openSAP
Enterprise Deep Learning with TensorFlow

Week 06 Unit 01

00:00:05 Hey, welcome to Week 6 of the Enterprise Deep Learning course. Congratulations, you're on the home stretch.

00:00:10 So today we're going to be talking about some advanced deep learning topics. And for Unit 1, we're going to be talking about how we can generate new images

00:00:18 using generative adversarial networks. Now, what are generative adversarial networks (GANs)?

00:00:25 Well, I think it really helps to break it into the key three words here. So we have generative – so they generate something.

00:00:32 We have networks, with an emphasis on the plural. So there are multiple networks and they are trained in an adversarial process.

00:00:39 So they oppose each other in terms of what they are trying to optimize against. But really what they are and what we use them for is...

00:00:48 we use them to generate new images based on an existing data set. And furthermore, the images that we create with GANs look visually convincing.

00:00:59 Now, why is this? Why do we use GANs for this particular task? So methods in the past that we've used to generate images have generally yielded kind of blurry results.

00:01:10 So they assume there's some kind of Gaussian noise or blur to the image just so that they're not too incorrect.

00:01:16 So before GANs, images tended to kind of stick out if they were generated. They would be kind of blurry and unclear, like that image at the top there.

00:01:26 But that's something that a discriminator should be easily able to tell apart. So these days, classification networks have superhuman performance.

00:01:35 So being able to tell something like "Hey, is this image blurry or not?" is an easy task. So we're going to take it one step further and see how we can use discriminative networks

00:01:47 in order to improve the performance of generative networks. And that is how a GAN is going to function.

00:01:53 Let's talk about these different components of the GAN. The first component, and the most important one, is the generator network.

00:02:02 Now, the generator network is the one that we're going to keep with us when we are trying to generate new images at the end of this process.

00:02:08 So it is the most important network in the equation. So why do we want to use this network?

00:02:15 Why is it so important to generate these images? Well, first of all, this is an important network because it's a hard task to solve.

00:02:24 So classification – usually there's one, two, or maybe even a thousand possible classes. But an image – there is a lot of complexity in what can come out of an image.

00:02:35 And so we want to yield visually convincing results. So we have to architect our networks very specifically in order to do that.

00:02:44 And this is something that is very useful because we can use it for something like inspecting the data in our network, or we can actually apply it to undersampled classes in our network

00:02:55 in order to increase the amount of synthetic samples that we have of a certain class. So if we don't have enough data of a certain type,

00:03:03 we can actually generate it and use it as a form of data augmentation. So the generator network can do a lot of interesting things.

00:03:13 Now, how does it work? Really a neural network can take a mapping from basically anything

00:03:22 and map it to anything else given that the network is complex enough and that you have enough training examples to do it.

00:03:31 What we're going to do here is learn is to transform Gaussian – you know, normal, randomly distributed – noise into an image. Now this sounds crazy,

00:03:42 but it's actually just going to take a bunch of random white noise essentially, and turn it into a flattened out image.

00:03:49 So diving into a little more detail on what that network might look like... We have our generator network.

00:03:55 We're going to feed in just a vector of Gaussian noise. So this could just be a bunch of, you know, grayscale pixels essentially.

00:04:04 And in this particular situation, we're going to use two dense hidden layers and just end up with a flattened out image where every output is a pixel in that image.

00:04:15 So the idea here is that we're going to turn this noise into convincing fake images. The point of the generator is to create images that look like they are real.

00:04:29 Okay? But there's a catch:

00:04:32 It's not actually going to see the training data set at any point. Ever. There's another network here that's very important – the discriminator network.

00:04:44 Even though we're not going to be using this at time where we're actually generating the images, the discriminative network is the true power of the GAN.

00:04:54 Discriminative models have superhuman performance. So on ImageNet, for example, we have achieved superhuman performance with convolutional neural networks

00:05:05 and identifying objects. So this is awesome because being able to pick apart a synthetic image from a non-synthetic image –

00:05:15 that's a binary classification task, that is something that a discriminator should be easily able to do. There's another trick here:

00:05:24 I mentioned that generators never actually see the training data directly. So when we say that these networks are adversarial,

00:05:33 they're not actually enemies per se. The discriminator is actually taking more of a teaching role in this situation.

00:05:39 So what it's going to do is it's actually going to tell the generator network not only "Hey, I know that this image is fake", but "Here is why I think that this image is fake and how you can improve".

00:05:50 So the discriminator actually performs as more of a teacher in this situation. So it's very important.

00:06:00 So how do we actually go about discriminating images? And it turns out that this is just a very classic binary image classification task.

00:06:09 We have to either figure out whether this is a synthetically generated image from the generator network or whether this is a real image from the actual training data set.

00:06:22 So there's really no fundamental difference in the discriminator network, you know, between something like building a cat vs. dog classifier.

00:06:30 So if we actually dive into the architecture here, it's going to look very familiar. We're just going to use a dense feedforward approach.

00:06:36 So we're going to flatten out the image, we're going to use two hidden layers, and we're just going to have a "0/1", "real/fake" output multiplication.

00:06:44 Great. The goal of the discriminator is to look at the generator and say:

00:06:50 "Hey, is this image a generated fake, or is it a real image?" In this way, these networks are adversarial.

00:06:59 So the generator is a generative model that is going to be generating images. The discriminator is going to be the generator's adversary and it's doing to try to pick apart

00:07:09 whether or not the generator is creating convincing fakes. So how do we actually go about doing this training process though?

00:07:18 It turns out that it doesn't happen all at the same time. The networks actually take turns.

00:07:22 And so the way it works is that the discriminator is going to take the first turn. What it's going to do is it's going to get several batches of images from the generator,

00:07:35 which is going to be completely untrained at this point. So the generator is going to essentially spit out a completely noisy image.

00:07:42 We can see an example on the top there. Now what it's going to learn is how to pick those apart from images that we feed in from the real data set,

00:07:52 which we're going to label as "real images". So this should be a fairly easy thing for the discriminator to pick up on right away on its first turn.

00:08:00 Then it's the generator's turn. When the generator is learning, it's actually going to be learning from the discriminator.

00:08:08 Not from the training data set directly, but it's going to be learning from the discriminator, who is going to teach it how to create better convincing fakes.

00:08:16 The idea is that over time the generator is going to get better and better. So by some turn in the middle, we might start to be able to generate images like, you know,

00:08:25 for example, there's a "1" a little bit better. But by the end, we hope that we get something that looks totally natural.

00:08:33 So how do we train the discriminator? How do we take this first step? The first thing that we do is essentially just generate a bunch of fake images,

00:08:42 throw in a bunch of real images, and label them respectively "fake" or "real" – "0" or "1". And this becomes a basic binary classification task,

00:08:51 and we just feed it into the network as such. And we train the discriminator completely independently of the generator after that.

00:08:58 So once the generator has generated its images, it is done for this turn and it is the discriminator's turn to learn from that.

00:09:05 So it is a straightforward classification task. The generator is a little bit more complicated.

00:09:12 And if you can understand this part, you know how GANs work. Okay?

00:09:17 The idea is that we're actually going to combine both of these networks together. And you'll see visually how this is going to work in a minute.

00:09:26 The goal here is to get the generator to learn how to convince the discriminator that the images that it's producing are real.

00:09:35 So that's important. There are a couple of important things to note here:

00:09:38 So first of all, the generator and discriminator combination, you're only ever going to say that these are real

00:09:45 because we want the generator to learn to create things that are real. Second, the discriminator isn't going to be updated in this part.

00:09:54 The simple way of putting this is that it's the generator's turn. But if you're using a framework like TensorFlow or something like that,

00:10:00 it's very easy to take your optimizer and just set a particular section of the network to not update while it backpropagates. And you can achieve this effect.

00:10:10 So the idea is that the generator will be learning to fool the discriminator while the discriminator stays fixed.

00:10:17 But they're going to coexist as one large network. How do we create this network?

00:10:25 We can see that the generator starts from noise and ends up at a flattened out image. And the discriminator starts from a flattened out image and then ends in a 0 or 1 classification task.

00:10:40 If we just merge these two images together, we can see that we get a complete network here. This is actually a full neural network in and of itself, but it's a little bit unique.

00:10:51 You see those grayed out weights there, those ones we're still going to backpropagate the errors through,

00:10:56 but we're not going to update them. The only weights that are going to be updated in this network are the green weights in the generator network.

00:11:03 Okay? The idea is that we are going to pass in a bunch of noisy images to this combination network.

00:11:13 We're going to label them as "real". The discriminator is obviously going to be able to tell that these are fake.

00:11:18 But more importantly, it's going to be able to tell why these are fake, and it's going to convey that information via backpropagation back to the generator network.

00:11:29 So by doing this a number of times, we can actually train the generator to generate better fakes. Great.

00:11:39 There are some challenges in this process because there is a bit of a back-and-forth here. One of the key challenges is that one of the networks can completely overpower the other.

00:11:50 The generator oftentimes will become so good at generating fakes right off the bat that the discriminator actually can't learn to discriminate better over time.

00:12:02 What we actually do, and one of the most important parts of training again, is to give the discriminative network a head start.

00:12:11 Usually it takes the first turn, and it takes a very long first turn. So it gets very good at determining what looks noisy and blurry and what looks real.

00:12:22 And by doing this, we create a much stronger fundamental structure on which to improve our generator network.

00:12:29 Because we have a good discriminator, we're going to get a better generator because of it. Now, there's a second challenge – and this one's a little bit more of a research topic these days.

00:12:39 GANs today are primarily applied to images, and there's a reason for this. We can see that when we merge these networks together,

00:12:48 we're doing backpropagation through the entire process. And what this means is that each weight in the network has to be able to take small updates.

00:12:58 If these are discrete numbers – say, they have to be 0, 1, 2, 3, or if they're categories – there's no way to make small updates to that. Okay?

00:13:07 So generative adversarial networks tend to work best in the continuous case. And images can easily be rescaled to continuous just by turning them from pixel value between 0 and 255

00:13:20 to just a continuous number between 0 and 1, where 1 is the highest intensity of either a red value or a white value,

00:13:29 for example, for an image. There are some extensions of GANs that attempt to address this issue,

00:13:35 but if you are working with discrete data, there are situations where you might consider using other approaches.

00:13:46 And so we can just see that, looking at our losses here, yes, they are indeed continuous.

00:13:51 We can see some convergence happening here, and these are happening in very small increments.

00:13:56 So a discrete world not work for this training process. There's something that I think many of you have probably thought of at this point already,

00:14:09 and it's a very good idea. So one of the first extensions that you can make to just this classic GAN architecture that we've laid out so far

00:14:17 is to take this dense feedforward approach that we've used and use a convolutional neural network instead.

00:14:23 So we know convolution neural networks work great for images, so why not apply them to GANs? This is called a deep convolutional GAN, or a DC-GAN.

00:14:33 So as long as you have the proper training process and properly tuned parameters, this is going to yield substantially better results.

00:14:43 There's another type of network that I think is really cool, another type of addition to the GAN family,

00:14:49 and that is the conditional GAN. And this is where we actually operate in a supervised domain.

00:14:55 So note that in this entire process so far, we haven't actually had to have real labels about the actual content of the image.

00:15:04 So we haven't had to be able to tell whether a face is male or female, or whether they have glasses, or whether they're old or young.

00:15:14 But we want to be able to use this information. So what conditional GANs let you do is actually feed in that class as a vector

00:15:24 alongside your noise to the generator network, and alongside your image to the discriminator network.

00:15:29 And by doing this, when you are at test time, when you are actually generating images, you can feed in whatever label you want.

00:15:37 And the generator, given some random noise, will generate an image of the type you request.

00:15:44 And this is very cool, especially for data augmentation applications. So on our right-hand side over here, we can actually see an image of a woman,

00:15:53 and we are going to shift the class from a "1", which means "woman", all the way to "0", which means "man",

00:16:00 and we are going to switch the age from "old" to "young" as we go from the top left to the bottom right. And so we're actually going to see a shift in this person as we move across these classes.

00:16:15 This is really neat. So to sum it up, generative adversarial networks are a way of generating visually convincing images

00:16:26 based on an existing training data set. They consist of two competing adversarial networks,

00:16:32 so there's a generator network and a discriminator network. The generator network tries to create better fakes,

00:16:38 and the discriminator tries to figure out whether these are fake images or real images that it is being fed. So they have adversarial objectives.

00:16:49 That said, the discriminator is going to be teaching the generator how to improve. And the way these networks are trained is that they take turns.

00:16:58 So it's the discriminator's turn first and the generator's turn second, and then they alternate back and forth over time until you converge.

00:17:07 So what we're doing here is we're generating images using an adversarial objective, so the two networks have opposing objectives.

00:17:14 And there are multiple networks here. So you have generative adversarial networks (GANs).

00:17:22 That pretty much sums up what a GAN is at a high level, and some of the applications for them. So on our right-hand side here, we can see one more example that I think is really cool.

00:17:34 When you feed in labels to one of these conditional GANs that we talked about on the previous slide, they're actually compositional.

00:17:42 So if you actually want to add in "old" to a label, or "blond", or "bald", you can actually do things like that and get visually convincing results.

00:17:51 So we can see right here the top two images. If we take the original image and add a bit of the "young" class to it,

00:17:59 it actually turns it to a slightly younger version of the image. Or we can turn the hair from blond into darker hair.

00:18:07 So this is just one cool application, and I'm sure you can think of many more. Next, I'm going to pass it over to my colleague Miro

00:18:15 who is going to talk about some more advanced topics in deep learning – thank you for joining me.

Week 06 Unit 02

00:00:05 Hi, my name is Miro Enev, and I'm a Senior Solutions Architect at NVIDIA. Today, we'll be talking about reinforcement learning.

00:00:15 Reinforcement learning is one of the three buckets of applications in which you can describe machine learning applications.

00:00:23 In practice, these often overlap, but in order to better understand the context, let's look at them each in turn.

00:00:31 At a high level, the largest difference between them is the way in which the data set is structured. In supervised learning, we have the luxury of having a data set

00:00:42 as well as what an expert believes is happening during the samples presented. So in some sense, this is equivalent to having a supervised signal or a label.

00:00:56 In the case of unsupervised learning, you don't have the luxury of having the perspective of a trained expert from the field.

00:01:04 So our data set is just that – a bunch of data with no labels or annotations. And in the last kind of application for machine learning, we get to build our data set.

00:01:15 Initially, we have nothing, but as an agent with an opportunity to interact with the environment

00:01:20 and observe the changes that result as a course of its actions, we can build up and accumulate experience.

00:01:29 So let's quickly look at how these differences can affect the nature of the data that we have access to. In the case of supervised learning, as we mentioned briefly,

00:01:42 we have both samples of data, such as these numerical digits that were captured from handwritten checks. And for each of these digits, a human expert came along and provided a label.

00:01:57 So in the case of the 8 that you see here, we might have something like a number "8", "eight" written out,

00:02:04 or, as it's known, a binary representation, encoding in which position this data sample is from all the available positions, also known as a one-hot encoding.

00:02:17 So we have labels associated with our examples. And we may have something on the order of 100,000 examples.

00:02:25 That's around the margin for large supervised data sets. In some extreme cases, we've had so much human effort go forth in creating labels

00:02:36 that we have been able to reach the million mark with a data set like ImageNet. However, for the most part, supervised learning is tricky to do,

00:02:47 mostly because the desired target outcomes that a human expert would provide for each of the samples are hard to get and have high costs associated with them.

00:02:58 However, whenever such examples are available, lots of progress can be made. So moving on to the next application category – unsupervised learning.

00:03:11 Here the data set is not constrained in size, and partly because we don't have to get an expert's opinion about what's happening.

00:03:22 And with the rise of things like sensors, embedded devices, and mobile phones capturing digital sequences from Internet traffic,

00:03:33 we have an exponential growth in the amount of unsupervised data. And again, here we just have something like a long time series tracking some signal,

00:03:43 or large sequences of gene tagging. And we don't have the luxury of knowing what particular protein has been coded here,

00:03:55 or why this time series is behaving the way it is. We just get to see lots and lots of this kind of data.

00:04:04 And there are many interesting questions that we can ask with this kind of information. And we'll dive into that in a subsequent unit on unsupervised learning.

00:04:17 But for now, let's just recall that the main differences are that so far in supervised we have a data set composed of samples and labels,

00:04:26 and in unsupervised learning, we don't have the luxury of labels, we typically have large amounts of data.

00:04:33 So now let's turn our attention to reinforcement learning. So in the case of reinforcement learning,

00:04:40 we are introduced into a feedback loop with the environment. And when I say "we", this is a machine learning agent.

00:04:51 And so the machine learning agent gets a chance to build experience through its interactions with the environment. And more formally, each experience can be described as:

00:05:05 the current state of the environment – essentially what we observe the world to be. Also the action that we take in this particular environment state –

00:05:16 typically this action is chosen so as to maximize our reward. This is a somewhat abstract concept at this point,

00:05:28 but as we'll see in subsequent slides, one simple mapping to reward could be the score of a video game.

00:05:37 So given the current state, can we choose an action to increase our score? And so to complete the description of a single sample in our data set for a single experience,

00:05:48 the remaining items that we need are the state of the environment as a result of the action that we took and what the reward was that happened in the time step following our action.

00:06:01 And so, armed with this information, we can now progress and improve our agent using the tools in reinforcement learning.

00:06:14 So just to really quickly recap this information, and so that it fits with the graphic that's shown here...

00:06:23 We're building experiences, and at each time step we get to execute an action, labeled as "A_t", where "t" denotes the current time step.

00:06:35 So at the same time step, we also get an observation from the world and a reward. And then the environment...

00:06:47 gets a change as a result as a result of our choices, and so it receives our action and produces the next time step,

00:06:55 which isn't necessarily shown in the diagram. Essentially, it emits the next observation that's available to us,

00:07:01 that we will subsequently receive. And it will also emit a reward at the next time step.

00:07:08 So that's the essential description of the feedback loop required for reinforcement learning to apply. Now, what if we take this and apply it to a domain such as video games?

00:07:25 This is a great application that's been very successful. Part of the reason why is because, as we mentioned earlier,

00:07:33 we have a very straightforward mapping from an action to a reward in this world, where, again, the reward is the numerical score presented by the target environment.

00:07:48 And so in this example, we're able to sort of fill in all the blanks required to have reinforcement learning. And instead of a human in the loop here,

00:07:58 as our agent we have built a neural network, specifically the raw pixels coming from an Atari game world are sent into some convolutional layers.

00:08:09 that get to build features about what's happening. And then those are subsequently sent to some fully connected layers,

00:08:17 and those fully connected layers get to decide how good of a state we're in, which is important in order to improve the agent,

00:08:27 and also produce an action where this action is any of the possible combinations of directions and button presses in the Atari game world.

00:08:38 And so, now that we've replaced the agent with a deep learning network which can interpret screen pixels and choose a game action,

00:08:49 we can allow this agent to build experience by trying different actions and different states and learning the appropriate reward mappings.

00:08:59 And as we allow this agent to train over long periods of time, it actually gets surprisingly good at these games.

00:09:09 One example of this, which I have personal experience with, is training a network like this to play Atari Boxing.

00:09:20 And we used quite a simple network, just two stacks of convolution and pooling layers, with a single thousand-neuron flat dense layer connected to action traces and value expectations.

00:09:42 And using this network, which had maybe, let's say, on the order of 100,000 free parameters, we were able to train a boxing agent such that it was able to outscore the AI,

00:10:02 and in many cases, towards the end, not even allow a single hit. And one interesting aspect of games like these is that the input...

00:10:16 is being sort of rolled out into time to allow for the network to learn dynamics. And one formulation of this essentially binds multiple frames together.

00:10:29 So four successive frames become a single input to a network. And so an observation really is four frames from the environment.

00:10:37 We could have structured this differently. One possible alternative view here could have been a single frame connected to the network that we've shown here.

00:10:47 But this would network subsequently connect to our recurrent layer prior to making its action and value decisions,

00:10:55 which would then allow it to also reason over time. Nonetheless, this is the basic environment in which these agents are trained.

00:11:05 And in practice, we see really great success whenever such reinforcement learning applications are built in industry.

00:11:17 One example of this is robots that are used to separate items into different categories inside of warehouses. Multiple tasks that are similar to this would fall under the umbrella of "bin picking".

00:11:34 And in one case, a Japanese bin-picking robot was self trained through trial and error in determining where its magnetic gripper had a high probability of being sent to using a separate control algorithm,

00:11:52 so as to make a successful extraction of the part from the disordered pile that's presented in each bin. After eight hours of training on this task,

00:12:05 the robot was able to provide these probability maps that you see here in the figure, which were matching human performance.

00:12:15 And so if you just take the top most probable sample from this distribution, you essentially know where to go next.

00:12:27 And this was all done within eight hours, and the folks who were doing this essentially said that beyond eight hours the performance plateaued.

00:12:38 But again, you're matching or in some cases exceeding human performance. Exceeding it not only if for the reason that you have a robot that doesn't get tired and can work throughout the day,

00:12:52 and doesn't lose concentration because of sickness or family events, and so on. And so whenever such a paradigm can be applied in industrial settings, it's extremely powerful.

00:13:06 Google also recently showed that they were able to reduce their power bill using a reinforcement agent that was periodically enabled to control the state of their data center cooling.

00:13:18 So many such applications, I think, will come in the future. But hopefully now you have sufficient background to appreciate

00:13:26 what the requirements are in order for a reinforcement learning agent to operate. Thank you.

Week 06 Unit 03

00:00:05 Today we'll be doing a deep dive on unsupervised learning as part of our Week 6, Unit 3.

00:00:13 To recap, unsupervised learning is one of the three broad contexts into which machine learning applications can be described.

00:00:22 We already talked about reinforcement learning in the previous unit, and we briefly described the differences between supervised and unsupervised learning.

00:00:29 Let's recap those differences before diving in. At a high level, the thing that makes unsupervised learning stand out from the other two alternatives

00:00:39 is that data sets in the unsupervised learning case are much larger, and we don't happen to have domain expertise which has been applied to that, to the data.

00:00:49 So we don't happen to have labels or annotations associated with each example. This is different from supervised learning, where we happen to have both the data in its raw form

00:01:04 as well as an expert's opinion about what's happened. As is seen in the slide, this can be something like handwritten digits that were written on check back in the 1980s,

00:01:20 which some poor grad students had to come around and label. And this particular data set, known as MNIST,

00:01:29 is one of the most widely used tutorials/data sets for deep learning. And here, if you want to experiment with it,

00:01:42 you will able to train a neural network to recognize these digits, and its capacity to do this recognition is based on the fact that it has a large number of examples

00:01:52 from which to learn the mapping from the label to the raw data. This particular data set, the one we're referencing, the MNIST data set,

00:02:01 has 70,000 labeled digits in it. And this is fairly typical of most supervised learning cases,

00:02:09 which, on average, are in the tens of thousands, hundreds of thousands, and very rarely in the millions of examples.

00:02:19 This is different from unsupervised learning where, just to reiterate, we don't happen to have any labels.

00:02:26 No humans have had the capacity to come around and provide their opinions of what's happening in the data. But conversely, we happen to have much larger data sets.

00:02:37 This is partly due to the accumulation of data that we're seeing because of embedded sensors, because of our smart devices that we carry with us everywhere, capturing the world,

00:02:48 as well as the fact that pretty much everything these days can be digitized until the flow of information is such that it lends itself to being recorded in digital format as well.

00:03:01 And so, in the unsupervised learning domain, we have these massive data sets, typically millions of samples, tens of millions of samples, hundreds of millions of samples.

00:03:12 And often, the size of our data set is so vast that we can only really analyze the last, let's say, month of data. And unlike in the supervised case, where we have a very targeted approach to trying to accomplish some objective,

00:03:31 such as "Let's classify this digit", in the unsupervised learning case,

00:03:39 the types of questions that we can ask are very different. And so, one of the most important applications to unsupervised learning coming from the deep learning world

00:03:53 is the capacity to detect anomalies. If we look at a deep learning network capable of detecting anomalies, it looks something like this.

00:04:04 This is known as an autoencoder network
– it kind of looks like a bow tie. And just as a preface – why is anomaly detection important?

00:04:13 Anomaly detection is very important and can be applied to almost any field. If you are interested in financial instruments and financial markets,

00:04:28 anomaly detection can be used to detect fraud. It could be used to detect arbitrage opportunities.

00:04:36 If you are in the IoT space, anomaly detection might be useful for detecting power failures or water leakage. If you are in the industrial smart warehouse space,

00:04:52 anomaly detection might be used to detect... something like a machine failing that would require prognostic maintenance.

00:05:03 So really, the applications of anomaly detections are very numerous. And for almost every domain, there's a use case that can benefit from being able to flag these things that are happening

00:05:15 that are outliers and, in many cases, deserve our attention. So how do we do this with deep learning?

00:05:24 Well, we've got a network like this, where on one side you see the input layer of neurons. And each of these neurons is connected to a raw data sample.

00:05:37 A raw data sample can come from a window of time series data, from a sensor. It could come from text data of a paragraph.

00:05:47 It could come from the pixels of an image. It could be of any form.

00:05:53 The key idea behind this architecture is that this original input representation is forcibly compressed, such that each layer that you see here gets narrower and narrower

00:06:06 until you get to the most constrained layer, which is also known as the bottleneck. And the bottleneck is having to summarize the representation of the input,

00:06:18 and subsequently provide that representation to a decoding stage of layers, which are having to take that reduced dimensionality structure

00:06:31 and go back out to the original dimensionality. And so, in some sense, this network is doing an encoding from high dimensionality to low dimensionality

00:06:41 in order to produce the bottleneck summary, and then doing a decoding to go from that low dimensional summary back out to the original dimensionality.

00:06:51 And so, if this network is provided with a sufficient number of examples, it can use the difference between the input and the reconstructed output from the decoder.

00:07:03 And using this difference, it can tune the parameters of the network, in other words, tune the weights of the network,

00:07:10 such that the difference between the input and the output becomes smaller and smaller, even though the network is forced to summarize along the way.

00:07:20 And so, once these networks are trained, they become really great at anomaly detection

00:07:28 because any time you have a large gap in the input and the output, in other words, a large reconstruction error,

00:07:36 this is indicative of an example that somehow can't be described by the statistics of everything else that has happened in this domain. For whatever reason, this example can't be accurately summarized and reconstructed.

00:07:52 And so, it's capturing behavior that's outside the norm. And this is a very important tool because we have built a learning machine

00:08:03 that is capable of alerting us to these scenarios, these anomalies, and generating red flags for us without anyone coming along

00:08:14 and necessarily building labeled examples of what anomalies mean beforehand. So this is one of the main advantages of this approach.

00:08:26 In order to instantiate this and make it a little more practical, let's consider how we might apply this technique to a time series problem.

00:08:36 So imagine that we have some time series signal that spans many years – how might we build an anomaly-detection-based deep learning model for it?

00:08:48 The first thing that we probably want to do is determine how our network is going to look. So another way to think about this is to say:

00:09:00 "Well, in my particular domain, what is the amount of time that I need to look at, at one sitting, in order for me to consider whether or not an anomaly is happening?"

00:09:12 So if this was happening inside of a smart warehouse, the time horizon that you might consider might be on the order of hours.

00:09:21 If this was some kind of a historical trend for financial stock data, the amount of time you consider may be on the order of years.

00:09:30 And so, this is one of the most important architectural choices that you make in these kinds of problems – choosing the time horizon, or the size of the window, if you will,

00:09:41 that determines the size of the input to the network. In this particular case, let's assume that we have data that is captured on monthly intervals,

00:09:52 and here we'll just take one year as our window size. The next thing we do is build a data set of these windows,

00:10:01 where each successive window is very much like its predecessor, but it captures one new sample that the predecessor has not seen.

00:10:11 And so, if one window spans from 0 to 10, the next one will span from 1 to 11,

00:10:19 and so on and so forth. And by having such overlap, the windows guarantee that no anomalous event will occur at the edge of one window,

00:10:30 such that half of the anomaly is happening at the edge of one and the other half of the anomaly is happening at the very start of the next,

00:10:37 and so by being split in this way, can't really be seen by any one window. Well, if we have sliding windows, that won't happen

00:10:44 because essentially, we're almost guaranteed that one of those sliding windows will capture the anomaly in its full glory. So this preprocessing step may seem tedious,

00:10:56 but it's really critical in these kinds of applications to choose the right time horizon and to make sure that you allow this time horizon to propagate in a sliding window fashion
so that you have overlap.

00:11:10 And once we've generated the data set in this way, we're ready to start feeding it into the network. What we do next is we provide these sliding windows as inputs,

00:11:23 which you can see here at the bottom of the slide. And we initially set the random weights of the network and observe what it produces.

00:11:34 And each time that we provide one of these inputs to the network, we take the difference between that input and the network's reconstruction,

00:11:43 and use it as a learning signal for the network to update its parameters. If we do this enough times,

00:11:51 the network becomes very good at being able to take in our input, summarize, and then subsequently produce the output reconstruction.

00:11:59 And so, here we're seeing the behavior of the network after it's trained. And we can see that there are some differences between the input and the output reconstruction,

00:12:09 but they're fairly minimal. And so, this is within the realm of normal deviation or randomness.

00:12:19 We now have trained an anomaly detection network, specifically an autoencoder deep learning network.

00:12:28 And what we can do next is we can take all the examples in the data set, or potentially future examples that we present as input to this network,

00:12:39 and think about what kinds of things are happening. As an example, we might feed in something to the network,

00:12:50 realize that it's producing a high reconstruction error, flag it as something that we need to consider,

00:12:56 and then think about how we might interpret that red flag. One of the best ways that we can do this kind of interpretation is to look at the bottleneck layer – the most constricted layer –

00:13:09 and take the activations of those neurons while this red flag is being activated. Because the bottleneck activation neurons are so few to begin with,

00:13:18 they're very close to a dimension that can be visually interpreted by humans. So what we can do is we can take those, you know, 3, 5, 10, 15 neurons in the bottleneck, depending on your architecture,

00:13:33 and apply a mathematical dimensionality reduction technique, like principal component analysis, to take those to a two or three-dimensional space where we can then do some kind of clustering

00:13:47 and use our intuition to try to interpret what these red flags are, what the dense regions of red flags are, and whether they are somehow capturing anomalies of a particular type.

00:14:00 And that's actually exactly what happened in this research that was done by United Technologies when they took a NASA data set, applied these techniques,

00:14:11 and were able to determine instances where ball bearings were starting to flake, or rotational motors were starting to tear down,

00:14:20 and a space-flight data set. So just to recap:

00:14:28 In the unsupervised learning domain, we have very large data sets. We don't happen to have the luxury of having labels.

00:14:36 However, we can build really interesting learning tools that allow us to think about the structure of our data, as well as to detect anomalies and interpret what those anomalies mean. Thank you.

Week 06 Unit 04

00:00:05 Today we'll be talking about applications of deep learning to mobile devices. This is Unit 4, Week 6.

00:00:15 When we consider what it takes to take deep learning to mobile devices, at a high level, the workflow breaks out into the following substeps.

00:00:27 First, we do data engineering and prepare the data set. We have thought hard about a business problem, we've determined how to collect data,

00:00:38 we've collected it – now we're ready to get it to be prepared for ingestion for a machine learning application. So this is where we might normalize the data so that it all fits inside the same range.

00:00:52 We might apply mathematical transformations that can augment the data. This could be things like slight rotations, mirroring images, or things of this nature,

00:01:02 in some sense to squeeze the most out of the data that you have. You might also spend some time thinking about how to handle missing data at this point.

00:01:11 So do you leave a blank? Do you try to create a straight line from the last time stamp that you had data for to the current one that you have data for,

00:01:21 and in some sense interpolate the missing data? These are all questions that are answered in the preparation stage.

00:01:27 But assuming that that hurdle has been passed, the next thing we think about is: "What is the best architecture?" Essentially, "What is the best neural network model for our problem?"

00:01:38 And since we don't we have a deep theoretical understanding of what neural network shape is optimal for the problems that we face, we have to experiment.

00:01:49 And so, this is typically where you apply a lot of compute. And oftentimes, this is done in cloud environments

00:01:57 where you instantiate many experiments with many different architectures and many different parameter settings, and find the one that seems to perform the most robustly on data that it hasn't necessarily seen in training.

00:02:13 And so, this experimentation stage, once that's completed, we now have a set of models, or a single champion model,

00:02:22 that is ideally suited to a task that we care about. The next challenge becomes: "How do we deploy this model?"

00:02:31 This could be introduction into a production system, or it could be something like deploying to an embedded device or a mobile device.

00:02:41 And so, this last step is the one that we'll be focusing the majority of our conversation around today, specifically: "Now that we have a trained model, how do we deploy it to mobile devices so that they can run effectively

00:02:55 in constrained memory compute and energy utilization settings?" So if we look at that a little more closely,

00:03:05 modern mobile devices are, in some sense, limited compared to the cloud, but at the same time they're extremely powerful.

00:03:16 You know, a modern mobile phone is equivalent to a laptop that may have come on the market less than a decade ago. So they have fast compute units, they have large amounts of memory.

00:03:31 However, the bottleneck typically ends up being the lifetime of the device in terms of how much energy is required to run our model.

00:03:42 And so, the challenge becomes: "Can we be effective at running our AI deep learning application while minimally impacting the expected lifetime of the battery charge?"

00:03:58 And so, let's investigate that a bit further. If we look at the most expensive aspect of running inference on a mobile device,

00:04:11 it's not the actual number crunching required to compute the activations of the neurons, but rather the movement of the data,

00:04:21 the querying and the response that happens between the CPU and the memory. And specifically, each query and each response is energetically expensive.

00:04:33 And it's not so much that each query and response is expensive, but it's just that we end up doing so many of them that, in bulk,

00:04:40 they end up becoming a burden on our battery. And so, if you look at a typical convolutional network that you might apply to a problem,

00:04:51 such as determining what objects are in a scene that a mobile phone is being pointed at, such a network may have millions of parameters.

00:05:01 And in order to be able to compute data through this network, this may require multiple passes of communication between the CPU and the memory.

00:05:14 So the first pass would require getting the data from the inputs, you know, the pixels essentially,

00:05:23 and maybe the next few layers of the network. And all of those weight parameters can be queried in a single batch

00:05:31 from the CP registers reading from the memory. But as soon as those first few layers are computed,

00:05:38 we have essentially saturated the capacity of the compute unit that it needs to query the next set from memory. And so, a large, modern network may have between 10 and 20 queries

00:05:53 that it needs to do in batch between the CPU and memory, depending on architecture. And so, one way that we can extend the battery lifecycle of the mobile unit

00:06:07 is to find ways that we can reduce the number of these fetches that are performed by the CPU in communicating with the memory,

00:06:17 in order to compute the activations of the network. The first optimization that can be applied in this setting is known as "weight pruning".

00:06:28 And weight pruning is actually something where we take inspiration from biology. And this is something that's happening aggressively when we're young.

00:06:39 At a year old or two years old, a child will have very large amounts of synaptic pruning occurring. And even throughout adulthood, they will have something like 200 synapses that get lost on a daily basis.

00:06:55 But generally speaking, this is okay because we have so many billions and trillions of connections between our neurons such that the loss of a few is actually helping us become more effective.

00:07:12 So what actually happens when we remove weights from a network? That's the definition of pruning – the reduction of the number of weights.

00:07:22 In some cases, we find that removing a few of the weights, or even a large number of the weights, doesn't degrade the performance of the network significantly.

00:07:35 So one way to do this in practice is to train a large network, and then start to randomly pick weights and opportunistically set them to 0,

00:07:46 or just drop them from the network. And in doing so, we find that as long as you're continually streaming training data through this network and monitoring its performance,

00:07:59 oftentimes you can get away with significant reduction – you know, 10 to 20 times reduction in the number of weights that you need

00:08:07 in order to have high accuracy for your particular problem. And so, this is something that we find in practice, and has been put to use empirically in many situations.

00:08:20 One of the best ways to reduce the battery consumption for mobile devices is to do this pruning process while considering which layers end up being the most expensive from an energetic perspective.

00:08:37 So specifically, you can build a model that predicts how many successive memory fetches will be required for your network,

00:08:48 and try to target those layers for pruning which are responsible for the most communication overhead between the CPU and memory.

00:08:58 And typically, this is the fully connected layers. And by targeting those for the pruning process,

00:09:06 you can end up getting a tremendous decrease in both the number of parameters required while slightly degrading accuracy

00:09:16 and also the amount of energy required to be able to compute to the network. The next optimization approach that we can consider is known as "quantization".

00:09:29 And quantization is the concept that we can actually use fewer bits to represent data. Again, because we're now in the state where we have spent a lot of heavy-lifting computing to determine our best model,

00:09:49 we can take that best model that we've trained and reduce the precision with which the weights and the activations of the neurons are expressed.

00:09:58 And so, typically during training, it's critical to have high precision in the representation of weights and activations – something like 32 bits or 64 bits is pretty standard.

00:10:11 And this is required in order to be able to capture the infinitesimal, minuscule changes in the parameters that are being produced by each learning example – essentially, these are the gradients in backpropagation.

00:10:26 While it's critical to express those in high fidelity during training, once we have trained the network, running new data through that network

00:10:35 does not require that same fidelity in the expression of the weights and activations. So what we find is that we can actually take those high-precision values

00:10:44 and take them into much lower-precision representations – something like 8 bits – and shrink the dynamic range significantly.

00:10:56 And this is visually demonstrated on this slide, where we see the openSAP image shown in 32-bit floating-point format,

00:11:07 as well as in integer-8 precision below. You can see that much of the same shape is retained.

00:11:14 Some of the colors actually get mixed into the black – so gray and black tend to meld. And that's part of the loss that you end up getting because you squeeze the dynamic range so much.

00:11:24 But nonetheless, most of the important bits come across, and that's the same kind of thinking that happens in this quantization.

00:11:31 So by expressing the model in this reduced format, you get a significant reduction in the amount of memory that is required.

00:11:41 And also, you can get a lot more throughput in each of the batch transfers that are happening between the CPU and memory. Again, reducing the need to do those computationally – or rather energetically – expensive transfers.

00:11:56 One note to keep in mind is that there's an additional quantization step that needs to occur during preprocessing and postprocessing.

00:12:09 Essentially, you need to make sure that your inputs are quantized and your outputs are dequantized. But in many software packages these days, this is all handled for you.

00:12:22 So just to recap: When we're approaching this workflow knowing that we're going to be targeting a mobile device,

00:12:32 we essentially start by defining our model architecture in, typically, some high-level language like Python, which would subsequently instantiate a compute graph.

00:12:45 And we stream our training data through this compute graph to optimize it and to tune it to the particular learning problem.

00:12:52 We may do these steps with multiple experiments at the same time until we arrive at one particular architecture we really like.

00:12:59 Once we have determined that model that we're going to be deploying to production, we will apply these optimizations that we just discussed,

00:13:06 so essentially these pruning and quantization steps. And once we have done that, we have reduced the number of weights we have to represent

00:13:14 and we've also reduced the way in which we represent each of the weights that we've kept. And so this has led to a significant improvement in compute time, memory footprint, and energy requirements.

00:13:27 And so, we can package this all together into a serialized binary, let's say, for runtime deployment on a mobile platform.

00:13:37 And in doing so, we now have completed the workflow, and we have a successful deep learning application running on mobile devices.

00:13:49 Just for reference – there's a growing number of tools that are available for helping with these kinds of optimizations, essentially helping you go from a trained model to something that can be deployed on a mobile platform.

00:14:03 At NVIDIA, we produce TensorRT – "RT" stands for "real time". And this platform has many optimizations built into it.

00:14:11 There are similar offerings being built by the folks at Google and Apple, as well as Amazon and some research institutions.

00:14:20 This is an incomplete list, and the number of solutions keeps changing frequently. So if you're interested in this space, keep track of these software packages,

00:14:32 as well as everything else that's on the horizon. Thank you.

Week 06 Unit 05

00:00:07 Welcome to the final instalment of the openSAP course on Enterprise Deep Learning with TensorFlow. What an amazing six weeks it's been together!

00:00:16 Let's have a quick recap of the subject matter areas that we covered. In the first week, we got started with deep learning: a historical overview of the fields,

00:00:26 starting with the very first artificial neural networks and the trajectory to the complex structures of today.

00:00:32 In the second week, we dipped our toes into Building TensorFlow Applications, with few layers of artificial neurons that can already solve some amazing problems

00:00:41 with a very few lines of code. Week 3 was about our first deep dive into deep learning proper,

00:00:47 with sequential models aimed at natural language text and other sequential data types. The following week was all about Convolutional Networks

00:00:55 that explored the locality in time or in space of a signal which are essential for record-beating image processing, video processing,

00:01:04 and other types of sensor signal processing performance. The second-to-last week, Industry Applications of Deep Learning, showed us many exciting use cases,

00:01:14 whether it's in line-of-business functions like finance or in industry-specific topics where deep learning can change the game for enterprises around the globe.

00:01:22 And, last but not least, we covered some pretty advanced topics in deep learning that are leading edge in this final instalment of this series.

00:01:31 So what you should take away from this is that deep learning is not a branch of applied magic or dark sciences. It's very much an engineering discipline where we look at practices that have worked in the past,

00:01:45 practices that have worked in other cases, and we combine and apply them to solve real-world problems

00:01:52 until they are solved well enough to matter to businesses in the real world. So this is an engineering skill to solve practical problems,

00:02:02 not so much a set of theories or a set of abstractions. And you have already gained first-hand experience with the engineering skills required

00:02:12 to solve these kinds of challenging problems with a few lines of code and a large set of data so that you can transform the way business is done.

00:02:22 Practice makes perfect in this. And should this have sparked the appetite for more in you,

00:02:28 let's have a look at a few of the learning resources that our team and partners have found most relevant among the multitude of offerings that are out there on the market.

00:02:39 We like the deeplearning.ai course at Coursera, which is another massive open online course that helps you to double-click and go even deeper.

00:02:47 We also like the two-grade CS courses at Stanford University which focus both on text and on image processing with convolutional and with recurrent networks.

00:02:59 We're big fans of the Deep Learning book. A free version of this is available on the Web –

00:03:04 you can also choose to purchase a printed and bound version. And please head over to the SAP Leonardo Machine Learning microsite at sap.com/ml

00:03:14 to learn more about what we're doing to bring these capabilities to enterprise customers and about the leading enterprise applications offered by the largest enterprise vendor that is

00:03:25 Please stay tuned for the forthcoming openSAP course on Leonardo Machine Learning, which will help you deploy models you're building with TensorFlow on a scalable cloud infrastructure,

00:03:38 and combining predefined models with customized models and your own models to solve large-scale problems and string together solutions along an entire process value chain.

00:03:50 This was Enterprise Deep Learning with TensorFlow on openSAP. My name is Markus Noga, and the Machine Learning team at SAP and our partners

00:04:02 wish you the best of success for the upcoming exams. Thank you for staying with the course.

www.sap.com