openSAP
Enterprise Deep Learning with TensorFlow

Week 01 Unit 01

00:00:08 Hello and welcome to the openSAP course on Enterprise Deep Learning with TensorFlow. My name is Markus Noga, and I'm head of Machine Learning at SAP

00:00:17 at the Innovation Center Network. Our great team and our many great external partners

00:00:22 will be walking you through the installments of this course. We're working together to bring the intelligent enterprise to businesses worldwide.

00:00:31 Let's have a look at what the intelligent enterprise is about and what it means to businesses.

00:00:37 In the past, enterprise systems largely served the function of guiding transactional flows and workflows. Humans were doing most of the work while the machines helped enforce divisions of labor,

00:00:50 separations of concerns, four-eye principles, and other essential process safeguards, as well as adding up numbers.

00:00:58 Increasingly, with digitization, we also have all content about the business transactions available in the very enterprise systems themselves,

00:01:06 including both structured and unstructured content, like natural language text documents, the spoken word, images,

00:01:14 videos, Internet of Things sensor data, and more. And the fact that all this data is available in systems

00:01:20 means that we can learn from every business transaction, from every repetitive workflow.

00:01:25 We can apply machine learning and deep learning to create models that automate all common business functions. And this is what the intelligent enterprise is about.

00:01:35 Imagine a world where all repeating, repetitive, mind-numbing activities in a business are increasingly performed by machines, leaving humans to focus on the tougher, the harder, the more challenging,

00:01:48 the more personally rewarding, and professionally-growing kinds of problems to be working on. Would that not be a fantastic state of affairs?

00:01:57 The key ingredient for realizing this is deep learning. Deep learning is a subfield of artificial intelligence, machine learning, and neural networks.

00:02:09 In essence, it's about an architecture inspired by the interconnected neurons of the human brain, which collaborate together and work in many, many layers of processing steps

00:02:22 to solve the most complex of real-world problems with seemingly simple individual operations. This is what deep learning mimics and is inspired by,

00:02:33 but in no way limited to. And with today's deep networks, of hundreds or more layers of artificial neurons,

00:02:42 we're able to tackle natural language text processing, speech processing, image classification,

00:02:50 image detection, video processing, and more, at a level that's on a par with human performance,

00:02:58 or in many cases already slightly exceeding human performance on simple tasks. TensorFlow is a library that helps us do this in the most efficient manner.

00:03:10 Because TensorFlow allows us to describe neural network structures the way that engineers think about them when solving a problem.

00:03:19 And the task of running these networks most efficiently on the best available hardware and transforming them into instructions that train and apply networks in the most effective manner

00:03:30 is something we no longer need to worry about. With TensorFlow, we can define networks once and run them anywhere,

00:03:38 train them anywhere, and deploy them anywhere, and work in almost any language.

00:03:43 It started out with Python – Python is also what we're using intensively – but support for many other language families is growing,

00:03:50 and support for any kind of neural network architecture is growing. Development of TensorFlow is led by our great partners over at Google.

00:04:00 At SAP, we use TensorFlow as one of the key elements in the SAP Leonardo Machine Learning architecture. With SAP Leonardo, we aim to make machine learning incredibly easy to consume

00:04:14 for businesses around the globe. We're application-first, and we're also developing a library of reusable services,

00:04:23 both on the business level and on the functional building block level, that helps real-world application developers and practical engineers

00:04:31 to solve problems in unstructured data with deep learning and machine learning. When doing this, we focus strongly on business outcomes.

00:04:42 It's not about solving a theoretical problem; it's about driving revenue by improving sales targeting and improving sales execution.

00:04:52 It's about reimagining processes end-to-end, with a little digital intelligence at every step,

00:04:59 improving on speed of execution, throughput, and cost trajectories. It's about more rewarding roles for employees who are freed of routine tasks

00:05:09 and focus on those activities that help them grow personally and professionally. And it's about happier customers,

00:05:16 because imagine the last time you had a truly great customer service experience while waiting in a helpline or typing into a chat.

00:05:24 Chances are your experiences were less than optimal because humans have to take care of all tasks today rather than focusing on the tough ones

00:05:32 and leaving the easy bulk of the work to the
machine. This is what we can achieve by bringing machine learning to the enterprise in an easy way.

00:05:40 And last but not least, for businesses around the globe, this also unlocks the potential to drive product and process innovation

00:05:49 in a way that's never been possible before. SAP is ideally positioned to help you with this

00:05:56 because 76% of all business transactions touch an SAP system. And with data being the key fuel and the key ingredient

00:06:05 for successful machine learning applications in the enterprise, we're ideally positioned to help bring that intelligence into all steps of every business process.

00:06:16 The objectives for this course are to take deep learning and machine learning from a dark science – a variety of magic, or organic chemistry with many, many experiments

00:06:29 that you do and fail, and improve on, and fail again until it works – to a concrete engineering discipline for real- world software and application engineers,

00:06:39 and a toolbox that they can use to solve common business problems with machine learning. You will learn how to train, build, and test machine learning models for the enterprise

00:06:51 after having identified the right use cases to deploy deep learning and the right use cases where not to do this.

00:06:58 You will gain hands-on knowledge of how to solve these problems with TensorFlow, and you will get to hear best practices and insights from deep learning experts,

00:07:08 as well as about many, many interesting industry applications of this technology in every line of business and every industry.

00:07:20 So, this course is structured in six modules and a final exam. In this week, you'll be getting started with deep learning in a historical overview and an introduction.

00:07:30 In Week 2, we're looking into shallow neural networks – those composed of very few layers – that can already do some pretty amazing things.

00:07:39 In the third week, we're diving deeper into networks and sequence models. And in the fourth week, we jump right into computer vision with convolutional networks

00:07:47 for visioning and for text. Week 5 is dedicated to industry applications of deep learning,

00:07:53 where we show examples of how deep learning can transform the way business applications work and businesses operate in the real world.

00:08:01 And last but not least, we reserved Week 6 for advanced topics. There will be a weekly exercise as well as a final exam,

00:08:11 and I wish you the best of success for these as we're jumping off into the real content and the real heart of this first week,

00:08:20 where Damian Borth from the German Artificial Intelligence Research Center (DFKI) will be talking about the historical origins of artificial neural networks on computers

00:08:31 and the steps of evolution these have taken towards today's deep learning that's set to transform entire industries.

00:08:39 So over to you, Damian.

Week 01 Unit 02

00:00:08 Hello, my name is Damian Borth. I'm the Director of the Deep Learning Competence Center of the German Research Center for Artificial Intelligence.

00:00:15 In German: Deutsches Forschungszentrum für Künstliche Intelligenz. And today, we're going to speak about Unit 2: "From Neural Networks to Deep Learning".

00:00:24 Before we start, I would like to introduce the DFKI for you. And the DFKI is an institute founded in 1988, with a long history of doing research in artificial intelligence.

00:00:37 And up to now, we're the largest not-for-profit AI center in the world. 800 people, 400 studying, working on different topics dedicated to artificial intelligence.

00:00:48 The Deep Learning Competence Center itself was founded in December 2015 in Kaiserslautern, and can be understood as a horizontal entity connecting different DFKI sites,

00:01:00 like in Saarbrücken, Kaiserslautern, Bremen, and Berlin to foster collaboration within the DFKI.

00:01:08 We also have external collaborators, as seen on this slide. You see Berkeley working with Stella Yu, and the International Computer Science Institute,

00:01:17 Shih-Fu Chang from Columbia University, Benjamin Elizalde from Carnegie Mellon, and many others. And in particular, I'd like to put an emphasis on our collaboration with Nvidia.

00:01:27 So we were, since September 2016, one of the first two AI labs being awarded by Nvidia, which gives us great access to the compute infrastructure needed for deep learning,

00:01:43 and early access to the technology. So I'm going to speak about deep learning and the transition from neural networks to deep learning.

00:01:51 One of the important questions is: What are we actually doing? So what I'd like to show you here are three examples where you see what humans are actually doing.

00:01:59 So we see on the top row a picture and the human being able to recognize a flower inside of the picture. We see in the middle example a driving situation.

00:02:10 So a person is looking from the inside of a car at the street, and the human is then able to recognize: "Okay, I have to drive carefully due to the current traffic situation

00:02:19 and make a decision upon this information." And at the bottom, we see the game Go

00:02:24 And here, a human can observe the game and make a decision on where to move the next piece within the game,

00:02:30 with the goal of winning the match. So this is a mapping from an input to an output,

00:02:37 and humans are able to accomplish these tasks without really knowing inside of the brain what is happening. And a similar idea is what we're doing with neural networks.

00:02:46 We're aiming to accomplish the tasks that are illustrated on the slide with a mapping between an input and an output.

00:02:54 The input being a picture like the flower, and the output being the class label: This is a flower that we're recognizing here.

00:03:02 So this is basically what we're doing here. And if you think about deep learning,

00:03:06 we can say deep learning is an umbrella term for machine learning approaches all based on deep neural networks.

00:03:13 It is an end-to-end learning approach, where features are learned together with the classification function.

00:03:20 And you can also say deep learning defines a cascade of multiple layers of nonlinear processing units for feature extraction and transformation.

00:03:28 So nonlinearity is very important in this context. And each successive layer uses the output of the previous layer as input.

00:03:36 There's a processing from one layer to another that we have in deep learning. So let's go to neural networks.

00:03:43 So when we have neural networks, we process the signal from the input – here defined as the input layer – through a set of layers called "hidden layers" to an output that's the result of this processing.

00:03:55 And the units of computation are marked here in this presentation slide as circles – they're called neurons. And the signal is processed throughout the networks, the layers,

00:04:04 and at the end you will have a result. So this is the classical setup that we have in neural networks.

00:04:09 When you think about deep learning, we have multiple types of neural network architectures,

00:04:16 and I always like to explain these types of networks with respect to the signal they process. So we see at the top left the neural networks – that classical setup.

00:04:25 We see at the bottom left the convolutional networks – very important for processing and analyzing images. So here, images could be understood as a two- dimensional signal.

00:04:35 And the convolution is a two-dimensional operation. So very naturally, there is a fit between the analysis of images

00:04:42 and processing by convolutional neural networks. If you have a temporal signal, for example speech,

00:04:48 then you need to encode these temporal recurrences and correlations inside of a network structure.

00:04:55 So you need a network to encode recurrences. And here, LSTMs, long-short term memory networks, have been applied very successfully.

00:05:04 And if you're thinking about video and video processing, that's a two-dimensional signal over time.

00:05:10 So here you can combine a convolutional neural network with an LSTM to do analysis of this type of signal.

00:05:17 And the last example is here at the bottom right. A very interesting type of architecture called generative adversarial networks.

00:05:25 So these networks, GANs for short, are actually not only analyzing content, they are generating content synthetically that has all the characteristics of natural content,

00:05:36 therefore a very interesting area. The training is done in an adversarial setup,

00:05:40 so here the latest trend in deep learning. We will now go to convolutional neural networks,

00:05:46 and have a little bit more of a look inside at what is happening in this type of architecture. So when you think about convolutional neural networks,

00:05:53 and the "Image Understanding" task, the processing of images, these networks – and this is the surprise – are not a new invention.

00:06:00 They have been there for a while. One of the very first convolutional neural networks, LeNet, was published in 1989

00:06:08 for the task of "Handwriting Recognition". So people were using these networks to understand digits and characters.

00:06:15 Here the very famous data set EMNIST was used, and was very successful in this domain,

00:06:22 but it was very difficult to apply this type of network to other domains. And it took a while until AlexNet came out in 2012,

00:06:30 where the whole convolutional neural network took off, and really proved to work on the general-purpose domain of image recognition.

00:06:38 So what happened here in 2012? In 2012, we had the submission of AlexNet to the ImageNet Challenge.

00:06:44 That's a competition where international research groups submit their approaches. And these approaches are evaluated on the same test data.

00:06:54 So the results are very comparable. And in 2012, a very important date for deep learning, AlexNet was submitted,

00:07:02 and took first place, beating all the other networks, with a gap in-between the first place and the second approach that was listed in this.

00:07:12 So what was the reason that AlexNet was so successful? And this is usually the first question people ask,

00:07:17 wondering what happened in-between LeNet in 1989, for example, and AlexNet in 2012. Well, three things came into being.

00:07:26 First of all, with deep neural networks we have an architecture that is flexible, so we can add layers of neurons,

00:07:33 and increase, so to speak, the learning capacity of those networks. With an increased number of layers,

00:07:40 we also have an increased number of parameters we have to fit. So this parameter fitting is what we call "training".

00:07:46 And the more parameters I have, the more data I need to train to fit the network. In 2012, with ImageNet, we had a huge amount of training data,

00:07:56 and this training data had high-quality annotations and labels. Therefore we had the flexibility of the network, we had the training data.

00:08:03 The only thing that was missing was the machinery to compute those neural networks, to train those neural networks very efficiently.

00:08:11 So here people used graphical processing units (GPUs) to train those networks. And for AlexNet, for example, it took approximately two months to train the very first network

00:08:22 to beat this competition and be placed in first place here. So those three things came together – flexibility of the network, huge amount of training data,

00:08:32 and the machinery, the computational model on GPUs, where you can train this very efficiently. So just to show you a little bit of the numbers on what this really means...

00:08:42 Here we have the ImageNet results for the image classification task. Top-5 error rates, which means the lower the bar, the better.

00:08:50 We see here the traditional non-deep learning approaches from 2010 and 2011. We had a 28.2% error percentage and a 25.8% error percentage.

00:09:01 And then, in 2012, AlexNet came and really beat all the other networks with a huge drop in error, which means an increase in performance.

00:09:10 And then really the first network was eight layers deep, inspired by the human visual cortex that can also be interpreted as having eight layers.

00:09:19 And then people very quickly adapted to this new approach of deep learning, and applied deep learning throughout the years,

00:09:26 introduced new types and typologies of architectures, like, for example, VGGNet and GoogLeNet, the Inception architecture.

00:09:34 And recently, also ResNet for Microsoft with residual layers that really were able to overcome the vanishing gradient problem

00:09:42 with networks today that have hundreds or even thousands of layers. And we still see an improvement in performance,

00:09:49 so the error rates are getting lower and lower with deeper networks. And this is what we call the "revolution of depth".

00:09:57 So what makes the difference? To really show you what, from the conceptual point of view, is making the difference,

00:10:04 I would like to show you the following task. Here we see a traditional approach in "Image Understanding".

00:10:10 We see an image – that's the building of the DFKI in Kaiserslautern. And the processing pipeline – how this image was analyzed to understand that there is a building scene.

00:10:19 So the first thing that people did is they processed and extracted features from feature descriptors, here SIFT (scale invariant feature transform) and HOG (histogram of oriented gradients).

00:10:30 And those features were hand crafted, they were fixed, and therefore fixed in their nature.

00:10:36 Those features were extracted, and then there was usually an unsupervised way where those features were aggregated in a bag of visual words representation.

00:10:44 And then this representation was taken to a classifier, and the classifier was using this to classify whether this is a building or not.

00:10:52 Similarly, in the context of speech recognition you have an audio signal. This audio signal was being processed.

00:10:59 Features have been extracted, MFCCs here in this particular context. Then we had an unsupervised step

00:11:06 where the data was aggregated into a "Mixture of Gaussians" representation, and then this was taken for the classification step to classify what kind of word was spoken.

00:11:15 So those are traditional methods. If you compare those methods now to deep learning,

00:11:19 we can see that, first of all, the feature representation that is here was all done in ones. So we have this deep neural network being represented in a modular way,

00:11:33 as a cascade of non-linear transformation from the raw data, the pixels, to the classification that this is a "building".

00:11:41 So here we can see that those modules are all built next to one another. And then if we look inside of the network, we can see that it's a collective training of features and classification,

00:11:51 which creates a shared representation, in particular at the beginning of the network. This representation is internal.

00:11:57 And what is very important is "adaptive". So we can take these networks and then transfer them towards other concept classes

00:12:06 and other objects that we want to detect inside of those images. So that's the idea and comparison of deep learning methods, or "end-to-end learning methods",

00:12:16 as compared to the traditional approaches where we had the feature representation being separately treated as the classification

00:12:24 So when we go into the reference architecture now with AlexNet, we can say that what you see here:

00:12:30 AlexNet is an eight-layer architecture. So what we see here at the very beginning is three yellow layers that represent the image.

00:12:39 Three layers because the images are represented as RGB. And then you have five layers of convolutions.

00:12:45 And these five layers are actually aiming to learn the feature representation automatically. And then we have three layers at the later part of the networks that are called fully connected layers.

00:12:58 And these layers are actually responsible for the classification of the task. And both together, they define the neural network, the AlexNet architecture,

00:13:07 that is eight layers long or deep. And if we look a little bit more into detail,

00:13:13 we can see that in particular in the convolutional layers we have convolutional operations, we have pooling operations, and we have what were called "local response normalization" operations in the original architecture.

00:13:24 So those layers aren't in the current architectures anymore, but in the original AlexNet architecture, they were part of it.

00:13:31 So let's have a look at these particular elements of the network. So the convolutions are actually operating,

00:13:37 and these operations are calculating the convolution over a local image region with the feature map that this learned,

00:13:46 and the weights of the network, if you like. And here we have particular parameter constraints that we can apply.

00:13:52 One of them is that we define a receptive field to use the local connectivity between networks. And therefore, we also have a shared representation.

00:14:02 So this is what can be seen in the image, at the bottom right. So those are the convolutional layers.

00:14:09 Then what we have in these convolutional layers is 96 of the learned features that we can see here from AlexNet, and the visualization of those.

00:14:19 So here we can see we have a receptive field of 11x11 pixels, and three dimensions in the depth that goes

00:14:29 And here, the learned weights, as you see, they're more or less defining the edge detection at different orientation and block detections.

00:14:39 This is what you see here. If we then go further, we have the pooling layer component.

00:14:44 Here in the pooling layer, this is an operation which fuses multiple inputs into an aggregated output with respect to the spatial dimension.

00:14:54 So it increases robustness. And what very often is used is "max pooling",

00:14:59 which is illustrated here in the image at the bottom right. So take a look at the red area.

00:15:04 You have a 2x2 filter with a stride 2. So we have the values 1, 1, 5, and 6.

00:15:09 And after the pooling operation, the value 6 has survived and is then taken in a spatially reduced representation as output and input for the next layer.

00:15:20 At the end of those operations, for each layer we have an activation function. Usually in a neural network a sigmoid function is used.

00:15:27 However, in this AlexNet architecture, we have a rectified linear unit, a "ReLU", which is represented here at the bottom right with the formula.

00:15:36 And this function is firing when the aggregation of the previous operation reaches a particular threshold, here "0", as you can see.

00:15:45 And very importantly – this component is responsible for the non-linear behavior that we have in convolutional neural networks.

00:15:53 So now we've had a very brief overview of AlexNet architecture, eight-layered. If you look now at this architectural representation, this is one way of looking at the network.

00:16:04 There are multiple ways of looking at a network. We already saw one representation from the original paper from Alex Krizhevsky, at the bottom right.

00:16:12 But there are other representations, such as the "Caffee Visualization". We have the "Blob Caffee Visualization" that is displayed here.

00:16:19 We have the "Stacked Layer Visualization", or here the "Volume Layer Visualization".

00:16:23 And the surprise is they all look very different but they actually all picked the same network.

00:16:30 So this is all the same network architecture, this is all AlexNet.

00:16:34 And the challenge, actually, when you start to do deep learning and to read those architectures, the challenge is really to understand which parts in these visualizations, in these illustrations of the network architecture are new,

00:16:47 and which parts are actually known but just look different. So it's not easy, but it's very important to get into the details, to understand the parameters.

00:16:57 And with these slides, I would like to close this unit. Thank you for your attention, and we'll move on to the next one.

Week 01 Unit 03

00:00:07 Hi, welcome to Unit 3 of Enterprise Deep Learning with TensorFlow. My name is Karthik, and I am a Machine Learning Researcher in the SAP ICN Machine Learning Team.

00:00:19 In the previous unit, we saw a brief history of machine learning and neural networks, and discussed some common machine learning terminology that will be used in this course.

00:00:30 We also saw the relationship between different terms. In this unit, we will have a look at the software used in this course,

00:00:38 clone a GitHub repository, and we'll also look at Jupyter notebooks, which is the choice of environment that we will be using to develop machine learning frameworks.

00:00:51 These are libraries that we will be using in this course. It is recommended that participants install these libraries to experiment

00:00:58 and understand the topics discussed in this course. Installation instructions for these libraries are available on the course description page.

00:01:11 Let's look at Anaconda. Anaconda is a Python package manager that allows installation of Python and its relevant scientific packages.

00:01:21 It is built to ensure that packages and their dependencies are installed easily. It allows creation of sandboxed isolated environments

00:01:29 that are capable of maintaining separate versions of different libraries without affecting other Python projects.

00:01:36 Miniconda is a lighter version of Conda – with just Python and its related dependencies installed on demand.

00:01:43 So now let us look at Anaconda and what we can do with Anaconda. Let's start our terminal.

00:01:53 And here well see just a quick run for the version of the Anaconda that we're running. So "$conda --version" tells us a quick version of Anaconda.

00:02:02 Let's do a list. We'll see that we have quite a lot of packages installed.

00:02:07 So when we do a filtering of the list, we will see probably… Let's search for TensorFlow.

00:02:20 And we see TensorFlow as being currently available, it's currently available right here.

00:02:26 Let's check for in NumPy. So NumPy is currently version 1.11.1.

00:02:32 We have our different versions for different environments. Moving back, we'll look at Jupyter.

00:02:41 Jupyter is a Web application that is used for live programming and visualizations. It provides a neat separation of the language- agnostic layers,

00:02:49 which is the code editor from the kernels, which is the programming language. Jupyter allows writing code in Julia, Python, and R – hence the name Jupyter –

00:02:59 with support for many other kernels. Now let's look at how we can use Jupyter

00:03:04 and what we can probably do with the Jupyter notebook. So let's quickly open a terminal,

00:03:11 and in the Notebooks page, we will have a few notebooks that are available. Let's quickly run them.

00:03:17 And we should be able to see that we have… Let's go to localhost.

00:03:25 And we will see all the notebooks listed on this page. We will go to Week_01_Unit_03, which is going to be starting Jupyter notebooks.

00:03:37 So we have cells in the Jupyter notebook which are called blocks, which are going to run individual code commands.

00:03:44 So a cell is either a code or markdown cell. So you see, just like a typical code editor, we have menus on top,

00:03:53 and we have toolbars which are going to let us do different things, Different operations on our cells.

00:04:00 So, when we highlight a cell, and if we, say, do a B, we're going to add a cell below it. If we do an A, we're going to add a cell above it.

00:04:08 And an X will basically remove those cells. All these examples are provided in this notebook.

00:04:15 So we have some most commonly used shortcuts. And we also show some examples of how we run, say, Python over here.

00:04:24 Let's do a quick example. In this example, we are doing a simple summation and string formatting.

00:04:31 And we see that the output of the summation, the string itself, is available right below the cell.

00:04:36 This is the one good thing with Jupyter notebooks: We can effectively run the cell and have the output printed right below it.

00:04:44 So let's do a quick list comprehension and do a quick print of all the numbers that are available in this list.

00:04:51 We see that we can actually do very good formatting of the string and just print the output right below it.

00:05:00 We can also display the images using Jupyter and, over here in the cell, we can change it as a markdown.

00:05:07 And if you see the markdown, the code behind it's going to look very similar to what we do with HTML.

00:05:15 One more good thing with Jupyter is that we can run and plot and visualizations – the data itself. Here, we see a scatterplot and the line plot plotted right along the same plotting with matplotlib.

00:05:30 matplotlib is one of the libraries that we'll be using throughout this course for plotting and visualization of the results and the data.

00:05:41 So there's a great resource on datacamp that can be looked at for different variations of matplotlib and plotting itself. One more very interesting thing... since we're doing the machine learning course,

00:05:52 we thought it would be good to explore large data sets, and that's when we talk about Facets.

00:05:58 Facets is an open-source machine learning data sets visualization tool, which can be used to run different data, load different large data.

00:06:10 And in this example, let's look at the UCI data set, and we will be basically be loading a lot of data points here,

00:06:18 and some of the features that we will be using are basically Age, Workclass, Education, and all the other fields that are available in the UCI data set.

00:06:28 So quickly running… We're going to have basically a cell that's going to run a very beautiful visualization –

00:06:37 that's the Facets visualization – all right on the notebook. So if you see, we can actually have the charts displayed right by the data.

00:06:47 We have a data description available on the left and the charts themselves on the right. So we can actually see a "log" version of the data set.

00:07:00 We can actually expand the charts and see a better visualization. And we can also do the percentages – we can see what the percentage distribution of the data is here.

00:07:10 So all the data points that are shown in red have missing values, and that's why it's highlighted in red.

00:07:17 In subsequent weeks, we will actually have some other units which are going to use facets to actually explore the data

00:07:26 and build a machine learning estimator with TensorFlow. Moving on…

00:07:37 We'll talk about GitHub. GitHub is a version control platform that allows hosting and managing projects.

00:07:43 Git is the version control system that is used by the GitHub platform to perform all the operations.

00:07:49 Git allows for collaboration between teams and team members. It reduces code collision and also provides ability to make most operations to the repository right on the local system.

00:08:00 Some local operations include: Diff, Commit, file history, merging branches, revision of a file, and switching branches.

00:08:08 These are all different operations that can be done locally, even without connecting to the Internet.

00:08:13 Once you have an Internet connection, you can push all these to the repository. So let's take a look at a quick example of how we can use GitHub and clone a repository.

00:08:26 In this page, let's go to, say, SAP's GitHub repository, and say we would like to download the open UI5 sample app.

00:08:36 For example, this is a repository that you would go to for you to download the app. But you will see that there is a clone or a download,

00:08:44 so if you didn't want to have Git and all its features, you would probably just use a downward.

00:08:50 And you will get a download ZIP, which is going to be a file that is compressed, and you can unzip it just like any other file from the Internet.

00:08:56 But one more thing that you can do is copy the and you can do a GIT clone and paste the URL.

00:09:04 Effectively what this does is it goes to the repository, downloads it, and maintains a copy locally.

00:09:11 What this allows you to do is, when there are new changes pushed to the repository, you will also get all these changes reflecting locally when you do a Git pull.

00:09:26 So, coming up next, we'll talk about neural networks. We'll build our own our own neural network with just NumPy,

00:09:35 and we will explore the math behind this neural network and how it learns. And over and above that, we'll also learn some key terms used in machine learning.

00:09:44 And that's it. Thank you for watching, and we will meet in the next unit.

Week 01 Unit 04

00:00:06 Hi, welcome back to Unit 4 of Enterprise Deep Learning with TensorFlow. In the previous unit, we discussed the software and libraries that will be used in this course,

00:00:18 learned about code versioning using GitHub, and explored Jupyter and its capabilities.

00:00:25 In this unit, we will be building a neural network with plain NumPy. What is a neural network?

00:00:34 A neural network is a program that learns from input data. The idea behind neural networks was inspired by the behavior of the human brain and how it responds to stimuli.

00:00:45 So, why are neural networks so widely used? The reason for widespread usage of neural networks stems from the facts that:

00:00:52 One, no explicit rules or hand-crafted features need to be fed to it. Two, neural networks are exceptionally good at recognizing patterns present in data.

00:01:03 On the right-hand side of the slide, you will see a typical neural network design. Let's jump into a demonstration on what a neural network can do.

00:01:14 For this demonstration, we will talk about TensorFlow playground. TensorFlow playground is a lightweight neural network that runs on your browser.

00:01:23 We can teach about different hyperparameters and their impact on the output. The tasks that you can experiment with on TensorFlow playground:

00:01:32 It allows you to design different neural networks for classification or regression. Let's solve a simple classification task on TensorFlow playground.

00:01:44 For this task, let's go to playground.tensorflow.org. On the left-hand side of the browser, you will see different data sets and their descriptions.

00:01:52 Let's choose a simple data set for now. If you see the data set, the visualization for the data set is available on the right-hand side.

00:01:59 The blue points are from one class while the orange are from another. Going back to the left, we have different descriptions,

00:02:08 like ratio of training data, which is going to say how much test data you're going to use, and how much of the training data you're going to use.

00:02:15 The noise and the batch size. We'll talk about this in detail in the units coming up later.

00:02:21 But let's start a quick experiment to see whether, for a thousand epochs, this network learns anything at all.

00:02:29 So we're running about 500 epochs and let's stop it right here – 250 epochs here. If you see, the network has almost learned pretty well.

00:02:38 You can actually differentiate the blue points from the orange points. And if you see the test loss and the training loss, they're at zero.

00:02:47 This is an indication that the network has actually learned to predict the points very well between the test and the training. So it uses the training data to predict on the test data.

00:02:57 And the test loss shows that the network has learned very well – that it can actually generalize from the training data to the test data.

00:03:09 Moving on… So, what is a neural network and how does it learn?

00:03:15 In a neural network, the weights are updated with the objective of reducing error. How is the error calculated?

00:03:23 The output labels are an indication of how much of an error it is actually making. And what happens with this error?

00:03:30 Well, error is propagated back to the network to alter the weights of the neuron – and this is in effect what is called backpropagation.

00:03:39 For this task that we are going to do in this unit, we will basically implement a simple backpropagation using NumPy for a simple problem,

00:03:47 and we will see what the problem is in our notebook. So we will load the Jupyter notebook, just like last time.

00:03:55 And we will move to Unit 4 of Week 1. If you go to Unit 4, the notebook here, you will have some description of the data.

00:04:05 The problem that we're trying to solve is trying to predict a zero if the second input, say, probably is a zero, is a one, or one if the second input is a one.

00:04:17 So what we're trying to do is not a simple A x B sort of problem. We're trying to predict whether the output is going to be a zero or a one with just one input.

00:04:28 The others do not matter. The network here is actually quite nonlinear, so the problem here is nonlinear.

00:04:35 Let's quickly run through this notebook and see how, whether the network is actually learning at We'll import some of the simple NumPy exponential array and random of our libraries.

00:04:47 And this is a neural network class which is going to be a single neural network – a single layer network. And we have some methods here which are things like sigmoid, sigmoid derivative, and the train method itself.

00:05:01 The train method here runs for 10,000 iterations, which means that, at every step, the network is trying to predict an output,

00:05:08 and the error made in predicting that output is fed back to the neural network. So the forward pass, which is the prediction, is called feedforward,

00:05:17 and the backward pass, in which the weights are altered, is called backpropagation. Let's quickly run through this class.

00:05:28 The sigmoid here is going to change the network, so the outputs between zero and one, so this is a probability score,

00:05:35 while the sigmoid derivative is going to tell how confident the existing learned weights are. And the training itself is going to take just a few milliseconds here.

00:05:46 So let's quickly run this network, and we will see we're actually splitting this data into train and test,

00:05:52 since we have three inputs, which is are going to be zero or one, we have a total of nine combinations.

00:05:58 And effectively, we can actually see that the output here – the training data – is going to be six points out of the nine, where we have a split of ones and zeros,

00:06:09 which are equal – so an equal number of zeros and an equal number of ones. And the test points are basically points that the network was not trained on.

00:06:19 And let's quickly run through this example. We're initializing the weights of the network here,

00:06:26 and we can see that initially we have random weights, which are -0.25, 0.9, and 0.46. And let's quickly train this network.

00:06:34 This should be done in less than a second probably. And if you see, the network is already trained – 10,000 iterations –

00:06:40 and it's actually saying -4, 12, and -4. So now that we have the trained data, which is going to be on this training data,

00:06:49 we are now going to test it on the test data, which is going to be 1 0 0 and 0 1 1. The expected output of the test data is zero and one.

00:06:57 So let's quickly run the first row for test and see what happens. So if you see, the prediction from the network is close to 0.01, which is close to zero.

00:07:06 So in effect, what we're trying to do is we're trying to actually say a threshold. If you are to say less than 0.5, you're going say zero,

00:07:13 and if it's more than 0.5, you will say one. But in this case, it's 0.1, which is expected as close to be a zero.

00:07:22 So the next input point, test point, here, is
0 1 1. And as we expect, it's actually doing a 0.99 score of confidence that's actually close to one,

00:07:32 which is saying that the network has learned pretty well, and that it can actually generalize just based on six data points on the test data,

00:07:40 which is three data points – we have tested two of them. So as a task for you, you could probably do other examples:

00:07:48 You could implement an XNOR, or you could implement an AND, or you could do an OR. All you have to do is feed the inputs and the outputs,

00:07:59 so you don't have to worry about how this is going to be implemented. The network itself will understand the weights.

00:08:05 It will try to calculate the weights, and it will do the prediction for you. And this is how a neural network learns, so it's trying to predict,

00:08:13 and with the error it makes in prediction, it's actually modifying the weights of the network, and it's trying to predict again, with the object being to reduce the error in that prediction.

00:08:24 And, in effect, with simple data like this, we are able to actually produce almost with certainty that, even with the test data not being there,

00:08:33 and the training, we can actually predict the outputs here, which are zero and one.

00:08:38 So what did we do? We implemented a simple NumPy neural network with NumPy.

00:08:47 Thanks for joining me in this unit. In the next unit, we will have an introduction to TensorFlow.

00:08:52 We will also understand some TensorFlow, how it represents data, exports some concepts, and just compute graphs,

00:08:59 and solve a simple machine learning problem with TensorFlow. See you.

Week 01 Unit 05

00:00:07 Hi, welcome to Unit 5 of Enterprise Deep Learning with TensorFlow. In the previous unit, we used NumPy to build our own neural network from scratch.

00:00:17 We used a simple problem and solved it using a single-layer neural network. We also experimented with TensorFlow playground

00:00:25 and got a hint of some terms that will be used throughout this course. In this unit, we will introduce TensorFlow, a versatile machine learning library,

00:00:33 and understand key concepts such as tensors, computation graphs and sessions. So let's get right into it.

00:00:41 What is TensorFlow? TensorFlow is an open source library for numerical computation

00:00:47 that was released by Google with an Apache 2.0 license. TensorFlow is currently one of the most popular libraries to develop machine learning frameworks.

00:00:56 Why TensorFlow? TensorFlow is a highly flexible library for developing machine learning frameworks.

00:01:01 It's an end-to-end library that can be used for developing machine learning models to deploy the trained models for inference.

00:01:09 Frameworks developed using TensorFlow can be very easily moved from prototyping stages to production. So, how does TensorFlow go from development to production without much reengineering?

00:01:24 This is an illustration of the TensorFlow architecture. TensorFlow provides a device-agnostic execution framework

00:01:31 that is executed by the TensorFlow distributed execution engine, and what you deal with are the various language front ends that TensorFlow supports.

00:01:40 While Python is the most complete and easy-to- use API, TensorFlow supports other language front ends such as C++, Java, and Go.

00:01:50 TensorFlow layers provides utilities to help build models while estimators allow you to experiment with different architectures. TensorFlow Estimators provides a standard interface to perform tasks that you need with a model,

00:02:04 while TensorFlow Keras is a popular high-level API specification that is built on top of the core TensorFlow features, allowing easy prototyping of machine learning frameworks.

00:02:15 On top of the architecture lies Canned Estimators, efficient implementations of complete standard models that are ready to go out of the box.

00:02:28 What is a tensor? In simple terms, a tensor is an n-dimensional matrix.

00:02:34 Hence, an m x m matrix is an array of numbers in the shape of a square that is m numbers tall and m numbers wide, while an m x m x m tensor is an array in the shape of a cube that is m numbers tall,

00:02:48 m numbers wide, and m numbers deep. What are TensorFlow operations?

00:02:53 TensorFlow operations, also called ops, are nodes that perform computations on or with the tensor objects. What is a computation graph?

00:03:04 A computation graph, is a type of directed graph that is used for defining the computational structure. In the example on the right, we see how the final value c is calculated using the inputs a and b.

00:03:15 The "add" node in green is directly dependent on the earlier "add" and "multiply" nodes in blue. The "add" node in green is also dependent on the inputs nodes a and b,

00:03:25 but through nodes "add" and "multiply" in blue. Hence an indirect dependency exists on nodes a and b for the "add" node in green.

00:03:38 Let's move on to TensorFlow Sessions. What exactly are TensorFlow Sessions?

00:03:43 TensorFlow Sessions are responsible for graph execution. A session object can be used to calculate the value of a tensor object.

00:03:51 What are fetches? Fetches allow you to evaluate either a tensor or an operation object.

00:03:57 Fetches using a tensor object return a NumPy array while a fetch on an operation runs the operation and returns a None.

00:04:07 What is a feed dictionary? A feed dictionary allows you to override tensor objects in the graph.

00:04:14 This is especially useful when inputs are not known beforehand. An example of such an instance would be feeding sets of data points to a graph

00:04:22 when these input points could be read from a file, probably in small batches, or retrieved from a database. Let us put all this information about sessions and graphs together and write our first TensorFlow program.

00:04:38 We will open a Jupyter notebook, like we did in the earlier sessions. And we will go through Week 1, Unit 5, and we will open the notebook here.

00:04:48 This notebook is an introduction to TensorFlow. This talks about various operations that you can do with TensorFlow.

00:04:57 So let's run the first cell. The first cell is an import for TensorFlow itself.

00:05:02 And we see that we are running TensorFlow 1.3, and this can be done with a "tf.__version" command.

00:05:11 And we also use the TensorFlow interactive session. The interactive session allows us to run session objects in interactive environments,

00:05:20 just like in a Jupiter notebook. So the next thing we do is initialize some constants:

00:05:27 a and b, in this case. And we add those constants. With an interactive session, what we can do is evaluate the value of c right here,

00:05:33 without running the session with "session.run", like we do with a regular session object.

00:05:40 Similarly, we can do the same thing for subtracting a and b. And we can see that c can directly evaluate this value and it can print 12 here.

00:05:50 In the next cell that you can actually see down here, we have different math operations that can be done. For instance, we can multiply, we can divide, we can also do other operations like compare,

00:06:01 take the power of a and b, and we can also do conditional statements. So a lot more are available.

00:06:07 You can actually go to this URL, and you can see more control flow operations as well.

00:06:14 Moving on… Let's try to create matrices.

00:06:20 And we will see how to do some TensorFlow session object… let's run a session object here.

00:06:26 In this case, we create two matrices, a and b, which are a 3 x 1 matrix and a 1 x 3 matrix.

00:06:33 And we will use this to do a basic matrix multiplication. We use the "tf.matmul" operation, and this will take in the parameters,

00:06:43 the matrix a and matrix b. We will print this, and let's see what happens.

00:06:51 So here you see that TensorFlow is giving an object, which is a tensor object. It's saying that the "prod_op" is just a tensor object.

00:06:55 It is not evaluating the actual value – unlike in the earlier case, when we said "c.eval", since we were in an interactive session, it evaluated the value of c and gave us the result.

00:07:06 In this case, it just says that "prod_op" is just a tensor object. Let's create a session now.

00:07:13 This is how you create a session. We use a "tf.session".

00:07:18 And then we use this "session.run" to run the value, to get the value, evaluate the value.

00:07:24 And if we see that, we get the value of 105, which is the matrix multiplication of these two matrices.

00:07:31 And once we're done, we do "session.close", just like in any Python use case.

00:07:35 So we will basically open an object and we will close it. Suppose you don't want to do an open and a close, or you forget to close the object,

00:07:44 you can use the "with" block. The "with" block allows you to open the session as another object as "sess",

00:07:51 and what we do right over here, just like we did in the earlier cells, is evaluate this, and then we get the same value,

00:07:58 except that here the object is opened and closed by Python because of the "with" block. Finally, we'll also see how to run this on different computes.

00:08:09 Suppose I have a CPU, which has eight cores, like I do in my Mac right now, I can basically first of all see if I can run an "sysctl" command on my Mac

00:08:19 to see how many cores I have. And I can run the CPU operation here.

00:08:24 In this case, I'll just use device zero, which is the zero processor ID, and then I will simply run the same command.

00:08:30 But if you have other devices, other cores, for instance, then you can do the same thing, and then you just have to feed in the device ID.

00:08:43 If you had GPUs, for instance, you need to provide the GPU ID here, and you need to replace a CPU with the GPU value here, that's all.

00:08:53 So in this notebook, we've seen how we can do basic tensor flow operations, and how we can use the session object to evaluate some values.

00:09:09 What's coming up next? We will learn about the situations where we should use deep learning

00:09:16 and, more importantly, when not to use deep learning. We will also understand in which situations other classical machine learning techniques

00:09:23 or rule-based approaches are preferable. See you.

Week 01 Unit 06

00:00:07 Hello and welcome to the last unit of Week 1, "When to Use Deep Learning". My name is Daniel Dahlmeier.

00:00:13 I'm a development manager at the SAP Leonardo Machine Learning team, and I'm excited to be back here at openSAP with this course

00:00:20 "Enterprise Deep Learning with TensorFlow". In the last video, we introduced TensorFlow,

00:00:26 and we learned the basic concepts of TensorFlow, like compute graphs, and we saw how TensorFlow makes developing neural networks easier and more scalable.

00:00:35 In this unit, we will learn in what situations we should really use deep learning and in which situations maybe other classical machine learning techniques

00:00:43 or even rule-based approaches might be preferable. So, why really "deep learning"? Why is deep learning so successful?

00:00:53 When training a machine learning model, you often experience that the performance of the model plateaus after a while

00:00:59 as you feed the model with more and more training data. It is as if a lot of classical algorithms cannot saturate,

00:01:06 and they are not really able to take advantage of the large amounts of training data that we have today. Our deep learning, on the other hand, seems to be much better able to take advantage of these large data sets,

00:01:19 and keep improving with the size of the data. And, in particular, if you increase the size of the network,

00:01:27 increase the number of the neurons and the number of layers in the network, you see that larger networks often scale much better with the amount of training data that we have today.

00:01:36 So, in other words, the model can really take better advantage of the large training data that we have available in today's big data era.

00:01:45 So, when training data was small, or training data is small, deep learning models didn't really perform much better than classical machine learning algorithms,

00:01:54 but often they were comparable in their performance. But when the training data is very large, deep learning often outperforms other algorithms.

00:02:04 And as we are living in this world where large training data sets are available, much larger than even five years ago,

00:02:11 that's where deep learning really has started to take over, and show state-of-the-art performance in a lot of disciplines,

00:02:19 like computer vision, in speech, or natural language processing. So a lot of people approach us with the question

00:02:31 "When should we use deep learning, and in which situations, in which use cases, is deep learning preferable?" In general, deep learning and machine learning is a very empirical discipline

00:02:42 and it is often difficult to find hard and fast rules on what type of machine learning algorithm to use in what situations.

00:02:49 In fact, data scientists usually try a lot of different models and techniques when building a model and when they face a new task or a new data

00:02:57 Nevertheless, we often get asked: "When should I use deep learning?" And these are a few general guidelines that you should consider

00:03:06 to see whether deep learning is a promising approach to your problem. First, we already mentioned that large amounts of data are often a promising situation for using deep learning

00:03:20 because deep learning really shines when it has large amounts of training data available. And deep learning also is a state of the art in a lot of disciplines,

00:03:29 like computer vision, speech recognition, and natural language processing. So if you are processing images, audio data, or natural language,

00:03:37 then you might want to consider deep learning as a very promising approach here. And more generally, if your data has a very raw signal, that has very little structure by itself,

00:03:50 and if you want your algorithm to learn more meaningful representation, so the algorithm should really discover good features in the data,

00:03:58 rather than you manually engineering the features because it might be very hard, then deep learning might be a technique that you want to consider as well.

00:04:07 Take, for example, images where you have raw pixels which, by themselves, have very little information. And it would be very hard to write a good feature extractor which says:

00:04:15 If this pixel has this value, then the image shows a cat. But deep learning might really be able to learn these features from the raw data.

00:04:27 But deep learning is not a silver bullet that automatically solves all machine learning problems. There are certainly situations when deep learning should not be the first thing to consider.

00:04:39 First of all, a lot of people forget that if you can solve a problem with simple hand-written rules, then that might actually be the cheapest and easiest way to solve your problem,

00:04:48 and maybe you don't even need machine learning at all. Machine learning practitioners and deep learning experts often frown at the idea of writing hand- written rules.

00:04:58 But in a practical situation, if you can solve the problem with rules – well, why not? But if you have a machine learning problem, but your data might be very small,

00:05:11 then deep learning is less likely to shine here. And it might do worse, or at least not much better, than traditional machine learning algorithms.

00:05:20 And finally, in a lot of problems that involve structured data, for example, tables in a database,

00:05:27 where the algorithm can directly leverage the structure and the features that you might be able to write by hand,

00:05:33 then classical machine learning algorithms, like decision trees or the perfector machines, might be just as good as deep learning,

00:05:41 and sometimes easier, and sometimes even better in performance. So if we now put these guidelines into a simple decision diagram,

00:05:51 we get a simple "cheat sheet" for when to use deep learning. So let's start at the top left,

00:05:59 and just consider for a moment whether you can use simple rules to solve your problem. And if you can, well that might actually be the first thing you should try.

00:06:09 If you're convinced that rules will not do it and you need machine learning... if you're working on images, audio, or language data,

00:06:17 then you should probably try deep learning, or at least consider it. If you have a large amount of data,

00:06:24 and you're convinced that you need to learn the features from the raw data, then deep learning is also the first choice.

00:06:31 But if you have a small amount of data, or you are convinced you can write feature extractors by hand, so you have good features in the structured data,

00:06:41 then it might not have to be deep learning, and you should at least consider a few baselines using classical machine learning algorithms,

00:06:48 like SVM or Random forest, that might just work fine.

00:06:52 But it might still be the case that deep learning is a good fit for your problem, so you can try both. But maybe you just start with a simpler machine learning baseline.

00:07:02 In any case, you should always aim to very quickly implement a first simple baseline and then iterate quickly by seeing what errors your model makes and how you can improve from there.

00:07:14 As we have said many times now, machine learning is very empirical. And even experts are not always right with their intuition on what model works best.

00:07:23 So instead of having long arguments over what algorithms you think you should use and what you think might work best,

00:07:30 I really advise you to quickly build a baseline, and then iterate and see how you can improve the model step by step.

00:07:39 In this unit, we have seen why deep learning is becoming so successful, and we have discussed some criteria that you can use

00:07:45 to evaluate whether you should use deep learning in your project, or whether you maybe shouldn't use deep learning in your project.

00:07:52 This already concludes Week 1. And in the next week, we will move on to really building a TensorFlow application end to end.

00:07:59 We will learn the nuts and bolts of how to conduct a machine learning experiment, build a model using TensorFlow estimators, and serve the model using TensorFlow serving.

00:08:07 Now I just wish you good luck with the first weekly assignment, and see you in the next week.

www.sap.com