In this video, we'll provide a high level

introduction to reinforcement learning. Now let's go over the learning goals for

this section. In this section, we're going to cover

an overview of reinforcement learning at a very high level. We'll have a discussion about

the understanding of the approaches and implementation for reinforcement learning. Then finally we'll introduce reinforcement

learning implementation using Python. So let's start off with reinforcement

learning overview as we promised. Now, in reinforcement learning, the idea is that agents will be

interacting with an environment. And agents then are going to be the thing

that ultimately takes the action. So if you think in regards to

games which are very popular for reinforcement learning,

currently this would be the actual player. If we were thinking through a model that

was meant to figure out, for example, where to place ads on a web page,

the agent would just be that program that makes the decision where

the ad will be placed. And the environment is going to be

the world through which our agent moves. So if you're playing games such as chess,

this would be the actual chess board for thinking something like ads on a web page,

this would be the entire web page. And they choose from a set of available

actions, and again using the game example, this would be all possible moves that

an agent or a player can make in our game. In our ads example, this may be either

adding an ad, removing an ad, or taking option of neither removing or

adding an ad from the current page. And the actions that we take

are going to impact the environment, which in turn impacts

the agents via rewards. So when an action is taken, we have impacted the environment

where our agent exists. So if you think if we

move a piece in our game, we have adjusted

the environment of our game. If our move resulted in us getting

more points or winning the game, this would be an example of a reward. And our system would learn that

the actions taken were good actions. And similarly for ads example, our reward could be a result in

increase clicks or increase in revenue. Now something to note is that

rewards are generally unknown and must be estimated by the agent, so

oftentimes it will take many steps to reach towards that reward

stage of your game. If that's just to win the game more to

get to a certain place within the game. So I think for any game of any kind, oftentimes they'll take multiple moves

before you get any type of reward. And this process again repeat dynamically. So agents continuously learn how

to estimate rewards overtime. Now, advances in deep

learning have led to many recent reinforcement

learning developments. For example in 2013 researchers

from DeepMind developed the system to play Atari games and

actually beat humans in Atari games. And in 2017 the AlphaGo system

defeated the world champion in Go. So for the first time, the machines

were able to be a human champion in a complex game such as Go

using reinforcement learning. Now, in general, reinforcement learning

algorithms have been limited due to significant data and

computational requirements. So if you think about the infinite number

of possibilities at every juncture, if you're adjusting for every person

that visits your site, or even for games which is the example that

has proven to be successful. But the reason why it's taken so long is

that you think about something like Go or chess, the infinite amounts of

moves that anyone can make along with the following moving reaction to

those moves may lead to us needing a lot of data to train our

reinforcement learning models. Now more recently, progresses has been made in areas with

more direct business applications. And examples include recommendation

engines where recommending correctly could perhaps be a reward. Marketing with higher revenues or

higher clicks, again, being that reward mechanism,

and automated bidding. If you're able to optimize the amount

spent or paid per an item, and setting up some reward system

in that sense as well. Now the idea here is

that the agents again, if we think about reinforcement learning,

the agent take some action. That action effects the current

environment, and then feedback from that environment is passed back to

the agent in terms of a reward. So if it resulted in a positive result

in relation to our reward system, the agents actions are then reinforced. And then vice versa for negative results

if it ended up in a bad state and the agent is reinforced not

to take those same steps. Now reinforcement learning

problems will vary significantly. And solutions represent a policy

by which agents choose actions in response to the current state,

or in other words, since this is not directly supervised

learning, what takes our input? And comes up with the resulting action

is the policy, and that is what we ultimately try and optimize,

whatever that policy is defined as. And agents typically work to

maximize expected rewards overtime. And this differs from typical machine

learning problems because unlike with labels, rewards are not known and

often highly uncertain. We may not know at every juncture weather

actions resulted in immediate rewards, or even if it did, if those intermediate rewards will lead

to our larger goals of our network. Whereas with typical machine learning

problems the solutions remain static, with reinforcement learning as actions

impact the environment, the state changes, which continuously changes

the problem that we're working with. And then finally, agents face a trade off

between rewards in different periods, again, pointing to this uncertainty

that revolves around this reward system. Now just a quick introduction. We will get into Notebook, but

in Python the most common library for reinforcement learning is

going to be OpenAI GYM. So we're going to want to import our

GYM library to create an environment. We call gym.make, and there are actually

some environments that are going to be available to us according to

the strings that we pass and we'll see this in the Notebook, so

that we can specify the game or environments that world

in which we are living. And then about render, now that we've

created that environment object, we'll show the current

state of our environment. Now just to recap, in this section we

discuss reinforcement learning overview with an understanding of that

feedback loop, where the goal is for an agent to interact with

the environment to choose from a set of available actions to

increase possible rewards. And those rewards lead to reinforcement

of those actions within the environment. And we discussed how solution approaches

to reinforcement learning relied on the policy by which agents chose

actions in response to the given state. And as those actions

impact the environment, the state changes which changes

the problem we are currently working with. Then finally we closed out with

a quick introduction to reinforcement learning implementation in Python, which we're going to go into

further in our final notebook.

Welcome to our notebook here

on reinforcement learning. In this notebook, we're going to use the

library open AI gym, which you can install using pip install or following

the instructions that we have linked here. Some key concepts that we should be aware

of before we get started with AI gym are the idea of an observation. And that's going to be the current state

of the game and that will describe where your agent currently is within that

environment within the world of the game. There's going to be actions, and those

are the different moves that the agent can make, and

we discussed that in lecture. They're going to be this

idea of an episode. And one full game played were

just playing games here. One full game played from the beginning,

which we initiate with environment, not reset until the end where done equals

true is going to represent a single episode and will see this clearly as

we walk through the actual examples. And then the step is the part of

the game that includes one action. So that's just one specific

action in the game. And transitions from one observation where

you currently are in the game to the next, where you will be after

that actions taken. So we're going to import gym and

we're going to import pandas as well, use that as well. And we're going to play this game

using the Environment Frozen Lake, V0 here line 0 for frozen Lake. And if you're curious,

I'm going to load it up in just a second. There's going to be a lot of games

available in the Open AI Gym and you can click on this

link that we have here. And that will allow you to look at what

goes into each one of the different environments. Now this environment,

the goal of the game is that you start at this S portion and you try to

make it to the G here at the end. So you see the object of the game is

to get to the goal G without landing in any holes. So the idea is that you're walking

across some frozen pond and you can go into the frozen portions and

you should be fine. But if you step on an H if you step on a

whole then you will fall into the hole and you lost the game. Now built into that and

will discuss this a bit later. It's not going to be that you can always

necessarily go in the direction that you're trying to go. If you choose to go to the left, down,

right, or up, you may buy some small probability end up going the wrong

direction and falling in a hole. So we have to take that into account

as well, and that's going to be what we're trying to learn in terms of

the probabilistic best path to take, otherwise would be obvious where

to go in order to get to G. So we're going to create our frozen our

environment by calling gym.make and passing in this string, FrozenLake-V0. And there's going to be a bunch of

environments that are available within the gym. And again you can look at

the documentation in order to get to this. We're then going to get

our current observation. And when we call environment dot reset,

what we're doing is we're resetting that our values so that we are starting

at that starting point at S. And we'll see that current observation

just second, we print that out and we see that were at observation 0. And zero should indicate that were

at the beginning of our process. Now, I want to look a bit deeper into

what our environment actually is. I'm going to call environments,

which is environment we just initiated .environment for this specific environment

and look at the documentation. And the documentation we can

see more about the game. So you read here that winter is here and

you and your friends were tossing around a Frisbee

at the Lake at the park when you wild, throw left of recently out

in the middle of the lake. And the goal is to get to that Frisbee and

they have a cute story here. But the water is mostly frozen and

there's Frisbee shortage, so you need to get to that disk. But the ice is slippery, so you won't

always move in that direction that you intend, which I mentioned earlier. And the surface is described

using the grid that we have here. Where again S is that starting point, F is

the frozen surface, H is the whole, and G is the goal that

you're trying to get to. So that's how this game

actually is going to work. If we want to look at the actual

environment, whether it's this game or another, we can look at our current

environment where that orange highlight indicates

where we currently are. What is our current state? So I'm going to print

out our action space and the accent space will include four

discrete actions that we can take. It will just say here will

print this out so we can see. It just says here discrete 4, but

that describes that we can go up, down, right or left. And then if we want to take an action,

what we can do, because we don't know which actions

to take, we can take a random action. So we go environment.action space,

which is that We just pulled out and we pull out this method sample, and just

randomly chooses moving up, down, right or to the left. And each one of those different directions

are going to be indicated by some number. And we see here on that first try that

randomly, we took the third direction, or actually the fourth because

Python starts at zero. And I can run this again still at three,

and I run this again we see that it chose

a random different sample of one. So now let's act. Let's actually take this action. So our new action is equal to

that sample that we're taking. So we're going to move up,

down, right, or left. And then we take a step, and that step is going to indicate actually

taking an action within our environments. And we're going to take a step

in the new action direction, and that will output this tuple here. And we're saving these as

observation reward whether or not it's done in some extra information. And we'll discuss this

a bit further later on, we're actually going to print them out

right here, and we can take a look. So we see that the observation

were still observation zero. There's no reward, we only get a reward

for getting to the end of the game. Are we done with the game now? And then there's this extra information, something about probabilities in

regards to the steps that we're taking. So if we run this again, we see that

we now took a different step, and now our state is different. Our orange square is now below. Again, our reward will still be zero

because we haven't reached the end of the game. Are we done? No, and that same information holds

in regards to those probabilities. Now we're going to take

five different moves. So for iron range five, we just

continually do what we just did, and we're going to print out

at each one of these steps. What observation is, what the reward is,

whether or not we're done, that extra information. And then we're also going

to render the actual state, the actual environments at each step. So we see we moved down. Then we moved down again. Then we moved up. Went to the right, and we see that we

made each one of these moves throughout. Again, the move isn't necessarily

indicative of where we end up. There's a probability that we

go in a different direction and we have to take that into account. The reward stays at zero unless

we reach the end of the game. And here it says done, and the reason

why we're done is because we hit a hole. And if you hit a hole,

then your game is over, right? Then you have lost the game. The game is over, you get zero reward. So that's why we have done

actually equal to true here. Now hopefully we have a clear idea or

a guess at two what each one of these outputs mean given that our

observation starting off with zero, then it moved to four. As we see here, but

if we think about those observations, it would make sense that

we start off at zero. And these are the numerical values for

every single state within our environment. So zero, one, two, three, four,

five, six, and we see this is five. And that's observation five. The reward refers to

the outcome of the game, and we only get a one if we're at that G. Done tells us that the game is still going

and again, that game ends either at G or once you land in a hole. Then the info gives us extra

information about the world. And here it's probabilities, and we ask

you to perhaps guess what this means here, and we'll talk about

this a bit further on. But again, it's the idea of there's some

type of probabilistic determination rather than just a clear determination in regards

to each step that you're going to take. So now we want to simulate

an entire episode. So again, we described an entire episode as a game

that's played from start to finish. Whether you land in a whole or

if you land at G. So we start off at

the current environments, current observation being at zero

by resetting our environments. We know that done is equal

to false to start off. And while that's false, we set that

new action equal to the sample. We then take a random action. And we keep doing this again since we have

this while loop until done is set to true. And each time we run this,

we're going to print out the new action, the new observation, the reward, whether

or not we're done in that information. So you run this. And we see each one of the actions taken,

the observations where we ended up. And then finally we ended up

again at the observation five, which is the same hole we fell

into before, and our game ended. So some things to notice about

the actions taken, and where. So we ended up. If we look at some of the actions and

where we end up, so we see here action zero and

observation four, but we ended up at observation eight, whereas

here we are back at observation four and we took action zero and

we stayed at observation four. So there's a probability that we don't end

up actually taking that same action or ending up in that same state given

that we took that same action. And if we think about it, the action

seems like again up, down, left or right, but they also have some type

of stochastic term built into that. And there seems to be a one-third chance

of going into a different square rather than the square that you're trying to

given that you chose a certain direction. Now I'm going to pause the video here as

we walk through just this setup here of working with Gym. And in the next video we're going

to start to gather information so we can start to figure out our

reward system and whether or not certain actions throughout all of our

steps are good actions or bad actions. All right, I'll see you in the next video.

Welcome back. Here

we are in Part 1. In our first video, we

introduced everything that we needed to understand

for OpenAI Gym. Here we're going to run a

bunch of game simulations through to the end

and start to observe, and gather data on what type of actions actually

lead to rewards. Some things to keep in

mind is we're going to want to store each one of

the different run throughs. We're going to name each

one of those episodes. We discussed earlier how from start to finish will

be a single episode. We start again, that'll

be the second episode, and so on and so forth. We're going to do that for

1000 or more episodes. Here we're actually going to

do it for 40,000 episodes. At each step, we're

going to want to save the observation where

we currently are, the action taken from

that observation, and then the current reward, whether or not we've

reached a reward state. Some things to keep in mind

as we go through this, we're going to want to

reset our environments after each episode so

that we start anew. We saw before how we output this tuple of a new

observation, reward, whether or not it's done

in that extra information, every time we take a step, every time we take an action. We're going to want to

continue this game until done is equal to true. Until we're actually

finished running through, that's going to be one

of these outputs that we have and we saw how

we did that up above. Then some things that

we can work with if we think about

env.action\_space.n, that's going to give us the

number of possible actions. If we think about

that, that should be four possible actions; up, down, right, and left. Then if we want to take a sample, that's going to be that

random step and that's what we're going to be doing

as we gather data. Then if you want to know how many possible states are

on environment, we can call

env.observation\_space.n and that will tell us how

many possible states are. If we think about our

four by four grid, that means that's going to

be 16 different states. So things that you can look at in general as you work with

these environments. Walking through

what we have here, we're going to reinstate

our environment. We're saying that we want to go through 40,000 full episodes, so resetting up until the end. That's going to be our goal as we run through this for loop. We're going to save all of

them in this life memory. Then we say for i in range

the number of episodes. I'm actually going to

pull this out so that we can look at some of

these steps separately. This is the entire for

loop that I'm going to pull out and put into

a separate cell. We have the old observation. Starting off our old observation

will be starting at 0.0. Whether or not it's done,

we're starting at false. The total reward at this

point is equal to 0. The episode memory, again, we're in this for

loop that we just pulled out is set to blank. The while not done, we take a new action. That new action is just

going to be a random action. We saw earlier how we can call random action using

env.action\_space.sample. We can then get using that new

action and using env.step. That's how we take an action

within our environments. We can get the

current observation, the reward, whether

or not we're done. Then our total reward

for these steps will be equal to the pass

reward plus a new reward. Generally speaking,

this will only add up to 01 because once we hit one, we've hit the end of the game, or we can end the game if we land in a hole and therefore, we'd stay at zero. Then what we want to save at each step is going to

be the observation. Where were we before? What action did we take? What was the reward once

we took that action? Then which episode run? Which will stay the same

until we end the game. This is still all

within this while loop. We keep doing this

until we hit done. Let's actually just run this portion when I put

this in a cell below. We've run now one episode, and let's just see what our i. It's from a different for loop. We have i is equal to 4, so they're all going to

be listed as episode 4. But if we look at the

episode memory now, we see that we took a few steps, observation zero,

the actions that we took until we hit

the end of the game. We hit the end of the

game by ending in a hole, we didn't actually reach

through to our reward. Then when we put this

into a Pandas DataFrame, which is what we're

ultimately going to do, we're going to want

to save with it. Here, we have the observation, the action, the

reward, the episode. All of these will be different

features in our DataFrame. We're also going to

want the total reward, which is just going

to be those values that we added up as

new rewards came in. If we did hit a reward

and we did get to one, then we'd attribute one to every single one of our

different observations. Then we want to

look at this value, i times tot\_rewards divided

by number of steps, and what that does and we'll look at this a

little bit later on once we actually reach

that reward state. Is this going to weight each

of our steps according to how close we were to

achieving that award? In the first step, that

action is probably not as important as that last step

that led to our reward. That will get a

higher reward step, a higher decay\_reward. That's what that's

going to be called. We're going to extend

that life\_memory using this, oh wait, life\_memory sorry is up here, and I pulled that

out of the for loop, but we can add on

that ep\_memory and let's just look at what

the ep\_memory looks like. You see that it's going

to be the same as before, except now we added on that tot\_reward plus

that decay\_reward. That's going to be everything here within the for

loop and then that's going to be added

onto our life\_memory, which is our list here. Then at the end

we're going to take that life\_memory and put that

into a Pandas DataFrame. Let's run this, and this

will take just a bit to run as we're running through

40,000 different iterations. Now it's run, we're going

to call memory\_df.describe, and we see for each

one of our features, that we have the

number of observations or where we were and the

average space where we were, and usually or probably closer to the beginning before we end, the actions taken, the reward, that mean reward will be

useful to understand, but we'll get a little bit deeper into understanding

the full reward. That probably means more around the tot\_reward amounts

where we have 2.4, and what's going to be even

or make this a bit clear, well, first let's

look at the shape. We see that we have

306,997 different rows, and if we look at a

couple of values here, let's say memory\_df, such

that the memory\_df is, or let's just look at the first episode or let's look where there's

actually a reward. Well, let's look where the

tot\_reward equals one, and I'll explain why

in just a second. We have our tot\_reward

here equal to one, and we see that it took quite

a few steps to get there, and now we can start

to understand what each one of these new

rows actually mean. If we look at the reward

at each individual step, and these are in order, well, let's also just look at a

single one of the episodes. Let's say episode is 182 because we see that

that ended in a reward. If we look at each one of these different steps,

they're in order, and we see that the reward is zero all the way up until the end when we got

to that final step. With that, our tot\_reward

is then equal to one, and it says that for episode 182, given the steps that we took, we were able to get to that end goal that

we were looking for, and then the decay\_reward

weights each one of the steps differently according to how close we were to getting

to that final step. Here, we are only a step away, so we get more decay\_reward, whereas at the beginning

in the first one, those steps probably

aren't as important in regards to getting to that final reward that

we're looking for. That's going to play a

role in deciding how we're going to ultimately optimize

on a rewarding system. Then to see how often we

actually got to the reward, we're going to group by episode. Grouping all of our values by episode we're summing the reward, and recall that each episode will either sum to zero or sum to one. As we saw here, these are all zeros for the reward feature except

for that final value. If we group by episode, it'll either be zero or one, and if we take the average value, then we can see on average

how often we were successful, and we see that we were only successful 1.4

percent of the time. If we just take random steps, we probably won't be

successful very often. The next goal and what we'll

discuss in the next video, is actually leveraging the

data that we gathered, all of those 300,000 x amount

of rows that we now have, to come up with more

intelligent steps to take throughout our system. All right, I'll see you there.

In this video, we're going to go about actually

predicting what the next step should be. So in order to do that, we're actually

going to model using those observations that we just built using that

data frame that we just built. So we're going to create somewhat of

a supervised learning problem now. And we're going to import random forests

in our extra trees regressor, and if we recall extra trees regressor is just

going to be more randomized version of random force where the splits

are going to be random. So we're going to set our extra

trees regressor to our model, set the number of estimators. We're then going to set what our

target value is going to be. So we need to create a wide variable

given the information that we have and the way that we're going to do that

is we're going to weight some of the different rewards that we

had within our data frame. So we had the actual reward at each

step and recall that's only going to be available really at that final step

if we did end up reaching that reward. We'll do 0.1 times the decay reward and

recall that that's going to more heavily wait those actions that are going to

lead closer to that final reward. And then we're going to put the most

weight to the total reward, whether or not any step along the way led to us

ultimately getting reward within that observation, within that episode. And then our x variable our x data

that we care about is just going to be the observations and

the action taken at that observation. So given the state that we're at,

what action did we take and deciding given that action whether or

not that lead to higher or lower reward? So we fit that given our x and y and that gives us steps to take along the way. And it gives us an outcome of

given the x values that went in, what is going to be the predicted y? And then ultimately if we think about it,

the goal will be take the observation that we're currently at, pass each one of the

optional actions that we have, and then return which one of those actions resulted

in our model giving the highest output. So let's see that here. So we have our model here, we're going

to use random forests regressor. We're using that same y and

that x that we just discussed and then we're fitting our model. We're then going to do 500 different

episodes this time just to see the results. We don't need to worry about this right

now and then we're going to save our life memory as we did before, and all of

these steps are very similar to before. We have our initial observation for

episode. We're not done yet, so our done is equal

to false until we actually get to done and our total reward is 0 and

we have this blank episode memory for the specific episode. Now here's our first major difference. Our predictions are going to take all

the old observation as well as i for i in range four. Recall that we only have four different

steps and if we look at say some old observation, we can see usually

that will say here that's at five and we can actually take

the model that we fit above. Well, actually let's just do this first. We can see what our data is we're

saying passing in this tuple and recall that this tuple should relate

to the observation in the action. So where observation five,

we can either take step zero or action zero, take action one,

take action two or take action three when

we're at observation five. We then call our model and call predict

for each one of these different tuples. And when we do that,

we have that model that we did above. We can actually call

model.predict here then. This is again referencing

this model above, not the model that we

have initiated down here. But you see that this outputs

four different values, where that maximum value

should be the next step, because that's predicting the most

amount of reward possible. And that's why we call np.argmax

on the output of what we had here. T.argmax and that will tell

us which action would lead to the highest possible reward. And then we take that new action, so

now our new action is going to be decided by that Argmax and then as we go along,

we keep adding on the reward. We append on the actual observation

that we're at the action. We took the reward and

the episode that we're at and we say all the observation and

run through this loop until we hit that done equal to trail, right,

this over here being equal to true. We're then going to incorporate that

the total reward into our episode memory. So once this is done running,

we're going to say if we hit any reward then we're

setting not equal to one or zero. Generally speaking as we discussed,

if we landed in a whole, we'd end up with a total reward of zero. If we were able to get to that end point, then we would end up with

a total reward of one. And then we add that onto our life memory

and then we have our new data frame with this new life memory and then we can look

at the mean value of our episode given the fact that we're now taking

more educated steps along the way. So I'm going to run this. This will take just a second to run, we're

going through 500 different iterations. Now we have a model that

has to fit first and also predict along every

single step along the way. So we're going to pause the video as

it's taking just a second to run. And then here we see the results

that we ended up getting and that may have taken just a bit of time. We see that it got here up to 62.4, so much better in terms of how often

it was able to get to that end goal. I'm not promising to be this high,

that's going to be somewhat random. There is a bit of a stochastic process

there, but still we see how much we were able to improve once we

implemented that reward system. Now that closes out our video here. In the last video in regards

to reinforcement learning, we're going to introduce

one last environment and see how we can work through

that environment as well. And hopefully as you start to go through

reinforcement learning on your own and play around in the open AI gym,

you'll be ready to hit the ground running. All right, I'll see you there.

Welcome back to our final video here on reinforcement learning. In this video, we're going to

touch on a new environment. We're going to be working

in a new environment so that you learn how to work outside of just that

frozen environment that we discussed earlier. Here we're going to work

with the Cart-Pole-v0. As before, you can look at the documentation which

can be helpful. But what we can

also do and what I want to reference here is the actual site on OpenAI

discussing this environment. Then we'll come back

to the documentation as well as, as more details. The idea of Cart-Pole

is that there's a pole attached by an unactuated

joint to a cart, as we see in the

picture to the right, which moves along a

frictionless track. That system is to control

by applying a force of either plus one or

minus one to the cart. The pendulum starts upright and the goal is to prevent

it from falling over. A reward of plus one is provided every single timestep that

their pole remains upright. Now our reward system

is different rather than there just being a reward

at the end of the game, there can be a reward

at every single step. Then the episode ends

when the pole is more than 15 degrees

from the vertical. If we were moved further

than 15 degrees from the vertical or the cart moves from that center more than 2.4 units. If you're able to

move it over a wall, keeping it upright for 2.4 units, then you also end the game. The goal is to keep

it up without moving 2.4 units for as

long as possible. The maximum, it also

maxes out at 200. That's going to be the

maximum amount of reward that we can get for

a single episode. Now some details that

are worth knowing that in our documentation here is that at each

timestep observations, nowhere state, we can

get the cart position, the cart velocity, and we have

the min and max for each. This only goes from

negative 4.8-4.8. The velocity can go as fast

or as slow as possible, or as fast in the

opposite direction. We have the angle that goes

from negative 24 degrees to 24 degrees and then the

velocity at the tip. That's going to be

different observations that will be available to us. If we think about it, again, we're going to want to leverage those observations in order to come up with our modeling of

how to optimize on rewards. There's only going to be

two different actions rather than the four

that we had before, either pushing the cart to the left one unit or pushing the cart to

the right one unit. Now many of these

steps are going to be very similar to

what we did before. First we're going

to gather our data by just taking random actions. We see here that we're using

the env.action\_space.sample. We're doing a random action. For each random action, we get the observation. Now again, the

observation is going to have four different values. We're going to get the reward. Again, we can get a reward

now at every single step, whether or not it's done and we talked about the criteria

that being done. Either it moves a certain amount of units without falling, it does go beyond

certain degrees, so therefore it does fall. Or if we're able to

keep it balanced for 200 times steps and then

some extra information. Now our total reward also is

going to be more important. As each time step we can add on more rewards

and we can end the game and have a pretty successful game without maxing out

the total reward, which would be 200. Then we're going to

append to our memory. This is going to be

something we put into our DataFrame what the

observation 0 was. I don't recall which one's which, but we had the velocity. We can call this again,

if you're curious. We had the position, the velocity, the angle, and the velocity at the tip. Those are going to be our

different observations. Then we have the action taken given the state

that we are in, which is highlighted

by observation 0-3, we get whether or not we had a reward and we keep track

of what episode ran. Then we set the old observation

to the new observation and we continue until we

reach the end of the game. Then at the end of the game, we're also going to save in our dictionary what

the total reward is, as we did so that when we

get our DataFrame at end, we can look at that as well. We run this and we put this into DataFrame as we did before. We see that the mean number

of steps that we're able to take without us failing is 22. We can look at our memory df. We see the observations and the average

values for each one of the observations and the

different actions taken, which should average

out at 50 percent because it's either zero or one. The reward average most of the

time is very close to one. It only failed at zero, and that only happened at

the end of the game so, on the 23rd step. Then what will make this clear is if we look

again as we did before. Let's first go here and

let's actually look where memory\_df.total\_reward

is the max value. Hopefully we have

one here where it go all the way to 200. We only got up to 94 here. I guess that's because

we're taking random steps. Probably once we optimize

will be able to get to 200. We can see that we took

either an action of one or zero throughout

and we are able to keep it balanced

without moving to 2.4 units for 94 different timesteps. Now as before, let's

create our regressor. We're also going to

have to create our y-variable that we're

trying to optimize on. If we see here, we're

going to create this comb reward or

that combined reward. That's going to be 0.5

times the actual reward at each timestep plus

the total reward. We're going to optimize

more on whether we are able to remain higher on

that total reward. Then we're going to fit. Here are extra-trees regressor on those different observations

and the actions taken. That's similar to before or

except before observation or our state could only

take on one value here as four different values

that describe that state. We have our action

and we're trying to optimize on this

memory\_df.com\_reward. Given all of these

different values, what's going to be the

predicted combined reward that we just came up with? Now that we have our model fit, we're going to use the same

steps that we did before. Here, we're taking

that old observation. It's actually, again

show this above. First we have to run this. Then we're going to

look at this above. We see that we have all of the observation values and then the action that

we'd want to take, either zero or one. We have either zero or one. That's going to be that

plus i in range 2. That's going to be

our input values and given those input values, we can come up with a

prediction as we did before. We call model.predict and we get the two

different values and we just choose that

maximum value again. At each step, we choose

whether to go left or right, according to which one maximizes the potential output given

the data that we've gathered. Then all of the steps from there are essentially the same, saving it all into memory. Then we can see,

given our new model, how much further

along we were able to keep that cart balanced. I'm going to run this and

we'll pause the video again and we'll

come back once it's done running and

discuss those results. Now, as we see here

in our output 49, we were able to get up to

113.77 on average in regards to how many different

steps are able to take and each step is going

to add onto that reward. We see by optimizing our model

on our combined rewards we are able to greatly increase

from 22 up to 113.77. Now I also pulled out here

from our new DataFrame, the actual total rewards and where those total

rewards maxed out. As mentioned, if

we get up to 200, it'll stop the game and say that you've accomplished that

highest goal possible. We see we got here up

to 200 for 2,000 rows. This is for every

action. That means we got there about 10 times. We see we're able to max

out what you are not able to do when you are

only taking random steps. Now that closes out our video here on reinforcement learning. I encourage you to dive into the OpenAI website and

keep playing around with different environments

that may be available to you to keep learning more and more about

reinforcement learning.

# Summary/Review

## **Generative Adversarial Networks (GANs)**

The invention of GANs was connected to neural networks’ vulnerability to adversarial examples. Researchers were going to run a speech synthesis contest, to see which neural network could generate the most realistic-sounding speech.

A neural network - the “discriminator” - would judge whether the speech was real or not.

In the end, they decided not to run the contest, because they realized people would generate speech to fool this particular network, rather than actually generating realistic speech.

These are the step to train GANs

* Randomly initialize weights of generator and discriminator networks
* Randomly initialize noise vector and generate image using generator
* Predict probability generated image is real using discriminator
* Compute losses both assuming the image was fake and assuming it was real
* Train the discriminator to output whether the image is fake
* Compute the penalty for the discriminator probability, without using it to train the discriminator
* Train the generator to generate images that the discriminator thinks are real
* Use the discriminator to calculate the probability that a real image is real
* Use L to train the discriminator to output 1 when it sees real images

## **Reinforcement Learning**

In Reinforcement Learning, Agents interact with an Environment

They choose from a set of available Actions

The actions impact the Environment, which impacts agents via Rewards

Rewards are generally unknown and must be estimated by the agent

The process repeats dynamically, so agents learn how to estimate rewards over time

Advances in deep learning have led to many recent RL developments:

* In 2013, researchers from DeepMind developed a system to play Atari games
* In 2017, the AlphaGo system defeated the world champion in Go

In general, RL algorithms have been limited due to significant data and computational requirements.

As a result, many well-known use cases involve learning to play games. More recently, progress has been made in areas with more direct business applications.

## **Reinforcement Learning Architecture**

The main components of reinforcement learning are: Policy, Agents, Actions, State, and Reward.

Solutions represents a Policy by which Agents choose Actions in response to the State

Agents typically maximize expected rewards over time

In Python, the most common library for RL is Open AI GYM

This differs from typical Machine Learning Problems:

Unlike labels, rewards are not known and are often highly uncertain

As actions impact the environment, the state changes, which changes the problem

Agents face a tradeoff between rewards in different periods

Examples of everyday applications of Reinforcement Learning include recommendation engines, marketing, and automated bidding.