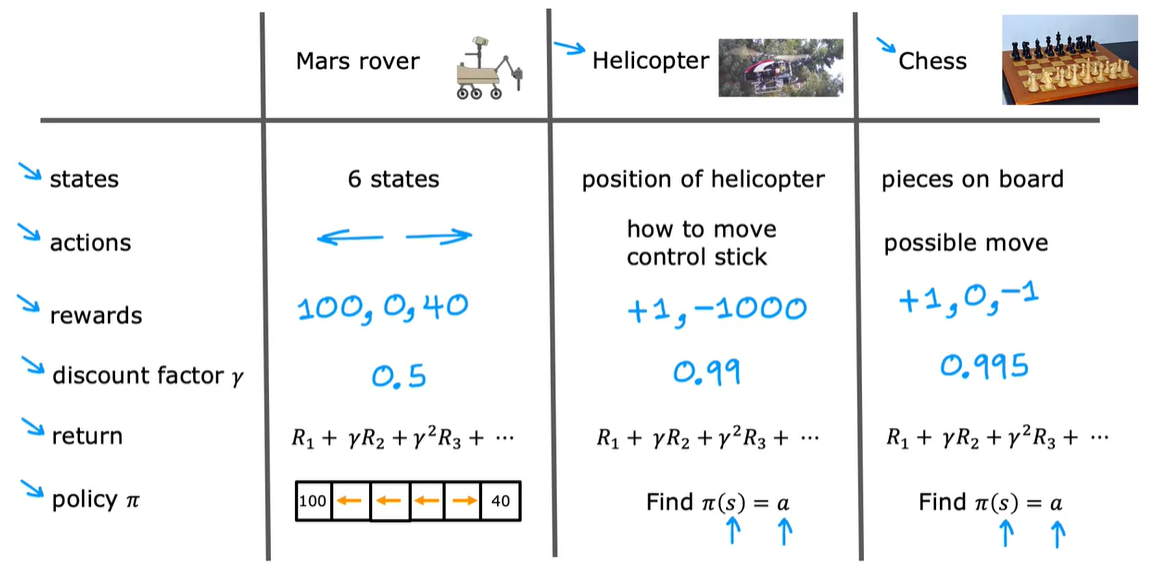
In the last video, we've gone

through quite a few concepts in

reinforcement learning from states to actions to reward,

to returns, to policies

# Video : Review of key concepts



We've developed a reinforcement learning formalism

using the six state Mars rover example.

Let's do a quick review of the key concepts and also see

how this set of concepts can be

used for other applications as well.

Some of the concepts we've discussed

are states of a reinforcement learning problem,

the set of actions, the rewards,

a discount factor, then how rewards and

the discount factor altogether

use to compute the return,

and then finally, a policy whose job it

is to help you pick actions so as to maximize the return.

For the Mars rover example,

we had six states that we numbered

1-6 and the actions were to go left or to go right.

The rewards were 100 for the leftmost state,

40 for the rightmost state,

and zero in between

and I was using a discount factor of 0.5.

The return was given by this formula and we could have

different policies Pi depict

actions depending on what state you're in.

This same formalism or states, actions,

rewards, and so on can be used

for many other applications as well.

Take the problem or find an autonomous helicopter.

To set a state would be the set of

possible positions and orientations and

speeds and so on of the helicopter.

The possible actions would be the set of

possible ways to move

the controls stick of a helicopter,

and the rewards may be a plus one if it's flying well,

and a negative 1,000

if it doesn't fall really bad or crashes.

Reward function that tells you

how well the helicopter is flying.

The discount factor, a number

slightly less than one maybe say,

0.99 and then based

on the rewards and the discount factor,

you compute the return using the same formula.

The job of a reinforcement learning

algorithm would be to find

some policy Pi of s so that given as input,

the position of the helicopter s,

it tells you what action to take.

That is, tells you how to move the control sticks.

Here's one more example.

Here's a game-playing one.

Say you want to use

reinforcement learning to learn to play chess.

The state of this problem would

be the position of all the pieces on the board.

By the way, if you play chess and know the rules well,

I know that's little bit more information

than just the position of

the pieces is important for chess,

but I'll simplify it a little bit for this video.

The actions are the possible legal moves in the game,

and then a common choice of reward would be if you

give your system a reward of plus one if it wins a game,

minus one if it loses the game,

and a reward of zero if it ties a game.

For chess, usually a discount factor

very close to one will be used,

so maybe 0.99 or even 0.995 or

0.999 and the return

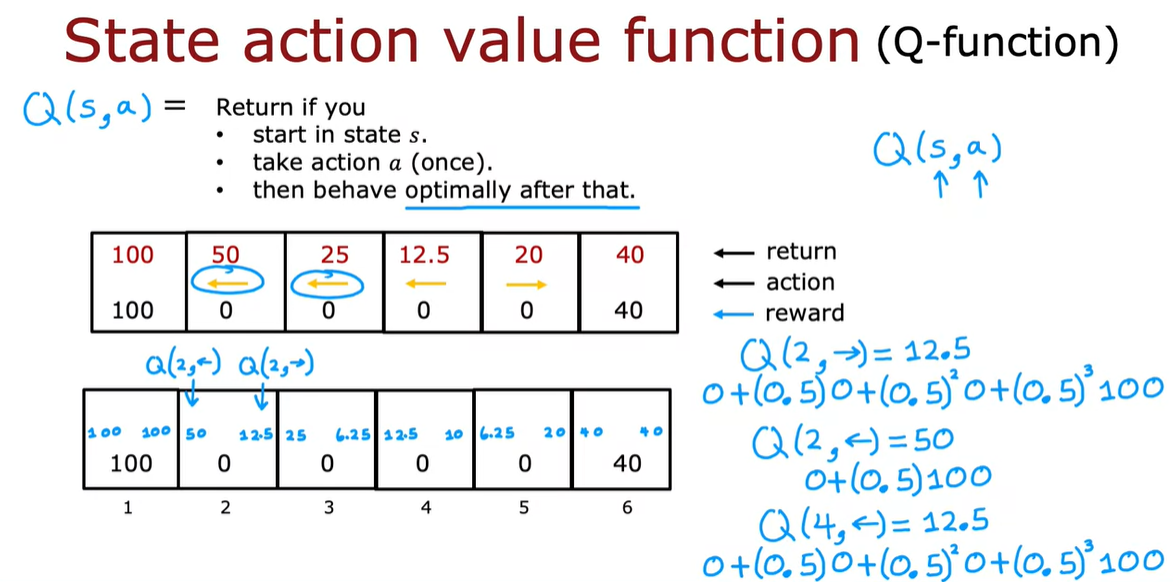
uses the same formula as the other applications.

Once again, the goal is given

a board position to pick a good action using a policy Pi.

########################### STATE ACTION VALUE FUNCTION #######################

# VIDEO : State Action value function



when we start to develop reinforcement learning hours later this week,

you see that there's a key quantity that reinforcement learning arrows will try to

compute and that's called the state action value function.

Let's take a look at what this function is.

The state action value function is a function typically

denoted by the letter uppercase Q.

And it's a function of a state you might be in as well as

the action you might choose to take in that state and QFSA.

Will give a number that equals the return.

If you start in that state.

S and take the action A just once and after taking

action A once you then behave optimally after that.

So after that you take whatever actions will result in the highest possible

return.

Now you might be thinking there's something a little bit strange about this

definition because how do we know what is the optimal behavior?

And if we knew what the auto behavior, if we already knew what's the best action to

take in every state, why do we still need to compute Q of SA.

Because we already have the auto policy.

So I do want to acknowledge that there's something a little bit strange about this

definition.

There's almost something a little bit circular about this definition, but

rest assured When we look at specific reinforcement learning outcomes later will

resolve this slightly circular definition and will come up with a way to compute

the Q function even before we've come up with the optimal policy.

But you see that in a later video.

So don't worry about this for now.

Let's look at an example we saw previously that this is a pretty

good policy Go left from stage 2, 3 and four and go right from State five.

It turns out that this is actually the optimal policy for

the mars rover application When the discount factor gamma is 0.5, so Q of S.

A will be equal to the total return If you start from say that

take the action A and then behave optimally after that.

Meaning take actions according to this policy.

Shown over here, let's figure out what Q of s,a.

Is for a few different states.

Let's look at say Q of state too.

And what if we take the action to go right well if you're in state two and

you go right then you end up at state three And

then after that you behave optimally you're going to go left from ST three and

then go left from state to and then eventually you get the reward of 100.

In this case, the rewards you get would be zero from state to zero

when you get to stay three zero when you get back to state two and

then 100 when you finally get to the terminal state one and

so the return will be zero plus 0.5 times that plus 0.5

squared times ac plus 0.5 cubed times 100.

And this turns out to be 12.5 And so

Q of ST two of going right as equal to 12.5.

Note that this passes, no judgment on whether going right is a good idea or not.

It's actually not that good an idea from state to to go right, but

it just faithfully reports out the return if you take action A and

then behave optimally afterwards.

Here's another example.

If you're in state to and you were to go left, then the sequence

of rewards you get will be zero when you're in state two followed by 100.

And so the return is zero plus 0.5 times 100

that's equal to 50 in order to write down The values of QSA.

In this diagram, I'm going to write 12.5 here on the right

to denote that this is Q of state to going to the right.

And then when I write a little 50 here on the left to denote

that this is Q of ST two and going to the left just to take one more

example What if we're in ST four and we decide to go left.

Well if you're in stage four you go left,

you get rewards zero and then you take action left here.

So zero gain, take action left here, zero and then 100.

So Q of four Left results in rewards zero because the first action is left and

then because we followed the optimal policy afterwards You can reward 00 100.

And so the return is zero plus 00.5 times that.

Plus 4.5 squared times that plus 0.5 Q times that.

Which is therefore equal to 12.5.

So Q4 left is 12.5.

I'm going to write this year as 12.5.

And it turns out if you were to carry out this exercise for all of the other

states and all of the other actions, you end up with this being the Q of s,a.

For different states and different actions And then finally at the Terminal State.

Well it doesn't matter what you do, you just get that terminal reward 100 or 40.

So just write down those terminal awards over here.

So this is Q of s,a.

For every state state one through six and for the two actions,

action left and action right.

Because the state action value function is almost always noted by the letter Q.

This is also often called the Q function.

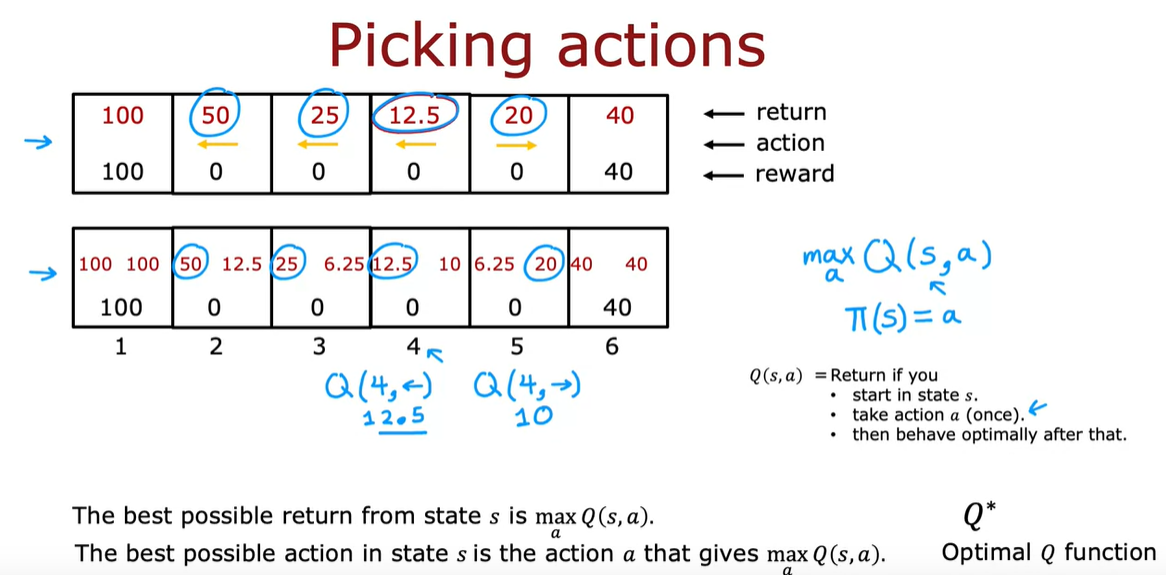
So the terms Q.

Function and state action value function are used interchangeably and

it tells you what are your returns or really what is the value?

How good is it?

Just take action A and ST S and then behave optimally after that.



Now it turns out that once you can compute the Q function this will

give you a way to pick actions as well.

Here's the policy and return.

And here are the values two of s,a.

From the previous slide.

You notice one interesting thing when you look at the different states which

is that if you take state to taking the action left results in a,q.

Value or state action value of 50 which is actually the best possible return you can

get from that state.

In state three two of s,a.

for the action left also gives you that higher return

is therefore the action left gives you the return you want.

And in state five is actually the action going to the right that

gives you that higher return of 20.

So it turns out that the best possible return from any state S.

Is the largest value of Q,F, S.

A maximizing over A.

Just to make sure this is clear what I'm saying is that in say stay for

There is two of state four left which is 12.5 And q.

of state four right, Which turns out to be 10.

And the larger of these two values which is 12.5 Is

the best possible return from that state four.

In other words the highest return you can hope to get from State four is 12.5.

And it's actually the larger of these two numbers 12.5 and 10.

And moreover, if you want your Mars Rover to enjoy a return of 12.5

rather than say 10 then the action you should take is the action A.

That gives you the larger value of Q of s,a.

So the best possible action status is the action A.

That actually maximizes Q, of s,a.

So this might give you a hint for why computing Q, of s,a.

Is an important part of the reinforcement learning algorithm that will build later.

Namely if you have a way of computing Q of s,a.

For every state and for every action then when you're in

some state s all you have to do is look at the different actions A.

And pick the action A.

That maximizes Q of s,a.

And so pi F.

S can just pick the action A.

That gives the largest value of Q of s,a.

And that will turn out to be a good action.

In fact it turned out to be the optimal action.

Another intuition about why this makes sense is Qof s,a.

Is returned if you sudden status and take the action A.

And then behave optimally after that.

So in order to earn the biggest possible return,

what you really want is to take the action A.

That results in the biggest total return.

That's why if only we have a way of computing Q f s,a.

For every state taking the action aid that maximizes return under

these circumstances seems like it's the best action to take in that state.

Although this isn't something you need to know for this.

Cause I want to mention also that if you look online or look at the reinforcement

learning literature, sometimes you also see this Q function written as Q.

Star instead of Q.

And this Q function is sometimes also called the optimal Q function.

These terms just refer to the Q function exactly as we've defined it.

So if you look at the reinforcement learning literature and read about Q.

Star or the Q function,

that just means the state action value function that we've been talking about.

But for the purposes of this course you don't need to worry about this.

So to summarize if you can compute Q of s,a.

For every state and every action,

then that gives us a good way to compute the auto policy pi of S.

So that's the state action value function or the Q function.

We'll talk later about how to come up with an algorithm to compute them despite

the slightly circular aspect of the definition of the Q function.

But first let's take a look at the next video at some specific examples of what

these values Q of s,a.

Actually look like

# Video : State-action value function