In this next set of videos, we're going to introduce another unsupervised learning and an often more powerful

generative model, namely Generative

Adversarial Networks. Now let's go over the learning

goals for this section. In this section, we'll

have an overview of Generative Adversarial

Networks or GANs. We'll discuss their

origin story as well as the motivation

behind why they work. We'll discuss the actual

training procedure for GANs and how we'll use

two adversarial networks, one for generating classes and one for discriminating

which classes are real. Then we'll discuss some

key hyperparameters that govern how GANs train and why GANs maybe even more sensitive to these hyperparameters compared

to normal neural networks. Now, part of the motivation of GANs is a realization

that the means by which neural networks interpret a new example make them vulnerable to

adversarial examples. A broader example if you were

to think of trying to learn a spam filter and

once a neural net has learned what makes an email

spam versus not spam, it then becomes possible using that same network to begin

designing emails that look as much as possible like the non-spam emails that can

trick our actual network. These are adversarial examples. If we look here at our image and we are

trying to determine whether or not this is

a hand written image, if our neural net learned a

bunch of handwritten images, then it actually knows all

the features that make up a handwritten image according

to that neural network. Therefore, we can

replicate that to produce something that according

to the neural network, will be classified as

a handwritten image. Now, we won't get into the math, but a way that we have learned to generate these

adversarial examples, such as that spam

that looks like it's not spam or non-written, digits just that look like

handwritten digits is to take a training set and focus on adjusting our original images in that backward pass in relation to each one

of our gradients. The invention of GANs

was connected to the neural networks vulnerability to these adversarial examples. Researchers were going to run some speech synthesis contest to see which neural

network would generate the most realistic

sounding speech and they would have

a neural network, the discriminator, to judge whether that speech

was real or not, but they decide not to run

the contest because they realize with neural networks, every possible for people

to generate speech that were just going to fool

this particular network, given the way that

it would be trained, rather than actually

generating realistic speech. The researchers

realized they could solve this by having

the discriminator continually improve

at distinguishing between real and fake speech. They could do this by feeding it real speech alongside fake

speech or in other words, introducing their own

adversarial examples and by incorporating the gradient of the resulting discriminator

with respect to the input. That loss in respect to x, back to another neural network that was actually

generating that speech, they realized that they

could train a network to generate very

realistic speech. This is the root of our

adversarial networks. We're trying to improve both our generator model so that it creates

better fake output, as well as our

discriminator model, which is meant to differentiate real and fake output

at the same time. They are going to be

adversarial networks with opposite goals. Again, we're going to have

that generative model trying to trick the network so that it cannot discriminate between

false and true examples, whereas we're going to

have discriminator working hard to get better at

discriminating between the two. As both networks try

to beat the other, they continue to improve. With this in mind,

what exactly are GANs or Generative

Adversarial Networks? GANs provide a way of training two neural networks

simultaneously. One of the neural

networks works as the generator and learns

to map random noise to images with the

goal of making them indistinguishable from those

within our training set. Looking at this image,

we start off with our generator network and that starts with an

input which is just going to be some random noise. Then tries to great image

indistinguishable from the training set images and not the same as

any particular image, but rather trying to find

similar properties of the image value distributions

in that training set. Then that produces

an image which is fed through the

discriminator and that discriminator is meant to decide for which

images are generated or are fake and which ones are the actual images from

our training set. Now, going back to GANs

and their original paper, GANs was established as

a training network in 2014 in a paper by

Ian Goodfellow.. They showed how they

were able to generate new high-quality 28 by

28 pixel MNIST digits. Those MNIST digits again are

those handwritten digits. The model relied on simple architectures for both the generator

and discriminator, where there were no

convolutional layers or rather just fully connected layers and ReLU activations

between each layer. There was also no dropout

and no regularization using either the generator or the

discriminator networks. The results were what

we see here below, where the images

produced were nearly indistinguishable from the handwritten values

in our training set. Again, none of the images

just shown existed in that original dataset

and just to show these alongside the actual

training set images, we see the generated nine and the generated

zero versus that from the training set

and then the same for the respective twos and

eights that we see here. Now that closes out this video. In the next video, we'll dive a bit deeper into the actual GAN training

procedure. I'll see you there.

In this video, we're actually going to

take a look at the generative adverse aerial network training procedures. So diving right in. The first step will be as usual with our

neural networks to start with a randomly initialized waits for our generator and

discriminator networks. Now to start our generator process,

which is meant to find some distribution that does well in representing

the actual values in our training set, be those images or whatever it may be. Will be to pass through

just a random noise vector. So our input is going to

be a random noise vector. And the goal is that our generator network

will be able to take this random noise and come up with the appropriate weights to

generate the complex distributions of our images of whatever our training set is. And then that generated image should

then be the same size as the actual training set images as they will later be

fed through our discriminator network. The next step, step 3, will be for the

discriminator to predict the probability that the generated image is a real image,

or one that our network has just graded. So once again, that is going to pass

through our discriminator network, our produce image, and the goal is to

discriminate between actual training images and

those are generated network is producing. And the output will be the probability

that we're working with an actual image and not a generated one. Step 4, we're then going to

want to compute the losses, both assuming that the image

we just generated is fake. And a loss function, assuming that

the image generated was real and we'll get into this in just a second. We're going to call the loss function,

assuming it was fake, which it is, of course. Just a reminder we are working here

in the realm of that generated image. So we're going to call the loss function,

assuming it was fake, L0. As in how far away was our discriminator

from predicting there's a zero probability that this

generated image is not real. And then we're going to call the loss

function, assuming it was real, which again, it's not going to

call that loss function L1. Representing how far discriminator

was from predicting this was 100% a real image. And then these two combined will

produce our total loss function, which we can later leverage to train both

our generator and discriminator networks. Now here in step 5, we'll make a little bit clearer why we

have these two different loss functions. So our discriminator network is going to

only care about correctly predicting that this image is not a real image. So I want to use that loss function L0, we're talking about

the discriminator network. So how far off was the output from

predicting 0 is what we're trying to get here. So we back propagate in relation to the

loss of how far off we were to saying that this is not a real image. And then we update the weights of only

are discriminator network accordingly. Now, where does our L1

penalty come into play? Our L1 penalty, again, the L1 being assuming that our

image produced was actually real. That's going to be ignored by

our discriminator network, coz our discriminator network

doesn't want to optimize, assuming that a fake image is real. The goal of computing L1 is to

ultimately tell us if we're doing a good job of producing an image

that seems realistic. And that's our output from our

discriminator is too far from 1. So we compute are gradients in regards

to L1 and back propagate through without training the discriminator and

pass that out ultimately to our generator. So rather than using that gradient

to train our discriminator network, we use it to actually update our

weights for our generator network. So we continue to back propagate using

that L1 gradient to update the weights of our generator. Now what's missing in this training

procedure that we just walked through? The goal of our discriminator network is

to learn to classify fake images as fake, real images as real. And for it to actually improve its ability

to distinguish between fake an real images. We're also going to

have to give it images. From the actual training set. So our next step is for

the real images within our training set to be passed through and

we want to calculate the probability that the image passed through from

our actual training set is real. So we passed that real image XR before we

were at working with XG, the X generated. And we want to know the probability

that this is actually a real image, and that's going to be the goal

of the discriminator now. We then compute a new L1 loss for

our real image. So again, when trying to predict a real, we're actually going to want our

discriminator to output one. So we're actually going to use this also

to update our discriminator function. And the L1 in regards to XR is going

to be how far off our probability was, what should be as close to 1 as

possible from that value of 1. And we can use that L1 loss to train and update our weights appropriately

within that discriminator network. And we repeat this procedure with new

random noise from the generator each time. And continue until images from

the generator begin to look real. Now we must note that values of losses

from discriminator and generator may still be fluctuating when the generator

is producing realistic images. So we can't use these alone to

determine when to stop training. And there are ways of quantifying image

quality within our generative process, such as the inception score, to help us find where we should

actually stopped training. That's a bit beyond the scope of the

lesson, but I encourage you looking into it if you want to delve a bit

deeper into working with GANs. Now, I want to pull out here a quote

from the original paper on GANs. The generative model can be thought of as

analogous to a team of counterfeiters, trying to produce fake currency and

use it without detection. While the discriminative model is

analogous to the police trying to detect counterfeit currency. Competition in this game that drives

both of these teams to improve their methods until the counterfeits

are indistinguishable from the genuine articles. So we think if the police are too lax, then the counterfeiters won't have

any incentive to create better bills. If the police are too good, then

perhaps they won't have any motivation to build those bills in the first place. So there's going to be this balance

that we also want to take into account. Now, training GANs effectively is highly

dependent on both the generator and discriminator learning at the same rate. So the ability of two networks to learn

is affected by network architectures, different learning rates,

different loss functions, different optimization techniques. And in general, GANs are going to be more

sensitive than traditional neural networks to choices on any of these dimensions. So if you think about what I just

discussed in relation to the dangers of overfitting. In regards to that balance when we talked

about the counterfeiters and the police. We know that in regards to overfitting,

we now have that compounded by the fact that we are working with

adversarial networks with competing goals. So if the discriminator is too good, for

example, wouldn't be able to train at all as fake examples will have too high

of an error with the discriminator. And if the discriminator is too lenient,

on the other hand, there would not be much learned in

respect to creating realistic images. So what you should be doing

to train GANs yourself. Since GANs are fairly new, it'll be that

much more important to actually take a deeper dive into the research and

original papers on the subject, such as Ian Goodfellow's original paper. And some examples of where GANs

are used currently are deepfakes, which you may have seen in

some spoof videos where you can take an existing photo or video and

replace it with someone else's likeness. Age interpolation or making people in

images look older or younger than they actually are, and even taking text and

producing images related to that text. And here we have some links and

some examples of GANs being used for generating fake images of people and

learning lip sync from audio. Now, just to recap. In this section, we discussed an overview

of Generative Adversarial Networks or GANs and the idea of how we can leverage

adversarial examples to generate more realistic samples. We discussed the training procedure for

GANs and how we use specific loss

functions to update the weights. Of our discriminator, and

generator networks, to optimize our model. Then finally we discuss

some key hyperparameters, that govern how GANs train. And how we need to be careful, when fine tuning them due to

the problem of stability, with working with both generator and

discriminators within the same network. Now that closeout our video here on GANs,

in the next video we're going to discuss some other topics that

you should be aware of, when working with deep

learning models in general.

In this video, we're going to touch on

some final additional topics that we should be aware of when working

with deep learning models. Now let's go over the learning goals for

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

discuss computational issues, and with that specialized hardware when

working with these deep learning models. And then finally, just to ensure that

we have Interpretability of our models, we are going to briefly touch on

Locally-Interpretable Model Explanations or LIMEs. To help us gain some Interpretability when

working with these deep learning models that are generally more on

the black box end of a model. Now we're going to jump right

in here to hardware for artificial intelligence,

specifically GPUs. Originally, GPUs were

designed to process graphics. GPUs have become a popular

deep learning workhorse. Reasons being that they feature thousands

of small simple cores specialized for numeric, parallel computations. There's going to be many transistors

dedicated to computation. The GPUs excel at repeated

similar instructions in parallel. GPUs are going to be optimized for

parallel data throughput computation. And major neural net breakthrough since

2012 have been powered by these GPU computations. With that performance with GPUs

has since increased by more than 5x what we talked about in 2012. And once the data is in the GPU memory,

the bottlenecks are actually very small. And if you're looking for

a popular GPU on your own videos, very popular GPU used for deep learning. And just to highlight the main takeaway

from all this is that GPUs are great at parallelization, and

we're going to look into why right now. So the CPU is only made of

a couple dozen cores or cores are the power of the CPU

to do any small calculations. Whereas GPS on the other hand, are made

of hundreds of may be weaker cores, but they're able to run all these

calculations in parallel. So looking at the CPU and

the breakdown, we have the ALU or the arithmetic logic unit. And that's the portion that can execute

simple arithmetic and logical operations. We have control, and control here is to

decode instructions into commands and calls on the ALU to perform

the necessary calculations. We have here the cache,

which serves as high speed memory or instructions can be

copied to an retrieved. And then we have the DRAM,

which is for moral longer term memory. And then if we look at this breakdown for

the GPUs, and all the colors should fit accordingly. We see that for the GPUs we spread

out the control in cache, and they're much smaller for each portion. And for each we have a ton of ALU units

that can parallelize the workload of our operations. Now, GPUs are specialized

processors great for specific tasks such as gaming and

deep learning. CPUs, however, are still going to be

the main computation engines on most computers, as there

are much more versatile. Now, some differences between the two

will include that CPUs have dozens, of cores, whereas GPUs have

thousands of less powerful cores, which allows for a high parallelization. But for GPUs if tasks are not being

parallelized this may not work as efficiently as CPUs. CPUs are going to have fewer ALUs and

a lower compute density than GPUs. And we saw this in the image

in our prior slide. CPUs are lower latency and

have larger cache memory. So compared to GPUs, they can make data

more immediately available to our users. GPUs are designed for parallel tasks,

which is incredibly powerful for a high amounts of matrix computations, such as

what we have in gaming and deep learning. GPUs in general will perform well for

a single instruction performed over a large amount of data

that can be parallelized. Whereas CPUs will perform better for

a wider variety of tasks that don't use as much data and

don't require as much parallelization. Now again,

GPUs are specialized processors. And CPUs, however, are again still

that mean computation engine, so a bit more on that. GPS will have additional overhead when copying data from the main

memory compared to CPUs. CPUs will be better when a large

number of memory swaps are needed, as they have more efficient

memory units than GPUs. GPUs are poor for

tasks that cannot be parallelized. As mentioned earlier,

that is really the specialty of GPUs, being able to parallelize in general. GPUs are poor for heavy processing on

fewer data streams, it's really better for those larger data streams. CPUs excel at serial tasks and

are easier to program. So when there is some type of

ordering to the task to be done and parallelization isn't leveraged,

CPUs will usually outperform GPUs. And finally, most popular

programming languages will compile to machine code to be

run on CPUs by default. Now, on a different note, because this is just some topics

in regards to deep learning. Deep learning models in general

are difficult to interpret. They're going to be many parameters and

complex networks, and connections with in any one

of our deep learning models. So one approach is to generate Locally-Interpretable Model-Agnostic

Explanations or LIME.. And LIME,

treats the model as a black box and focuses instead on the sensitivity of the

outputs to small changes in the inputs. So it'll test how well simpler

models will perform if we look at small changes in feature values for

specific samples. This will be analogous to feature

importance in this respect that LIME will summarize the sensitivity of regression or classification outcomes to

each one of our variables. And it will lean on linear models to

produce the feature importance, and thus nonlinearities and variables that

cannot be perturbed or can't be changed. Such as a binary variables that can really

only take on values of zero and one, and can't just be perturbed slightly or present challenges to

working with this approach. So to briefly recap. We discussed the computational issues on

with deep learning models and how we may want to use GPUs versus CPUs depending

on the model that we're building now. And we also even touched on why it's

important that CPUs are still being used in our current computers. And then finally we closed out what the

Locally-Interpretable Model Explanations. And discussed how we can use that in

order to dive deeper into our model, perturbing some of our examples in

order to understand some of the feature importances within our

deep learning models. Now that closed out our video here,

in our final video, we're going to discuss

reinforcement learning. All right, I'll see you there.