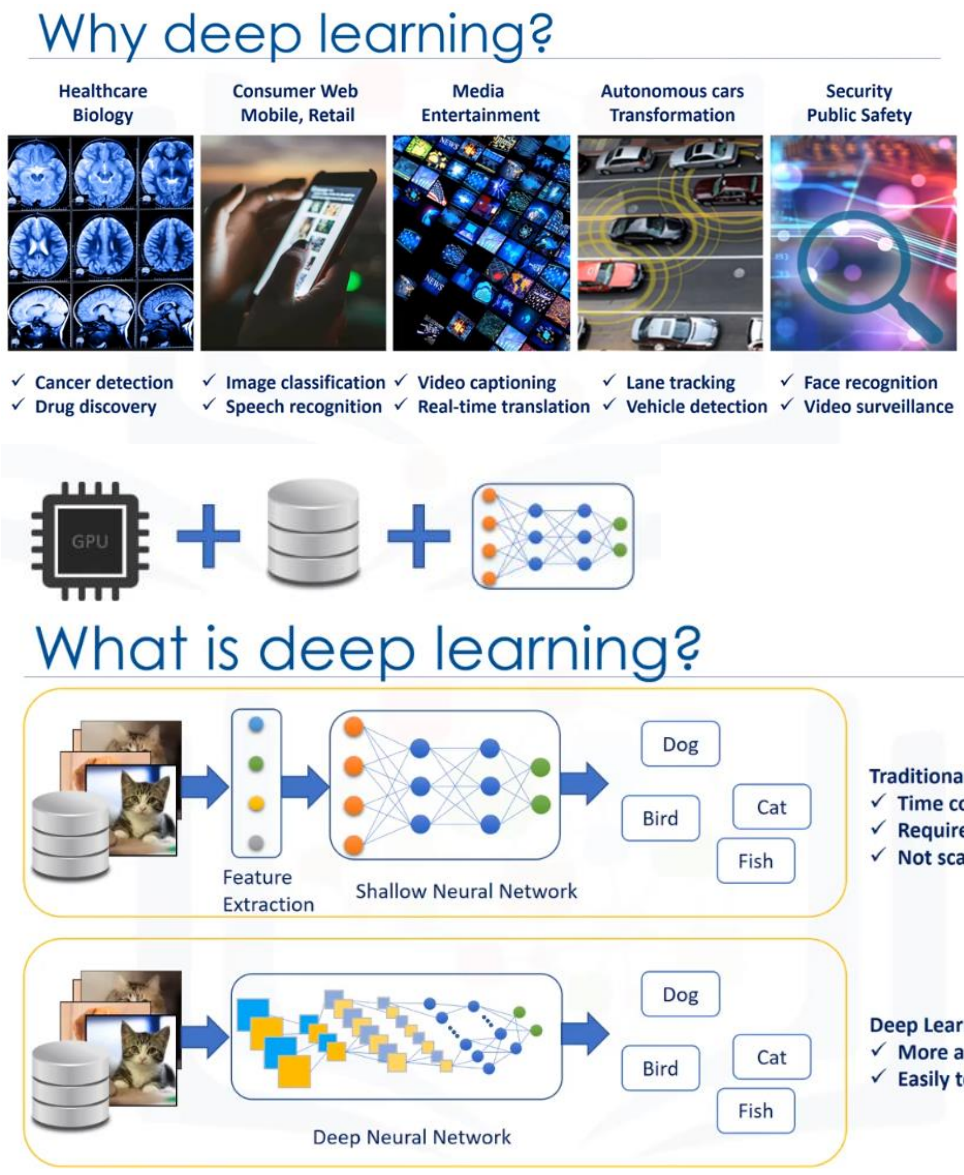
MODULE 1 – QUICK REVIEW ON DEEP LEARNING

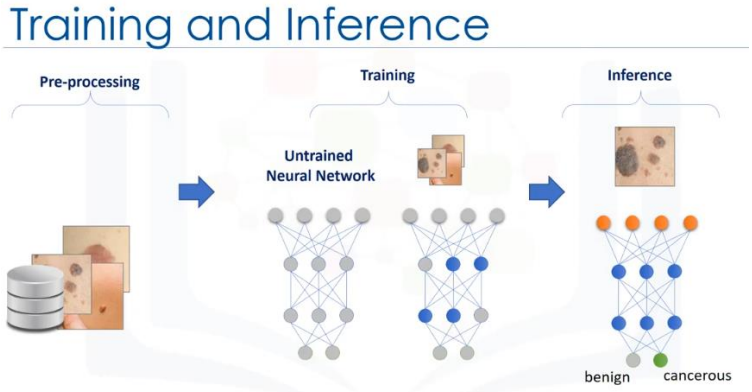
INTRO TO DEEP LEARNING



What is deep learning? And why is it such a hot topic today?
In this video, we'll begin to answer those questions. We'll also look at the difference between traditional shallow neural networks and deep learning.
I'm sure most of you have already heard about the usage of deep learning. It's not overstating it to say that the use of Deep Learning is becoming increasingly prevalent across all industries. For example, Deep Learning is trying to help
the health care industry for tasks such as cancer detection and drug discovery.
In the internet service and mobile phone industries, we can see various apps which are using deep
learning for image/video classification and speech recognition. For example, Google Voice, Apple Siri, Microsoft Skype, and so on. In media, entertainment, and news, we can
see applications such as video captioning, real-time translation and personalization,

or recommendation systems such as Netflix. In autonomous cars, Deep Learning is trying to overcome key concerns, such as sign and passenger detection or for lane tracking.
In Security, Deep Learning is used for face recognition and video surveillance.
Deep Learning is used in many other fields and domains as well.
Looking at all of these industries, we can see that the increasing popularity of Deep Learning today is due to three reasons: first, in the dramatic increases in computer processing capabilities; second, in the availability of massive amounts of data for training computer systems; and third, in the advances in machine learning algorithms and research. Now, let's take a closer look at deep learning and see why it's such a hot topic today? Assume that you have a dataset of images of animals, such as cats and dogs, and you want to build a model that can recognize and differentiate them. This model is supposed to look at the sample set of images, learn from the images, and get trained.
Later, given an image, we should be able to use this trained model to recognize the image as either a dog or a cat. Though this looks like a simple task, even for a very young child, computers often have difficulties in overcoming this task.
So, let's see how computers address these difficulties.
Traditionally, your first step in building such a model would be "feature extraction and feature selection." That is, to choose the best features from your images, and then to use those features in a classification algorithm, such as a shallow Neural Network. Doing this would result in a model that, given an image, could predict "cat" or "dog." Those chosen features could simply be the color, object edges, pixel location, or countless other features that could be extracted from the images. Of course, the better and more effective you are at finding feature sets, the more accurate and efficient you can become at image-classification.
In fact, in the last two decades, there has been a lot of scientific research in image processing related to finding the best feature sets within images for the purposes of classification.
However, as you can imagine, the process of selecting and using the best features is a tremendously time-consuming task and is often ineffective.
Furthermore, extending the features to other types of images becomes an even greater problem – because the features you've used to discriminate cats and dogs, cannot be easily generalized to things like recognizing hand-written digits, for example.
Therefore, the importance of effectively and accurately selecting features can't be overstated.
Enter "deep neural networks" – such as Convolutional Neural Networks.
Suddenly, without having to find or select features, this network automatically and effectively finds the best features for you. So instead of you choosing what image features to classify dogs versus cats, Convolutional Neural Networks can automatically find those features and classify the images for you. So, we can say Deep Learning is an algorithm that learns directly from samples much better than traditional approaches.
Thanks for watching this video.
End of transcript. Skip to the start.

DEEP LEARING PIPELINE



Start of transcript. Skip to the end.
In this video, we will look at real world problems that deep learning tries to solve,

as well as the deep learning pipeline. We'll also investigate why the deep learning pipeline is slow. To be honest, the problems that most data scientists are trying to solve when using deep learning revolve around things that are much more serious than recognizing cats or dogs from images.

Indeed, Deep Learning is being used to solve major issues in the health care industry, such as skin cancer, which is the most commonly diagnosed cancer in the United States. Over five million cases are diagnosed each year, costing the U.S. healthcare system over $8 billion. More than 100,000 of those cases involve melanoma, the deadliest form of skin cancer, which leads to over 9,000 deaths a year.

The key question becomes, "How can the use of computers solve issues that even leading physicians can't overcome?" While diagnosing melanoma early is the key to a patient's survival, there are a number of challenges in doing so.

First, there is a limited supply of doctors specializing in melanoma, making them costly to visit or difficult to access in many geographic regions.

Second, to the naked eye, there might be no obvious difference between a healthy mole and life-threatening melanoma. Even trained general practitioners, dermatologists, and surgeons struggle to interpret complex lesion morphology.

Third, it can sometimes takes weeks for lesions to be examined, biopsied and sent to pathology for diagnosis. So, what is the solution here? The solution is a computer system, smart enough and quick enough, to "see" cancer before a doctor with a microscope can recognize it.

A Deep Neural Networks model might be able to solve this problem.

A network that can say, with high accuracy, "this image is showing a cancer" or "this image is showing a benign tumour" almost the same way that a classification model can say "this is a dog" or "this is a cat." In this case, we can build a convolutional neural network (or CNN) to detect edges, boundaries, shapes, and attenuation of moles as features and use it to detect if a mole is 'cancerous' or 'not cancerous'.

In this Deep Neural Networks model, lower layers detect features like texture, irregular boundary structures, and so on. Intermediate layers may recognize entire lesions of different shapes and sizes. Ok, all of this is very good. So now let's put this all together to see what we should have.

Basically, if we look at the pipeline of our deep learning model, we can see the following phases: First, pre-processing input data.

Second, training the deep learning model. And third, inference and deployment of the model. First we have to convert the images into a readable and proper format for our network. Then, an untrained network is fed with a big dataset of images in the Training phase, prompting the network to learn.

Finally, we use the trained model in the Inference phase, which classifies a new image by inferring its similarity to the trained model. This model can be deployed and used as a melanoma detector. But we should consider that, in general, this pipeline is very slow for 3 reasons: First, training a deep neural network is basically a slow process. Second, building a deep neural network is an iterative process for data scientists, that is, it needs optimization and tuning, and data scientists need to run it many times to make it ready to be used.

And third, the trained model needs to get updated periodically, because new data needs to be added to the training set. Becausenald of these factors, the process is generally very slow. This leads to the question, "How slow is the pipeline?" As a rule of thumb: the smaller your system, the more time you'll need to get a trained model that performs well enough.

For example, building a recognition model might take multiple days with a simple Intel x86 architecture. And that's not surprising, given that we're working with a training set of a billion pieces of data, including radiological and photographic images of melanomas and carcinogenic lesions and tumors. Given these volumes, it's not surprising that it would take a few days to train the neural network, each time a data scientist wants to incorporate new data. Again, this wouldn't be so bad if it we were a do it once. But the process is an iterative one, in which we're trying to set and reset parameters, and run it again and again trying to make it better.

Now, think about a system that can complete this process in just 4 hours. Imagine what you could do with your model then. You might decide to train your model more frequently, which would make it much more accurate.

Or you can run the model many times and build two models that will merge to become an 'ensemble'

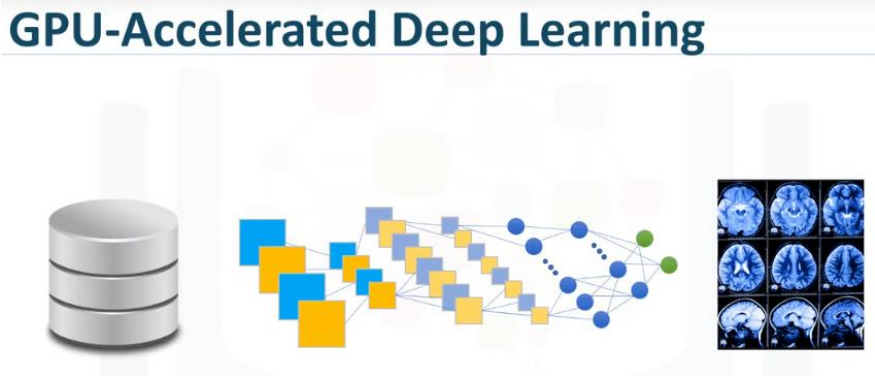
model (which is data science talk for when people mix multiple Machine Learning nets into a higher level one). So, training time is one of the key components to productivity, and indeed a key metric for deep learning.

But, how can we accomplish this? We'll answer this question in the next video.

Thanks for watching this video.

End of transcript. Skip to the start.

MODULE 2 – HARDWARE ACELERATED DEEP LEARNING



Hello, and welcome! In the following videos, we'll be reviewing GPU-accelerated deep learning. Have you ever tried to train a complex deep learning model with huge data? You should expect hours, days, or sometimes weeks, to train a complex model with a large dataset.

So, what's the solution? Well, you should use accelerated hardware.

For example, you can use Google's Tensor Processing Unit (or TPU) or Nvidia GPU to accelerate your deep neural network computation time.

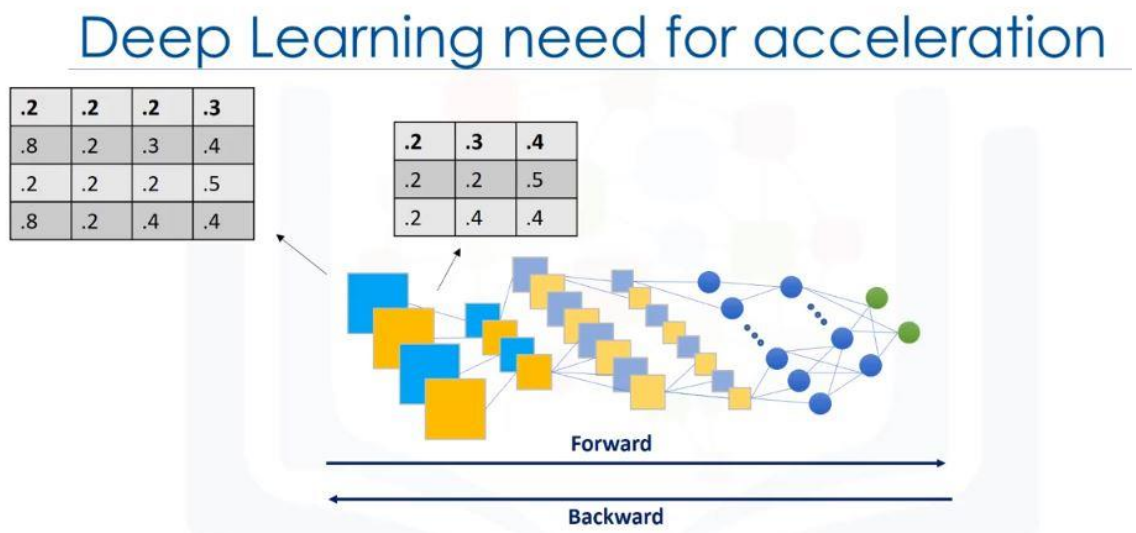
In the following videos, I will talk about what kind of hardware would be better for doing deep learning, and how GPU can help to accelerate it.

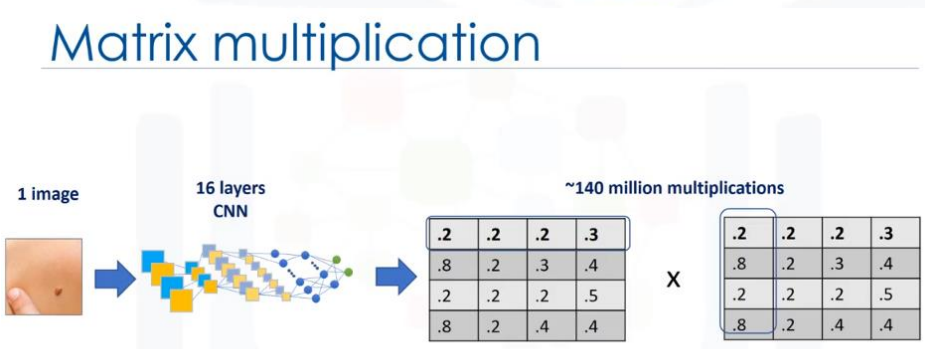
I assume that you have fundamental knowledge of deep learning concepts.

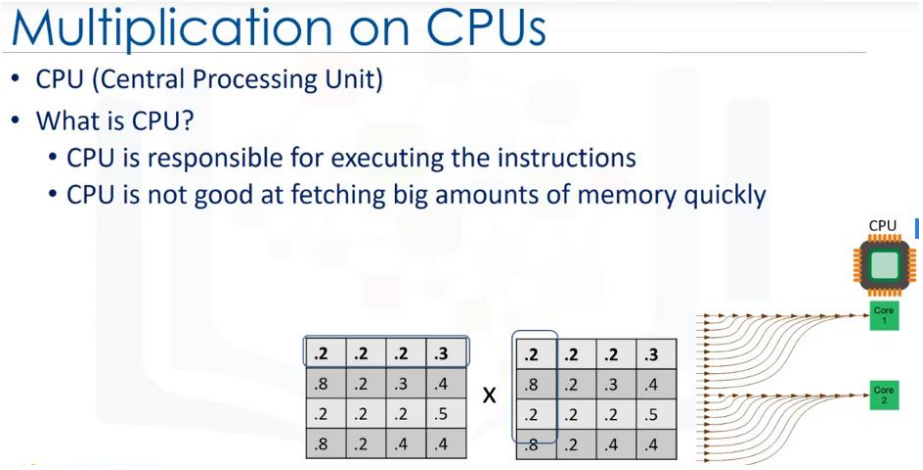
If not, first go through the previous videos or courses about deep learning and then come back to this module. Thanks for watching this video.

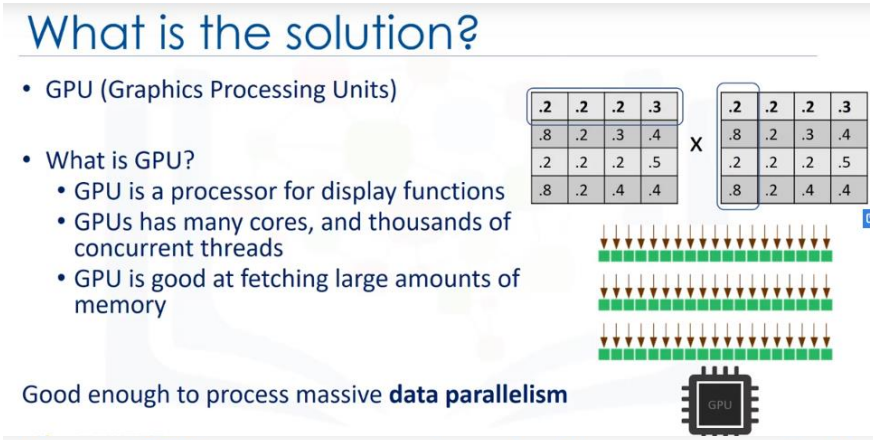
End of transcript. Skip to the start.

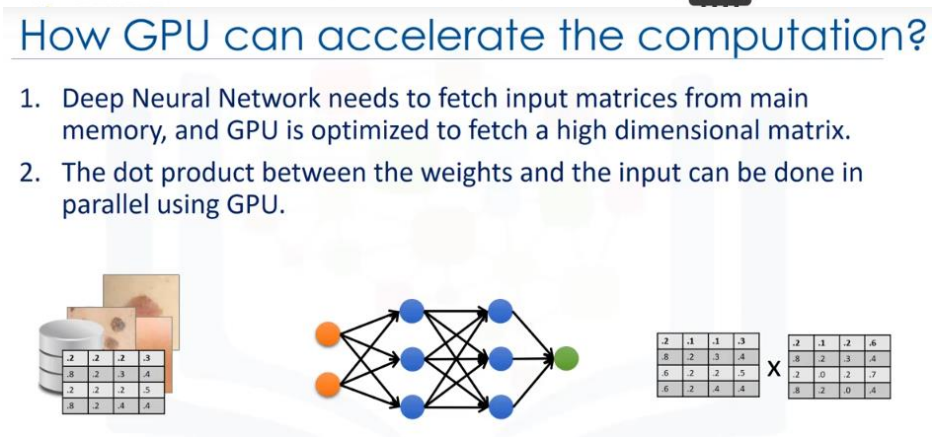
HOW TO ACCELERATE A DEEP LEARNING MODEL?











Start of transcript. Skip to the end.
In this video, we'll examine how we can accelerate the training process.
As mentioned before, some machine learning, and most of Deep Learning networks, need a lot of computational power for building a model.
We also mentioned that of the three phases in the deep learning pipeline, the training phase of the modeling is the most intensive task, and the most time consuming one.
Why? It essentially comes down to two reasons: First, deep learning is an iterative process.
When you train a deep learning model, two main operations are performed: Forward Pass and Backward Pass. In forward pass, input is passed through the
neural network and after processing the input, an output is generated.
Whereas in backward pass, we update the weights of the neural network on the basis of the error(s) we get in the forward pass. Second, deep learning involves heavy computations.
For example, in a Convolutional Neural Network, each pixel within a single image becomes a feature point after being multiplied by the color channel.
We can consider the first array as the input to the neural network, and the second array can be considered as weights or a filter of the network.
The size of these matrices are usually very big, that is, the data is high-dimensional here. So, considering that training is an iterative
process, and Neural Networks have usually many weights, which should get updated with each iteration, it involves expensive computations that are mostly matrix multiplication.
Therefore, Deep Learning requires much computing power.
Let's assume we just want to run a dot product of a 2 by 2 matrix; it includes many multiplications,
right? The first row of the first matrix multiplied

by the first column of the second matrix, and so on.
The challenging part is that we have many of these multiplications in deep learning.
For example, a convolutional neural network with 16 layers, has around 140 million weights and biases. Now imagine how many matrix multiplications
you would have to do to a pass of just one image to this network!
And think how many images you have in your training data set, and how many iterations you should do till your network learns properly. It would take a long time to train this kind of model if we use a sequential computational approach.
Now, think of doing all the operations at the same time instead of doing them one after the other. That is, the multiplications can be run in
parallel, and we then integrate the results. It would most definitively speed up the process.
This is called parallelism, which is very desirable in expensive computational tasks.
GPU will do the parallelism for us. But before we explain how GPU can do it for us, let's see why CPU cannot? The CPU (or Central Processing Unit) is the
part of a computer system that is known as the processor or microprocessor.
You probably heard of the x86, which is one of the more prevalent CPUs by Intel.
The CPU is responsible for executing a sequence of stored instructions, which in our case are multiplications. We need to fetch data and instructions from
main memory to be run by the CPU. CPU is good at fetching small amounts of memory quickly, but not very good for big chunks of data, like big matrices, which are needed for deep learning. CPUs run tasks sequentially, rather than in
parallel, even though they have 2 or 4 cores. Some companies used to build multiple clusters of CPUs to have a powerful system to do their processing in parallel, for example, built for training huge nets. These systems are usually very expensive and
as such, most businesses can't afford them. So, we can conclude that CPUs are not fast enough for operations on big chunks of data and they're not the proper use for high parallelism, as they are very slow for these kinds of tasks.
So, what is the solution? Well, the solution is the use of Graphics
Processing Units, or GPUs. The two key questions to ask here are: "What
is a GPU and how can it accelerate the computation?" To answer the first question, a GPU is a chip
(i.e., a processor) traditionally designed and specialized for rendering images, animations and video for the computer's screen. The second question is answered by the fact
that GPUs have many cores, sometimes up to 1000 cores, so they can handle many computations.
Also, GPUs are good at fetching large amounts of memory.
Although GPUs were originally invented and used for better and more general graphics processing, they were subsequently found to fit scientific computing as well.
We can say, GPU computing is "A Paradigm Shift in Programming," designed to efficiently process massive data parallelism with huge datasets.
Technically, there are many intricate reasons why a GPU is much better for handling matrices
multiplication than a CPU. However, to thoroughly review all those intricacies is beyond the scope of this video. So let's conclude with the main reasons
why GPUs are a good match for deep learning. First, recall that Deep Neural Networks need
to fetch input images as matrices from main memory. And GPUs perform very well at fetching
big chunks of memory. So, we can say that GPUs are optimized to properly fetch such high-dimensional matrices. Second, GPUs are the proper choice for parallelism operations on matrices, that is, they are very good where the same code runs on different sections of the same array. Now, recall that a Deep Neural Network is
a matrix of weights where the dot product between the weights and the input image needs

to be done many times. GPUs can also use their ability for parallelism to do all the multiplications
at the same time, instead of doing them one after the other. This type of concurrent processing
is exactly what's required in deep learning. So, the GPU parallelism feature reduces the computation times of the dot product of 18 big matrices.
Thanks for watching this video.
End of transcript. Skip to the start.

RUNNING TENSORFLOW OPERATIONS ON CPUs VS GPUS

Deep Learning with GPU vs CPU

I explained about the difference between GPU and CPU, but lets point out some important parts of using GPU here:

Performance of GPU vs CPU

Most of Deep Learning models, especially in their training phase, involve a lot of matrix and vector multiplications that can parallelized. In this case, GPUs can overperform CPUs, because GPUs were designed to handle these kind of matrix operations in parallel!

Why GPU overperforms CPU?

A single core CPU takes a matrix operation in serial, one element at a time. But, a single GPU could have hundreds or thousands of cores, while a CPU typically has no more than a few cores.

What types of operations should I send to the GPU?

Basically, if a step of the process encompass "do this mathematical operation many times", then it is a good candidate operation to send it to be run on the GPU. For example,

- Matrix multiplication
- Computing the inverse of a matrix.
- Gradient calculation, which are computed on multiple GPUs individually and are averaged on CPU

When should not use GPU?

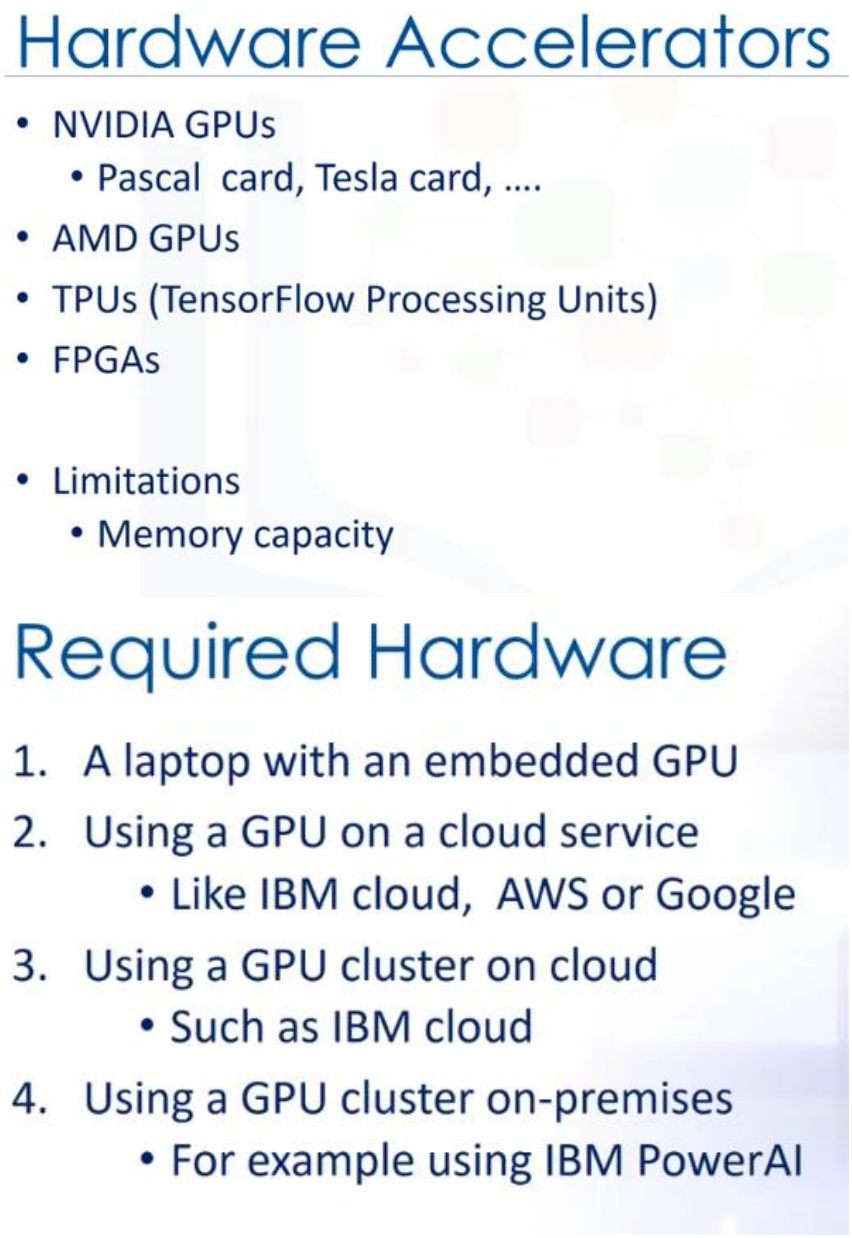
GPUs don't have direct access to the rest of your computer (except, of course for the display). Due to this, if you are running a command on a GPU, you need to copy all of the data to the GPU first, then do the operation, then copy the result back to your computer's main memory. TensorFlow handles this under the hood, so the code is simple, but the work still needs to be performed.

How to use GPU with TensorFlow?

It is important to notice that if both CPU and GPU are available on the machine that you are running a notebook, and if a TensorFlow operation has both CPU and GPU implementations, the GPU devices will be given priority when the operation is assigned to a device. Lets practice it in Labs.

MODULE 3 – DEEP LEARNING IN THE CLOUD

DEEP LEARNING IN THE CLOUD



Start of transcript. Skip to the end.
The key question now, is: where should I get an accelerator, such as a GPU, for running my deep learning pipeline? In this video we'll examine and compare different hardware accelerators. Let's quickly take a look at some well-recognized accelerating hardware (and their associated software) that have succeeded in reducing the training time, several times over. NVIDIA is one of the main vendors of GPU and with CUDA software on top of that, we see it on most platforms.
Let me explain CUDA. You don't have to understand low level graphics

processing to implement your algorithm code on a GPU.
Nvidia has a high-level language, known as CUDA, that helps you write programs from graphic
processors. Looking at different NVIDIA GPUs, you'll
find various features, architectures and cards, such as Pascal cards and Tesla, in which their performance and speed is highly dependent on their memory bandwidth.
Again, just recall that the most important feature of GPUs, which made them a good match for your deep learning, is memory bandwidth to fetch high dimensional data.
So, we can say it's an important metric for a GPU.
AMD cards are also an option, but AMD's OpenCL, the software on top of AMD cards, is not very popular among developers, who are working on deep learning libraries right now. TensorFlow Processing Units (or TPUs) is the
Google hardware accelerator solution developed specifically for TensorFlow, which is Google's
open-source machine learning framework. TPU promises an acceleration over-and-above the current GPUs. FPGAs (or Field Programmable Gate Arrays)
are programmable or customizable hardware. That is, in contrast to a processor that has a fixed number of resources, you build a custom circuit with resources and hence exploit all the parallelism in a program. Intel is working on creating faster FPGAs,
which may provide high flexibility. Now the question is that among all of these existing options, what kind of accelerator should you get?
You can use Google's TPUs, Nvidia GPU or even FPGA to accelerate your deep learning network computation time. These chips are particularly designed to support
the training of neural networks, as well as the use of trained networks (that is, inference).
But there are some general limitations in these accelerators as well.
Let me start by focusing on GPUs, as they're the most-popular accelerator.
GPUs have two key limitations: The first involves limited memory capacity.
Yes, GPUs are very fast for data parallelism and, as such, we can take full advantage of their massive computing power. That said, we still need to store the data
inside the GPU memory in order to access and process it.
Unfortunately, GPUs currently have up to 16GB of memory, so this is not practical for very large datasets. In this case you have to read data from system
memory, and it's a huge overhead. So, you need a platform that can handle fast memory access in system memory, and also fast data exchange between GPUs.
The second limitation of GPUs is that you cannot easily buy these accelerators and embed them into your local machine. They're usually expensive and there are
some dependencies and incompatibilities, which is the same as most hardware.
Also, sometimes, you need a number of GPUs to handle your big datasets.
So, these accelerators are not readily accessible, at least not for now.
At this point, the question is, where can I get a GPU to train my deep learning network?
First, you always have to keep in mind that deep learning requires a lot of computational power to run on, but it's totally dependent on the task at hand.
You don't really need to buy or find a large Datacenter with many GPUs to run your model.
There are a number of options that are available to you and you really should look around.
The first one to consider is that some personal computers have an embedded GPU.
For example, a laptop with a recent NVIDIA GPU should support CUDA, so you can at least play around with deep learning networks, and you can use it to train your deep learning, to some extent, but usually not enough to solve particularly deep learning problems.
In this case, you need to scale down the dataset, or the model, to something that fits on a laptop, which often delivers significantly worse results.
Many cloud providers are offering GPU services as virtual machines with GPUs that you can use, like IBM cloud, Amazon AWS, or Google cloud.

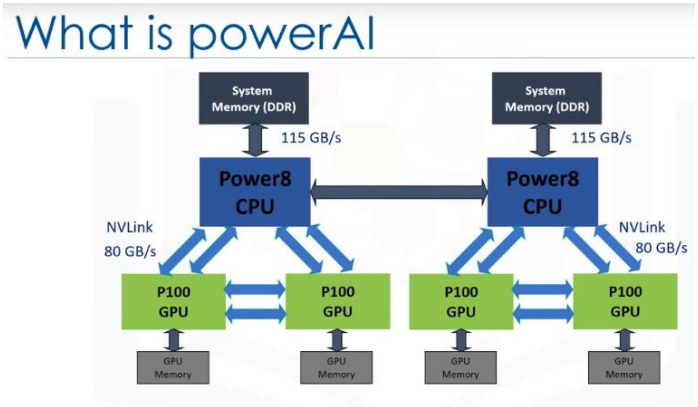
The difference between these providers is usually the hardware that they offer, and of course the price that you pay for it. In general, these are good options to start building and training your model, as you can configure an instance with the ratio of processors, memory and GPUs you might need. However, you should consider that you need to upload all your data on the cloud and train the network on the cloud, which sometimes is a hassle. Usually, you can find services that offer you single or multi-GPU access. My suggestion is to use a fast-enough single GPU to do experiments with sample data to verify many things before going full scale. After the experiments, you'll need an 8-or 16-GPU instance to finish the job. If your data is very big in terms of volume and computational requirements, you'll need a very large computational system to handle your data. In this case, you'll likely need a cluster of GPUs to distribute the whole computational workload. Using a GPU cluster and doing multi-GPU computing on the cloud, or on one of the cloud services, such as IBM-cloud, could be a good solution for such a scenario. Although using a cloud service is a good choice for getting started and solving some problems, you should keep in mind that building 'cloud-based deep learning' could be very costly -- especially if you need to train models for more than 1000 hours. In this case, using a GPU cluster or doing multi-GPU computing On-premise is the best option, as it'll allow you to keep your data locally and do computations on your servers, which becomes cost-effective. Additionally, your data might be sensitive, with need to analyze it yourself On-premises. If so, you may not feel comfortable to upload it into public clouds. In this case, you'll need to use an in-house system with GPU support, such as IBM PowerAI. Thanks for watching this video. End of transcript. Skip to the start.

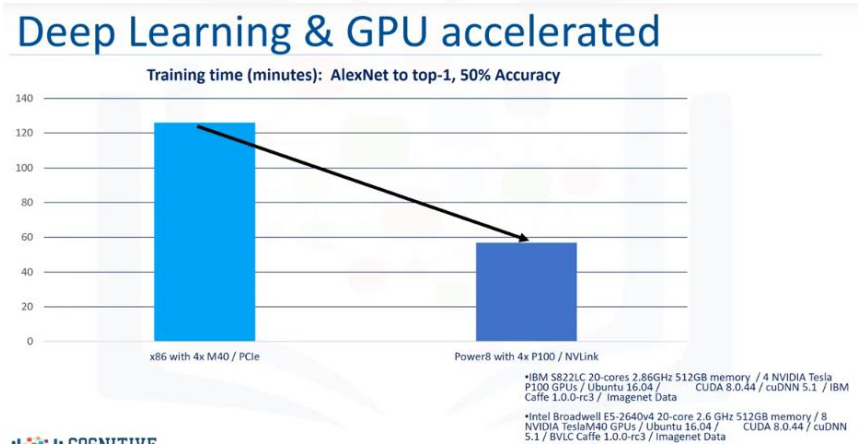
Deep Leaning in the cloud?

We mentioned that you should use accelerated hardware, such as Nvidia GPU to accelerate your neural network computations time on the could. But the problem is that your data might be sensitive and you may not feel comfortable to upload it into public cloud, and you need to analyze it on-premise. In this case, you need to use an in-house system with GPU support. One solution is using IBM's Power Systems with Nvidia GPU, and PowerAI. Built on IBM's Power Systems, PowerAI is a scalable software platform that accelerates deep learning and AI with blazing performance for individual users or enterprises. The PowerAI platform supports popular machine learning libraries and dependencies including Tensorflow, Caffe, Torch, and Theano.

Notice: PowerAI requires installation on IBM Power Systems S822LC for HPC server infrastructure. However, you do not have to install it for running the labs in this course, and you can easily click on 'Start Lab' in the lab section, to run the deep learning notebooks on PowerA

HOW DOES ONE USE GPU





Deep Leaning in the cloud?

We mentioned that you should use accelerated hardware, such as Nvidia GPU to accelerate your neural network computations time on the could. But the problem is that your data might be sensitive and you may not feel comfortable to upload it into public cloud, and you need to analyze it on-premise. In this case, you need to use an in-house system with GPU support. One solution is using IBM's Power Systems with Nvidia GPU, and PowerAI. Built on IBM's Power Systems, PowerAI is a scalable software platform that accelerates deep learning and AI with blazing performance for individual users or enterprises. The PowerAI platform supports popular machine learning libraries and dependencies including Tensorflow, Caffe, Torch, and Theano.

Notice: PowerAI requires installation on IBM Power Systems S822LC for HPC server infrastructure. However, you do not have to install it for running the labs in this course, and you can easily click on 'Start Lab' in the lab section, to run the deep learning notebooks on PowerAI.

Start of transcript. Skip to the end.
In this video, we look at the IBM offering for accelerating deep learning, and see how it has solved the limitations of using GPUs. IBM offers a platform called PowerAI for deep learning. On this platform, the problem of memory access is addressed. To start the story, consider that data, or rather Big Data, is located in system memory of your computer.
When an application starts to analyze this data using a GPU, you have to move chunks of it over to the GPU via the CPU. As you can imagine, moving data from the CPU

to the GPU creates a bottleneck because most of the data is going through a thin pipe, called PCIe. So, we can say that bandwidth is the problem here, which highlights a simple truth: it's the data flow that will determine the final performance for your workload. PowerAI takes advantage of NVLink to increase system bandwidth. What is NVlink? Let me explain.

NVLink is a high bandwidth protocol that enables faster GPU-GPU communication, by providing multiple point-to-point connections. For Deep Learning workloads, this decreases memory-cache copy time, reducing GPU wait, and allowing the GPUs to execute more training cycles in a shorter amount of time. You should also note that, on the PowerAI platform, full NVLink connectivity between CPU and GPU, provides the same wide data path between the GPU and system memory. This allows workloads using large training datasets a faster way to "reload." This is particularly important because large datasets can't be stored only in a GPU's limited memory.

The outcome is faster training and the ability to train with larger datasets for improved accuracy. So, in brief, two NVLink connections between GPUs reduces GPU wait, and high-speed connections between each GPU and CPU, enables the machine to perform fast memory access to large datasets in system memory.

IBM currently uses Tesla P100 and Power8 for this specific customization.

Built on IBM's Power Systems, PowerAI is a scalable software platform that accelerates deep learning and AI (or artificial intelligence) with blazing performance for individual users or enterprises. ThePowerAI platform supports popular machine learning libraries and dependencies, including TensorFlow, Caffe, Torch, and Theano.

I'm sure you already appreciate that deep learning software frameworks scale well with GPU accelerators. Nevertheless, it was important to show you how much faster the GPU acceleration can be using PowerAI.

Based on a benchmark, this new architecture leads to much faster training time.

Let's compare the training time of running a deep learning sample on hardware accelerators by PCIe's and Nvlink to understand the difference. Please notice that it's not comparing training time with and without GPUs, it's about how we can take full advantage of GPU acceleration for deep learning, using better bandwidth. Alexnet is a large, deep convolutional neural network to classify more than 1 million high-resolution images of a famous training set into the 1000 different classes. Today, this network is a common benchmark for deep-learning training. Training this network using 4 GPUs, allows you the capacity to reach 50 percent accuracy in almost 2 hours. But, running the same network on PowerAI shows an approximately two times faster learning, which is a promising result.

If you need more information on how PowerAI works, you can go to the links shown below the video. Thanks for watching.

End of transcript. Skip to the start.

WHAT PowerAI can do FOR YOUR BUSINESS?

PowerAI makes deep learning, machine learning, and AI more accessible and more performant. By combining this software platform for deep learning with IBM® Power Systems™, enterprises can rapidly deploy a fully optimized and supported platform for machine learning with blazing performance. The PowerAI platform includes the most popular

machine learning frameworks and their dependencies, and it is built for easy and rapid deployment.
0:00 / 2:55

MODULE 4 – DISTRIBUTED DEEP LEARNING

Distributed Deep Learning

Have you ever tried to train a complex deep learning models with huge data? You should expect days and weeks sometimes to train a complex model with large dataset. What about if you distribute the process not only on a few GPUs, but also on multiple servers?

The popular open-source deep-learning frameworks such as Tnesorflow and Caffe do not seem to run as efficiently across multiple servers. So, while most data scientists are using servers with four or eight GPUs, they can't scale beyond that single node. For example, when you try to train a model with the ImageNet-22K data set using a ResNet-101 model, it may take up to 16 days on a single server (S822LC for High Performance Computing) with four NVIDIA P100 GPU accelerators.

16 days – that's a lot of time you could be spending elsewhere.

And since model training is an iterative task, where a data scientist tweaks hyper-parameters, models, and even the input data, and trains the AI models multiple times, these kinds of long training runs delay time to insight and can limit productivity.

Most popular deep learning frameworks scale to multiple GPUs in a server, but not to multiple servers with GPUs. Distributed Deep Learning tries to reduce training times for large models with large data sets. It solves grand-challenge scaling issue by distributing deep learning training **across large numbers of servers and GPUs**. For example, imagine that you can run your deep learning framework over 256 GPUs in 64 servers. It would really decrease the training time greatly.

Distributed Deep Learning is a bunch of software and algorithms that automate and optimize the parallelization of very large and complex computing task across hundreds of GPU accelerators attached to dozens of servers.

Summary: IBM Research publishes in arXiv close to ideal scaling with new distributed deep learning software which achieved record communication overhead and 95 percent scaling efficiency on the Caffe deep learning framework over 256 NVIDIA GPUs in 64 IBM Power systems. Previous best scaling was demonstrated by Facebook AI Research of 89 percent for a training run on Caffe2, at higher communication overhead. IBM Research also beat Facebook's time by training the model in 50 minutes, versus the 1 hour Facebook took. Using this software, IBM Research achieved a new image recognition accuracy of 33.8 percent for a neural network trained on a very large data set (7.5M images). The previous record published by Microsoft demonstrated 29.8 percent accuracy.

A technical preview of this IBM Research Distributed Deep Learning code is available today in IBM PowerAI 4.0 distribution for TensorFlow and Caffe.

Deep learning is a widely used AI method to help computers understand and extract meaning from images and sounds through which humans experience much of the world. It holds promise to fuel breakthroughs in everything from consumer mobile app experiences to medical imaging diagnostics. But progress in accuracy and the practicality of deploying deep learning at scale is gated by technical

challenges, such as the need to run massive and complex deep learning based AI models – a process for which training times are measured in days and weeks.

For our part, my team in IBM Research has been focused on reducing these training times for large models with large data sets. Our objective is to reduce the wait-time associated with deep learning training from days or hours to minutes or seconds, and enable improved accuracy of these AI models. To achieve this, we are tackling grand-challenge scale issues in distributing deep learning across large numbers of servers and NVIDIA GPUs.

Most popular deep learning frameworks scale to multiple GPUs in a server, but not to multiple servers with GPUs. Specifically, our team (Minsik Cho, Uli Finkler, David Kung, Sameer Kumar, David Kung, Vaibhav Saxena, Dheeraj Sreedhar) wrote software and algorithms that automate and optimize the parallelization of this very large and complex computing task across hundreds of GPU accelerators attached to dozens of servers.
IBM Fellow Hillery Hunter develops new software enabling unprecedented GPU processing speeds.

IBM Fellow Hillery Hunter develops new software enabling unprecedented GPU processing speeds.

Our software does deep learning training fully synchronously with very low communication overhead. As a result, when we scaled to a large cluster with 100s of NVIDAI GPUs, it yielded record image recognition accuracy of 33.8 percent on 7.5M images from the ImageNet-22k dataset vs the previous best published result of 29.8 percent by Microsoft. A 4 percent increase in accuracy is a big leap forward; typical improvements in the past have been less than 1 percent. Our innovative distributed deep learning (DDL) approach enabled us to not just improve accuracy, but also to train a ResNet-101 neural network model in just 7 hours, by leveraging the power of 10s of servers, equipped with 100s of NVIDIA GPUs; Microsoft took 10 days to train the same model. This achievement required we create the DDL code and algorithms to overcome issues inherent to scaling these otherwise powerful deep learning frameworks.

These results are on a benchmark designed to test deep learning algorithms and systems to the extreme, so while 33.8 percent might not sound like a lot, it's a result that is noticeably higher than prior publications. Given any random image, this trained AI model will gives its top choice object (Top-1 accuracy), amongst 22,000 options, with an accuracy of 33.8 percent. Our technology will enable other AI models trained for specific tasks, such as detecting cancer cells in medical images, to be much more accurate and trained in hours, re-trained in seconds.

As Facebook AI Research described the problem in a June 2017 research paper explaining their own excellent result using a smaller data set (ImageNet 1k) and a smaller neural network (ResNet 50):

"Deep learning thrives with large neural networks and large datasets. However, larger networks and larger datasets result in longer training times that impede research and development progress."

Ironically, this problem of orchestrating and optimizing a deep learning problem across many servers is made much more difficult as GPUs get faster. This has created a functional gap in deep learning systems that drove us to create a new class of DDL software to make it possible to run popular open source codes like Tensorflow, Caffe, Torch and Chainer over massive scale neural networks and data sets with very high performance and very high accuracy.

Here a variant of the "Blind Men and the Elephant" parable is helpful in describing the problem that we are solving and context for the promising early results we have achieved. Per Wikipedia:

"…Each blind man feels a different part of the elephant body, but only one part, such as the side or the tusk. They then describe the elephant based on their partial experience and their descriptions are in complete disagreement on what an elephant is."

Now, despite initial disagreement, if these people are given enough time, they can share enough information to piece together a pretty accurate collective picture of an elephant.

Similarly, if you have a bunch of GPUs slogging through the task of processing elements of a deep learning training problem – in parallel over days or weeks, as is typically the case today – you can synch these learning results fairly easily.

But as GPUs get much faster, they learn much faster, and they have to share their learning with all of the other GPUs at a rate that isn't possible with conventional software. This puts stress on the system network and is a tough technical problem. Basically, smarter and faster learners (the GPUs) need a better means of communicating, or they get out of sync and spend the majority of time waiting for each other's results. So, you get no speedup–and potentially even degraded performance–from using more, faster-learning GPUs.

Our ability to address this functional gap with (DDL) software is most visible when you look at scaling efficiency or how close to perfect system performance scales as you add GPUs. This measurement provides a view into how well the 256 GPUs were "talking" about what each other were learning.

The best scaling for 256 GPUs shown before is by a team from Facebook AI Research (FAIR). FAIR used a smaller deep learning model, ResNet-50, on a smaller dataset ImageNet-1K, which has about 1.3 million images, both of which reduce computational complexity, and used a larger batch size of 8192, and achieved 89 percent scaling efficiency on a 256 NVIDIA P100 GPU accelerated cluster using the Caffe2 deep learning software. For a ResNet-50 model and same dataset as Facebook, the IBM Research DDL software achieved an efficiency of 95 percent using Caffe as shown in the chart below. This was run on a cluster of 64 "Minsky" Power S822LC systems, with four NVIDIA P100 GPUs each.

Scaling Performance of IBM DDL across 256 GPUs (log scale)

For training the much larger ResNet-101 model on 7.5M images from the ImageNet-22K data set, with an image batch size of 5120, we achieved a scaling efficiency of 88 percent.

We also achieved a record in fastest absolute training time of 50 minutes compared to Facebook's previous record of 1 hour. We trained the ResNet-50 model with ImageNet-1K model by scaling Torch using DDL to 256 GPUs. Facebook trained a similar model using Caffe2.

For developers and data scientists, the IBM Research (DDL) software presents an API (application programming interface) that each of the deep learning frameworks can hook into, to scale to multiple servers. A technical preview is available now in version 4 of the PowerAI enterprise deep learning software offering, making this cluster scaling feature available to any organization using deep learning for training their AI models. We expect that by making this DDL feature available to the AI community, we will see many more higher accuracy runs as others leverage the power of clusters for AI model training.