Demo & Intro
0:00
hey what’s up guys I have something that will absolutely blow your mind this is
0:05
the computer playing Super Mario Bros and guess what the computer has no knowledge about the internal State or
0:11
the mechanics of the video game rather it’s simply looking at the frames and then deciding what buttons to press
0:18
exactly how humans do it impressive right well if you want to learn how to
0:24
implement this stay tuned because that’s exactly what I’ll be sharing with you guys today and I cannot wait
0:30
this was done using a subclass of machine learning algorithms called reinforcement learning reinforcement
0:36
learning is the coolest thing ever in fact it’s been used to train drones and
0:41
robots find more efficient ways for matrix multiplication and even make the training process for other machine
0:48
learning algorithms faster oh and not to mention it was also used to train
0:53
everyone’s favorite chatbot chat GPT reinforcement learning is pretty different from its cousin’s supervised
0:59
and unsupervised learning in supervised learning the model is given a data set
1:04
where each point has a correct answer associated with it whether it’s categorical or numerical the model’s job
1:11
is then to predict these values accurately an unsupervised learning the data points
1:17
don’t have a correct answer but rather the models tasked with extracting General Trends or insights reinforcement
1:24
learning is its own Beast the objective of reinforcement learning is for the agent to maximize its rewards inside of
1:31
the environment we give the agent a positive reward whenever it exhibits behaviors we want
1:36
to persist and negative ones for behaviors that we don’t want it’s kind of like teaching a dog new tricks
1:42
another thing to note is that in supervised and in unsupervised learning you must have the data pre-collected
1:49
before you feed it to the model whereas in reinforcement learning we’ll see today our agent go out into the world
1:55
collect its own experiences and then learn from it it’s a completely different Paradigm when it comes to
2:01
training I wanted to share an implementation of the ddqn algorithm or the double deep Q Network I’m not going

2:09
to dive into the nitty-gritty of reinforcement learning theory I'd rather just share a more intuitive
2:14
understanding for those that have a base level understanding of machine learning but haven't really been exposed to
2:19
reinforcement learning before however for those that are interested I'm going to leave some of my favorite
2:25
reinforcement learning resources down in the description below along with the paper that originally pioneered the
2:31
double deep Q Network algorithm I first implemented this algorithm when I was a sophomore in college and I still
2:37
remember the excitement that I felt when I first watched the model start to train learn and eventually beat the level
2:44
I was absolutely awestruck I mean I understood all of the math and each line of code that I wrote to implement the
2:50
algorithm however there's just something that's still magical about reinforcement learning that gets me excited every
2:56
single time and my goal today is to be able to package that excitement and share it with you guys through this
Key Reinforcement Learning Vocabulary
3:02
video let's start off with the basics in order to understand the fundamentals of reinforcement learning we need to Define
3:08
these following terms let's understand each of them in the context of Super Mario Bros first is the agent the agent
3:17
today will be our neural network that's controlling Mario it's the one responsible for making decisions and
3:23
taking the optimal action the environment is what our agent will be interacting with it's going to be level
3:30
one one of Super Mario Bros on the NES a state is a snapshot of the environment
3:36
at any given time it's essentially what our agent sees today we'll be using four
3:41
consecutive frames from the game the reason we're using consecutive frames is so our agent can see motion for example
3:49
is the Goomba coming towards us or away from us well for us to know we need to
3:54
know the previous frames oh well hopefully our agent knows to jump actions are the inputs that the
4:01
agent has available to it to give to the environment today the agent will have access to these five button combinations
4:09
to press first one is with no buttons pressed and then the agent is able to move right a is used to jump and the
4:16

longer it’s held the higher Mario will jump and then B is used to move faster
4:22
once the environment gives the agent a state and the agent responds with an
4:27
action the environment also gives the agent a reward a reward essentially
4:33
helps the agent determine whether or not the action it took inside of that state was good or bad today we’ll be giving
4:40
the agent a plus one reward for each unit to the right that it makes in the video game it’ll receive a negative one
4:46
reward for each second that passes by in the games clock this is to disincentivize the agent from standing
4:52
still and then the agent will get a massive negative 15 points if it dies
4:59
now the concept of a negative reward may seem foreign to some however remember the goal of the rewards is to guide the
5:06
agent to Optimal actions and negative rewards still help the agent do just
5:11
that everything that we’ve talked about so far is summarized in this diagram it
5:17
essentially outlines how the agent and the environment interact with one another the agent gives the environment
5:23
actions and the environment returns States and rewards an episode for us
5:29
today will just be one attempt at the level an episode will end whether Mario dies reaches the flag or Runs Out of
5:37
Time a policy is a function that takes in a state and returns an action it’s
5:43
what our agent will use to make its decisions some reinforcement learning algorithms will have the policy be a
5:50
probability distribution however today in our ddqn algorithm implementation
5:56
we’ll be using the Epsilon greedy approach more on that later a value function takes in a state and returns
6:03
how valuable that state is in reinforcement learning it’s natural to
6:08
view some states as more valuable than others for example the state where you’re right next to the flag is
6:15
probably a little bit more valuable than the one where you’re falling down a hole we’re not going to be working directly
6:20
with the value function today however I thought I’d mention it because a lot of other reinforcement learning literature
6:26
brings it up an action value function takes in a state and an action and then
6:33
returns how valuable this pair is as you can see on the screen this state action

6:39
pair is probably more valuable than this state action pair I'm sure you can see why approximating this function
6:44
accurately today is our goal because it's going to let our agent take the
6:49
action inside of in state that has the highest value alright quick review we're
6:55
using reinforcement learning to train the computer to play Super Mario Bros our agent or the neural network that's
7:01
controlling Mario will receive a state from the environment remember a state is
7:06
four consecutive frames from the video game the agent will then take an action
7:11
if the action was good we'll give it a positive reward if the action was bad it will give it a negative reward and then
7:18
the agent will use these rewards to train itself over time time to take better and better actions eventually
7:25
we're hoping that the agent will learn how to beat the level and reach the flag that's waiting for it at the end that
7:32
was a lot to take in thank you guys for bearing with me we have three more Concepts to cover first is the Epsilon
7:37
greedy approach second is the replay buffer and third are some more details about the action value function after
7:44
that we'll assemble the algorithm and then implement it in code the Epsilon greedy approach is the strategy our
Epsilon-Greedy Approach
7:50
agent will be using today to choose its actions however to understand Epsilon greedy we first must take a step back
7:57
and understand the explore exploit dilemma a problem that's at the heart of reinforcement learning let's say our
8:04
agent has learned that jumping over Koopas is the most effective way to not die good I mean I guess it gets the job
8:11
done right however as I'm sure my fellow Gamers know that jumping on a Koopa
8:17
actually turns its shell into a weapon that can be used to eliminate other enemies if our agent constantly exploits
8:24
or takes the action that it deems to be the best in the current moment then it will never get a chance to explore other
8:30
potentially Superior strategies so how do we navigate this explore exploit dilemma enter the world of the Epsilon
8:36
greedy approach a clever solution to the explore exploit dilemma with probability Epsilon our agent Ventures into the

8:43
unknown taking random actions and with the remaining probability 1 minus Epsilon it’ll take the action that it’s
8:49
confident is the best one it’s essential to note that this best action of the agent is dependent only on what it knows
8:55
in the current moment it’s not necessarily the best action overall initially when our agent knows nothing
9:01
about the environment we’ll set Epsilon to 1 ensuring maximum exploration and
9:06
then over time as our agent continues to learn about the environment and its Dynamics will start to taper off Epsilon
9:13
from one to a very small non-zero number we’ll always keep Epsilon greater than zero just so our agent is always on the
9:20
lookout for new strategies the this means that if our agent did learn to jump over Koopas we’re hoping that with
9:26
the Epsilon greedy approach it’ll eventually randomly jump on one and then realize the benefits of doing so now
Replay Buffer
9:33
let’s cover the replay buffer it’s essentially a storage for our past experiences and rewards it’ll be used to
9:39
train the neural network there are a couple of reasons why we use a replay buffer rather than training on the
9:44
experiences as we collect them the first is because sequential experiences have a lot of correlation which can lead to a
9:52
lot of instability when training the model to eliminate this instability we randomly sample from our replay buffer
9:58
and the second reason is that we’re now able to reuse data which actually improves data efficiency in the replay
10:05
buffer we’ll be storing tuples of the state action taken reward received next
10:11
state and a Boolean flag indicating if the episode is done for any reason we’ll see how each of these come into play
10:17
when we sample the replay buffer to train our Network now let’s pull up the action value function that we mentioned
Action-Value Function Intuition
10:23
earlier this is often referred to as the Bellman equation remember our primary objective today is to approximate this
10:30
function enabling our agent to make optimal decisions in any given State the goal isn’t just about seeking immediate
10:36
gratification instead the agent aims to maximize its rewards over the long run humans intuitively do this as well for

10:44
example we may take a slightly longer route with less coins if it leads us to an item box at the end that contains the
10:50
invincibility star math notation is scary so let me translate this equation to English the value of taking action a
10:58
in state s is equal to the reward you get in the next time step plus the value
11:03
of taking the best action in the next state S Prime multiplied by a discount
11:09
Factor the discount Factor makes rewards from future States less valuable in
11:14
reinforcement learning this is important because future rewards are less predictable than the current one due to
11:20
the environment’s stochastic in the Stanford marshmallow experiment children were given a marshmallow and
11:26
were told that they could either eat it right now or wait 15 minutes for a second marshmallow it was to measure
11:33
their self-control when it came to delayed gratification essentially the researchers were measuring the
11:39
children’s Gamma or their discount Factor one marshmallow now or two in the
11:44
future our agent is kind of in a similar situation except for the children their
11:50
second marshmallow was guaranteed unfortunately our agent does not have that luxury as such it’s ideal for it to
11:58
value a guaranteed marshmallow in the present more than a potential marshmallow in the future a gamma of one
12:05
means that there is no discounting while a gamma of zero means the agent is completely myopic we want neither but
12:13
rather to strike a balance between the two this action value function is recursive in nature this means that the
12:20
value of a specific State action pair is not just based on the immediate reward rather it also depends on the value of
12:27
the subsequent State action pair dive deeper and you’ll see that this subsequent value in turn depends on its
12:33
immediate reward and the value of the next state action pair and the chain continues we assume that we take the
12:39
optimal action the action with the highest Q value at each recursive level thus the max function when you unravel
12:46
this recursive structure you’ll find that the value of any state action pair is essentially the sum of its immediate
12:52

reward and the accumulated discounted feature rewards until the end of the episode this first equation once you
12:59
assume you're taking the best action at each time step condenses down to this after you distribute the Gammas
13:05
appropriately the further out a reward is the more times it's multiplied by gamma still a little bit confused don't
13:12
worry I was too when I first learned this let's try to tackle this from my more visual approach take each ribbon on
13:18
the screen to be a reward the agent will get until it reaches the flag remember that these rewards are granted after
13:24
each step the agent takes in the environment and the number on the screen is arbitrary the ribbons are getting
13:30
progressively smaller to signify the discounting assuming the agent is currently in state s and takes action a
13:36
the value of this state action pair is the sum of this entire sequence of rewards discounted appropriately if S
13:44
Prime is the next state after taking action a in state s and a prime is the
13:49
best possible action in S Prime then the value of S Prime a prime will be this
13:54
sum again discounted appropriately we know the reward R we got from going from
13:59
s to S Prime so we can use that to help tune the estimate of the current state action pair's value to train our agent
14:07
We compare its predicted value of a state action pair to the Target value remember the target value is the
14:13
immediate reward from a move plus the disc discounted value of the best possible next move essentially We
14:19
compare its predicted value of the sum of these ribbons to the predicted value of the sum of these ribbons plus the
14:26
ribbon whose value we know with certainty because the agent just received that reward while both the
14:32
prediction and the target are estimates the target is considered to be more reliable why is that the case well
14:39
because the target is based on fewer future assumptions as a component of the target is the reward directly observed
14:45
from the agent's recent action making it grounded in immediate experience rather
14:50
than speculative projection once we have our predicted value and our Target value we can use the following equation to
14:57

update our estimate for those of you familiar with gradient descent you might recognize this to be exactly that Alpha
15:04
is our learning rate or the size of the steps our Network’s parameters take as they converge towards their optimal
15:09
values the part inside of the square brackets is the error or the difference between our predicted and Target values
15:15
this is exactly the derivative of the mean squared error loss function I’m sorry if all of that was overwhelming I
The DDQN Algorithm
15:21
know it was for me the very first time I learned it however the intuition of the action value function is the hardest
15:27
part I swear it gets easier from here so how do we even begin to approximate this function that has such a vast space of
15:33
potential State action pairs well we turn to our friend the neural network remember neural networks are great at
15:40
approximating functions and extremely high Dimensions the specific architecture we’ll be using today is the
15:47
convolutional neural network or a CNN CNN’s excel at extracting meaningful
15:52
visual features from its inputs whether its enemies holds or the flag if you’re not too familiar with CNN’s I highly
15:59
recommend nvidia’s learning deep learning book it’ll cover literally everything you need to know about CNN
16:05
the input size will be the dimensions of our state which we’ll see in just a sec the number of neurons in the output
16:12
layer will be the number of actions available to us which in this case will be five the value in each neuron
16:19
represents the predicted Q value for that Associated action paired with the
16:24
input State when running this algorithm we’re actually going to have two identical copies of our neural network
16:30
the first one will be the online Network this is the one that will actually be training the second one will be the
16:37
target Network the one we’ll be using for the ground Truth for our predictions to give to our loss function the target
16:43
Network won’t be trained however we’ll be intermittently copying the weights from the online network over to the
16:49
Target Network this is to increase stability during the training process to train the online Network we sample
16:56

the replay buffer to get a random Tuple containing a state action reward next state and done flag we then pass in the
17:04
state to the online Network to get the predicted Q values for the five actions the different sized controllers
17:10
represent how the network values each action differently however we only care about the action that was taken and is
17:17
present in the sampled Tuple this is our predicted value for this state action
17:22
pair to see how accurate it is we calculate our Target value to do so we pass in the
17:29
next state into our Target Network to get the predicted Q values for the five actions again this time we take the
17:37
highest predicted Q value represented by the largest controller to compute our
17:42
Target value we take the reward from our sample and add it to the predicted Q value of the next state multiplied by
17:49
gamma our discount Factor we would then pass these two values into our loss
17:54
function and perform one learning step for our online Network this technique is known as bootstrapping because our poor
18:02
little network is essentially pulling itself up by its bootstraps it’s using a crappy Target Network to train our
18:09
crappy online Network and then as the online Network improves just a little bit we’ll copy over the weights from the
18:15
online Network to the Target Network and now we’re using a slightly less crappy Target Network to continue training our
18:22
online Network we’ll repeat keep this process tens of thousands of times and eventually we’ll start to see our agent
18:28
actually start to learn I know this seems very Jank like we’re using a moving estimate to train another
18:34
estimate however in reality we’ll see how stable it actually is so let’s put
DDQN Pseudocode
18:40
together all of these moving parts and finally assemble our ddq1 algorithm we start off with initializing our starting
18:46
variables and resetting our environment which gives us the starting State second given the state we’ll choose an action
18:53
with our Epsilon greedy approach third we’ll use this action to take one
18:59
step inside of the environment as mentioned previously the environment will return a new state and a reward
19:05
we’ll then take the current state action reward the new state and the done flag
19:13
and store them all in our replay buffer then we’ll do one learning step which involves sampling the replay

buffer and
19:20
training the online Network then we'll take our next state and assign it to our
19:25
current state we'll repeat all of this until the episode is done and finally
19:31
we'll repeat this inside the loop for however many episodes we want to train for so let's finally get to coding this
Implementation in Code
19:39
up for the NES emulator we'll be using a library called gym Super Mario Bros it
19:45
handles all of the complexities of the emulator side of things for us it gives us neatly packaged python objects for
19:51
the States along with a simple way to input actions it does the this via a
19:56
simple API that's based on openai's gem API openai's gym library is a wildly
20:03
popular collection of different reinforcement learning environments that researchers or enthusiasts like us can
20:10
use to try out different algorithms they're incredibly easy to set up and use these days the most common approach
20:17
is to use gymnasium which is a maintained Fork of openai's gym Library by an outside team I wonder what openai
20:25
is working on these days also all of the code will be linked on GitHub right down in the description and if you ever need
20:32
to reference it throughout this video or even afterwards please feel free to do so so let's start off with a simple
20:38
example let's first import the libraries we'll be limiting our agent to only press these following button
20:44
combinations which is why we're importing write only then we create our environment object
20:50
and wrap it with the joypad space wrapper ensuring that the actions in write only are the only ones the
20:57
environment can accept then we set our done flag to false and
21:02
reset our environment this while loop will run for as long as the agent is still alive
21:08
first let's have our agent only press right on the d-pad we'll then take one step in the environment with that action
21:15
and get our done flag ignoring the other return values from the step function for now and finally because on line 6 we
21:24
have our render mode set to human calling env.render will display our
21:29
environment to our screen so we can see what our agent is doing as expected our
21:35
agent is not doing so well to add some dynamism let's have it randomly choose an action from our action

space
21:42
unfortunately our agent still sucks good thing we have an entire reinforcement
21:48
learning algorithm up Our Sleeve let's implement the wrappers first rappers are essentially a way that we can modify our
21:55
environment's outputs if you want some more information about writing your own wrappers I highly recommend checking out
22:01
the gymnasium documentation which is linked below we're going to be using four different
22:06
wrappers today the wrapper we're implementing will skip frames meaning it'll just take the action inputted for
22:12
the first frame and reapply it for however many frames we want to skip we'll aggregate the rewards over those
22:19
four frames this is useful for us because consecutive frames have a lot of overlap so it's redundant to reprocess
22:26
everything the next three rappers are already implemented by gymnasium resize
22:32
observation will change the dimensions of a frame from 240 by 256 pixels to 84
22:40
by 84. this is just to reduce the computational load similarly grayscale
22:46
observation will turn the frame from having a red green and blue channel to just one reducing the amount of data our
22:53
network will have to crunch finally so our Network can see motion we're going to take four consecutive
23:00
frames and stack them on top of each other to finally create our state object that will pass into our Network each
23:07
final State object comprised of a stack of four processed frames encapsulates data from 16 original game frames due to
23:15
the combined effects of frame skipping resizing and stacking in our
23:20
pre-processing steps the code for the wrappers is pretty straightforward as we're only implementing one of them we first
23:27
override the Constructor to take the number of frames to skip then we override the step function whenever step
23:34
is called we use the for Loop to take skip number of steps aggregating the
23:39
rewards the apply wrappers function applies our wrappers one by one
23:44
encapsulating our base environment like the layers of an onion at the end it
23:50
returns our fully wrapped environment object we'll use this function in our main.pi
23:56
let's next create our neural network with pi torch as mentioned previously we'll be using a convolutional

neural
24:03
network we’re going to have three convolutional layers followed by two linear layers the input shape will be 4
24:10
by 84 by 84 which is the dimensions of our state the number of actions will be five the
24:16
underscore get conv out function performs a dummy forward pass through the convolutional layers to get the
24:23
number of neurons we need in our first linear layer it’s 3136 with this specific architecture but
24:31
the function allows us to dynamically calculate it where we to change anything in the future
24:37
the freeze flag allows us to prevent pie Torch from calculating the gradients which we’ll need for the Target Network
24:43
remember we only use the target Network to calculate the correct values we want
24:48
our online Network to predict then finally we add our forward pass and then move the network to whichever device
24:54
we’re training on whether it’s the CPU or GPU let’s next create the agent class
25:00
these will be our Imports along with pytorch and numpy we’ll also import our
25:05
neural network that we just created for the replay buffer we’ll be using pi torch’s built-in tensor dict replay
25:12
buffer which will hold python dictionaries with tensors as the values
25:17
the storage mechanism it’ll be using is pi torch’s lazy mem map storage which
25:23
will use memory mapped files for easy access to our experiences along with
25:28
alleviating RAM usage we could have just used a python list for our experiences from which we sample
25:35
but that means sampling would be really slow and we’d be using a lot of RAM
25:40
for the Constructor we’ll take in the dimensions of our state and the number of actions our agent has access to we’ll
25:47
also use a learn step counter which keeps track of how many times we’ve trained our Network
25:53
we then set up our hyper parameters first is the learning rate or Alpha this
25:59
is the size of the steps the network will take when it’s updating its weights second is gamma which is our discount
26:05
Factor remember we want the agent to Discount future Rewards
26:10
third is the starting value of Epsilon which will be set to 1 initially to encourage exploration
26:16
fourth is the Epsilon Decay factor which will multiply Epsilon with after every
26:21
time step fifth is our minimum Epsilon so we always maintain some likelihood to

26:27
explore sixth is the batch size for our training and finally is how often we'll sync the
26:35
target Network weights with the online Network after the hyper parameters comes the networks again making sure the
26:42
target Network's parameters are frozen after adding our Optimizer and loss we create our replay buffer with the
26:49
capacity to hold a hundred thousand experiences then let's write our choose
26:55
action function that uses the Epsilon greedy approach as we can see when the random number is less than our Epsilon
27:02
value it'll choose a random action since Epsilon starts off at 1 the action
27:07
chosen will always be random as the value of Epsilon decays the probability
27:12
of using our online Network to choose the action with the highest Q value increases also because we want the index
27:19
itself not the estimated value of that action we're taking the ARG Max our
27:25
Decay Epsilon function multiplies the current value of Epsilon with our Decay factor and ensures it doesn't go below
27:32
the minimum Epsilon threshold the store in memory function takes in the tensors that we want to put into our
27:39
replay buffer organizes them in a dictionary and adds them to the buffer
27:44
the sync networks function checks if enough learning steps have passed and if so it'll copy over the weights of the
27:51
online Network to the Target Network now for the most important function the
27:57
learn function in the agent class we first validate that there are enough experiences in our replay buffer to
28:03
sample a batch from then we call the sync networks function we then clear our
28:08
gradients followed by sampling the replay buffer we store the results in the states actions rewards next States
28:15
and duns variables we passed the state's tensor through the online Network to get
28:22
our predicted values we index by the actions that we actually took because we only perform back propagation on those
28:29
values then we calculate our Target values we pass in the next States through our
28:36
Target Network and get the value that the best action yields we then multiply
28:41
it by gamma and added to the rewards we got in the current state the one minus
28:46
duns part sets all future rewards to zero if we're in a terminal state

28:52
we then calculate the loss using our predicted and Target Q values performing
28:57
back propagation to calculate the gradients and then perform a step of gradient descent with those gradients we
29:04
also then increment our learn step counter and Decay Epsilon finally let’s create our main dot Pi in this file
29:12
we’ll set up our environment as we saw previously and then create the full training loop again starting off with
29:18
our Imports this time adding our apply wrappers function and our agent class
29:24
we then create our environment applying the joypad space wrapper plus the other four that we talked about earlier we
29:31
also create our agent object and pass in the dimensions of our state and the number of actions from our environment
29:37
finally the training Loop looks very similar to our pseudo code for each episode we collect experiences and learn
29:44
from them until the episode is done first we choose an action given the starting State again this will be
29:50
completely random at first then we take a step in the environment and store the state action reward new state and done
29:58
flag in our replay buffer we then perform one learning update then we set
30:04
our current state to our new state and repeat fair warning training for 50 000
30:09
iterations can take a while I have a RTX 3080 and it took me about two days of
30:14
training non-stop however I’m excited to see what optimizations you guys can uncover by tooting the hyper parameters

The AI Beats the Level!

30:21
but now the moment you’ve all been waiting for after we finish training
30:27
this is the final result
30:39
thank you

Conclusion

30:56
well we did it guys we finally trained the computer to play Super Mario Bros I
31:02
hope you guys feel the magic of reinforcement learning that I’ve always felt in fact even bigger than that I
31:08
hope you realize that machine learning is so much bigger than just predicting labels from a data set or trying to
31:14
Cluster some points anyways if you have any questions about anything that we talked about today
31:21
please leave a comment down below and until then see you guys next time