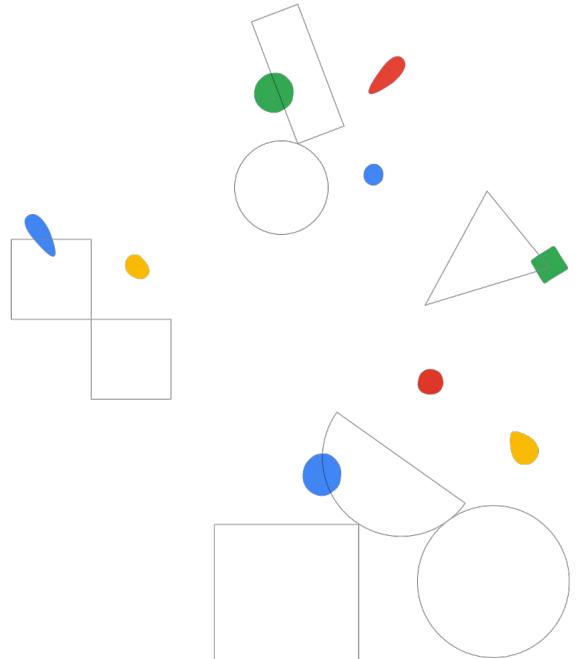




Specialization: Advanced Machine Learning on Google Cloud (Part 2)





Google Cloud

Designing High Performance ML Systems

Laurence Moroney

Hi, I'm Laurence and I'm a developer advocate on Google Brain, focused on TensorFlow.

In this course, you are learning the considerations behind architecting and implementing production machine learning systems.

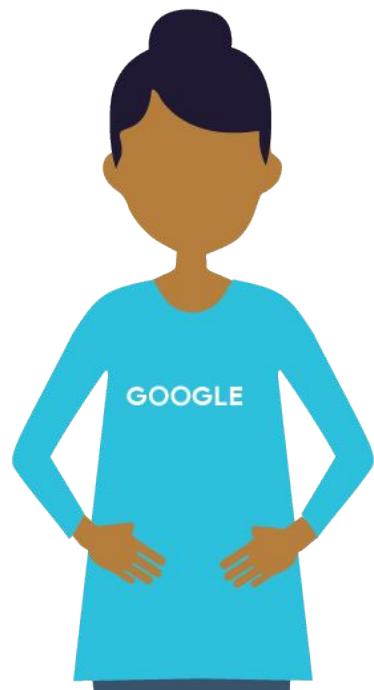
One key consideration is, of course, performance.

Learn how to...

Identify performance considerations for ML models

Choose appropriate ML infrastructure

Select a distribution strategy



In this module, you will learn how to identify performance considerations for machine learning models.

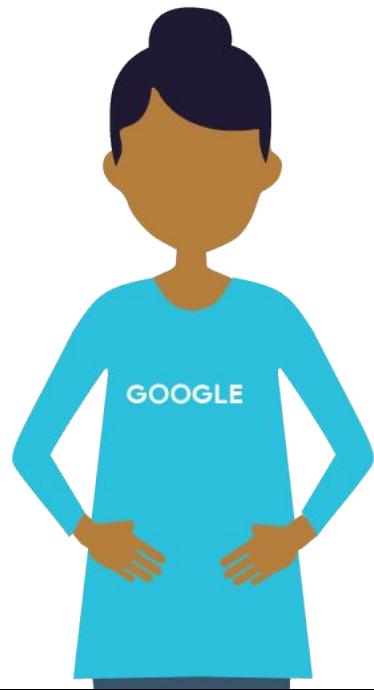
Machine learning models are not all identical. For some models, you will be focused on improving I/O performance, and on others, you will be focused on squeezing out more computational speed.

Learn how to...

Identify performance considerations for ML models

Choose appropriate ML infrastructure

Select a distribution strategy



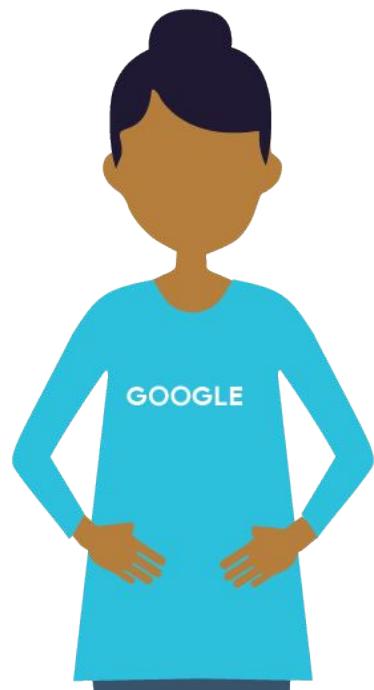
Depending on what your focus is, you will need different ML infrastructure -- whether you decide to scale out, with multiple machines or scale up on a single machine with a GPU and TPU. Sometimes, you'll do both, by using a machine with multiple accelerators attached to it.

Learn how to...

Identify performance
considerations for ML models

Choose appropriate ML
infrastructure

Select a distribution strategy



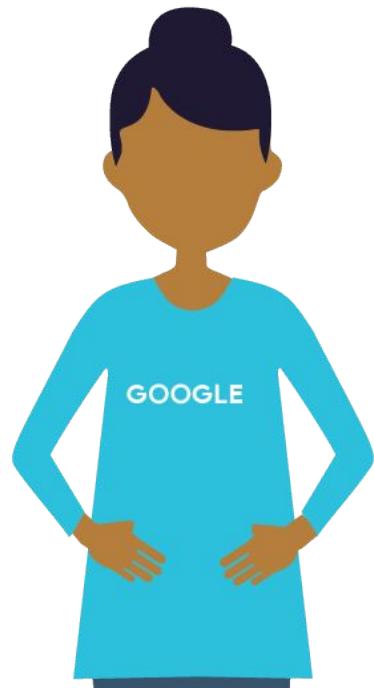
It's not just a hardware choice. The hardware you select will also inform your choice of a distribution strategy.

Agenda

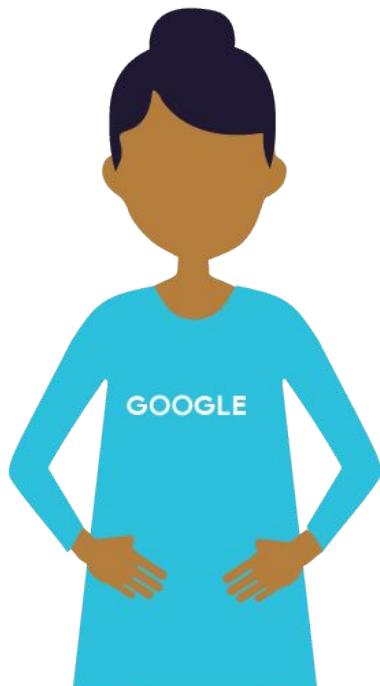
Distributed training

Faster input pipelines

Inference



We will start by talking about what high performance means in this context, and providing a high-level overview of distributed training architectures.

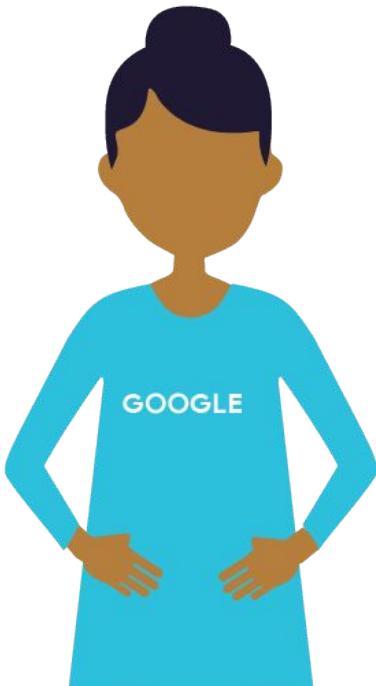


High Performance ML

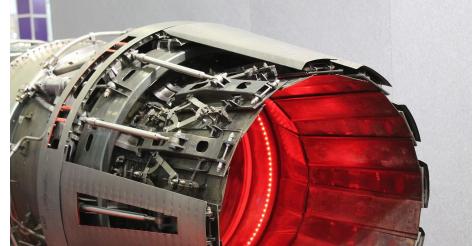


What does high-performance machine learning mean to you?

Image cc0: <https://pixabay.com/en/jet-engine-fighter-war-ali-2515044/>



High Performance ML



Does it mean “powerful”? the ability to handle large datasets?

Image cc0 engine:

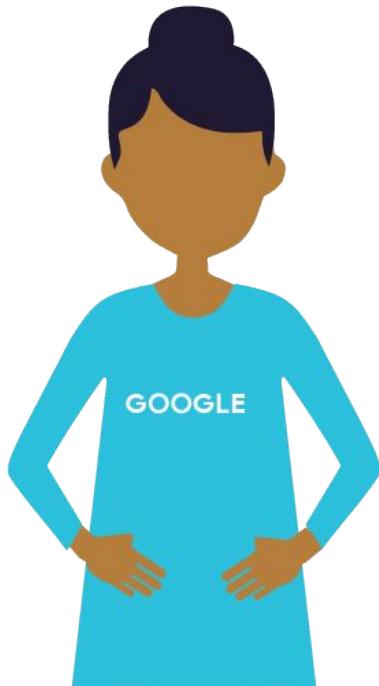
Image cc0: <https://pixabay.com/en/jet-engine-fighter-war-ali-2515044/>

Image cc0: rocket icon <https://pixabay.com/en/logo-icon-sign-symbol-rocket-3418127/>

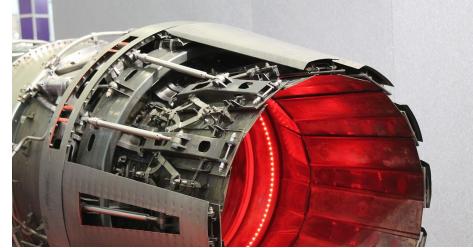
Image cc0: speed <https://pixabay.com/en/speedometer-tachometer-speed-148960/>

Image cc0: clock <https://pixabay.com/en/clock-black-wall-isolated-sign-163580/>

Image cc0: target <https://pixabay.com/en/dart-board-arrow-bull-s-eye-25780/>



High Performance ML



Doing it as fast as possible?

Image cc0 engine:

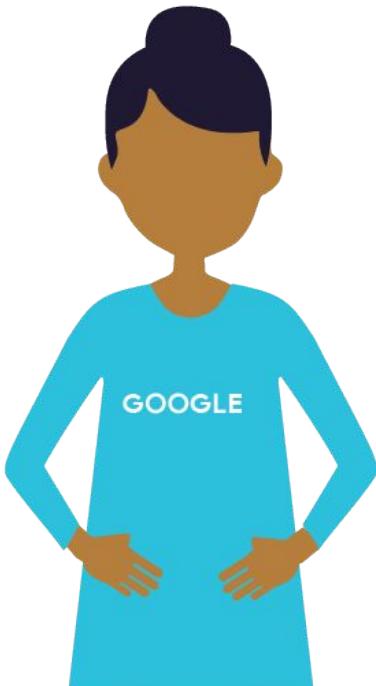
Image cc0: <https://pixabay.com/en/jet-engine-fighter-war-ali-2515044/>

Image cc0: rocket icon <https://pixabay.com/en/logo-icon-sign-symbol-rocket-3418127/>

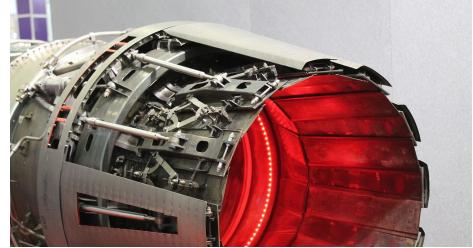
Image cc0: speed <https://pixabay.com/en/speedometer-tachometer-speed-148960/>

Image cc0: clock <https://pixabay.com/en/clock-black-wall-isolated-sign-163580/>

Image cc0: target <https://pixabay.com/en/dart-board-arrow-bull-s-eye-25780/>



High Performance ML



The ability to train for long periods of time?

Image cc0 engine:

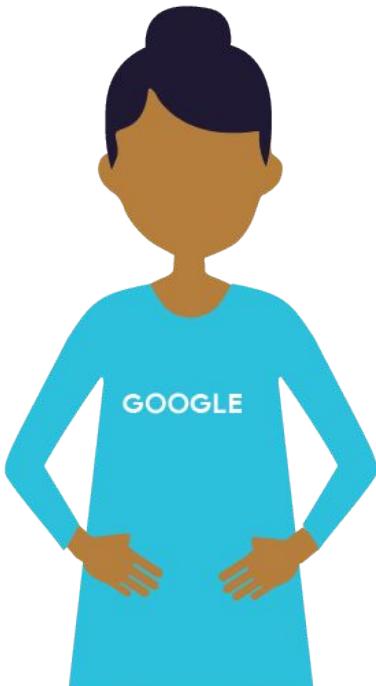
Image cc0: <https://pixabay.com/en/jet-engine-fighter-war-ali-2515044/>

Image cc0: rocket icon <https://pixabay.com/en/logo-icon-sign-symbol-rocket-3418127/>

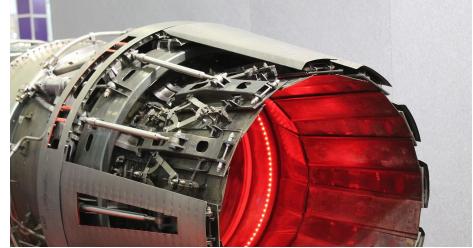
Image cc0: speed <https://pixabay.com/en/speedometer-tachometer-speed-148960/>

Image cc0: clock <https://pixabay.com/en/clock-black-wall-isolated-sign-163580/>

Image cc0: target <https://pixabay.com/en/dart-board-arrow-bull-s-eye-25780/>



High Performance ML



Achieving the best possible accuracy?

Image cc0 engine:

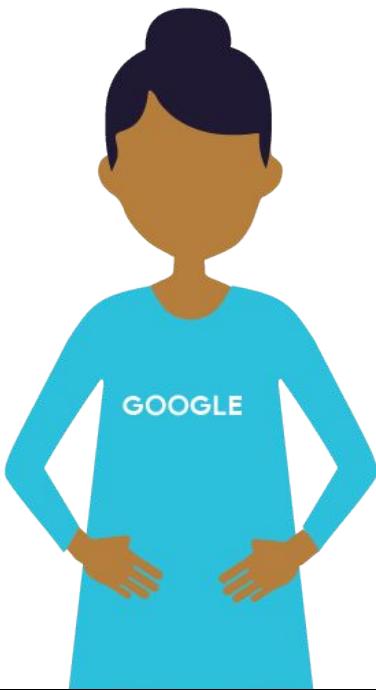
Image cc0: <https://pixabay.com/en/jet-engine-fighter-war-ali-2515044/>

Image cc0: rocket icon <https://pixabay.com/en/logo-icon-sign-symbol-rocket-3418127/>

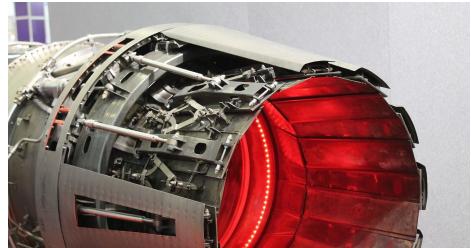
Image cc0: speed <https://pixabay.com/en/speedometer-tachometer-speed-148960/>

Image cc0: clock <https://pixabay.com/en/clock-black-wall-isolated-sign-163580/>

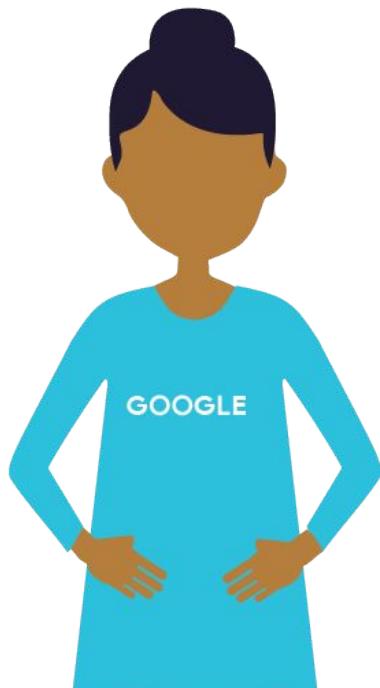
Image cc0: target <https://pixabay.com/en/dart-board-arrow-bull-s-eye-25780/>



Model Training Time



One key aspect is the time taken to train a model.



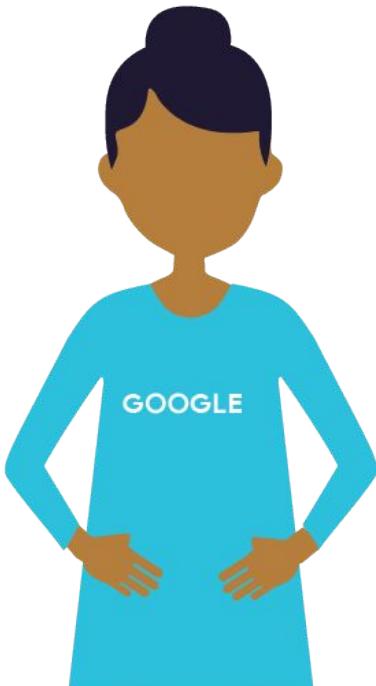
Model Training Time



If it takes 6 hours to train a model on some hardware/software architecture

Image cc0: Turtle:

<https://pixabay.com/en/amphibian-turtle-animal-armor-blur-1850190/>



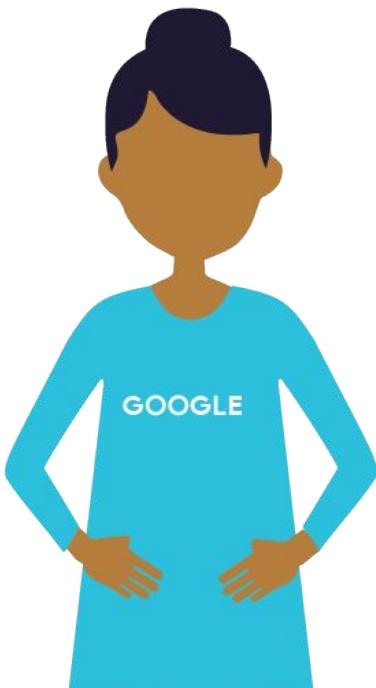
Model Training Time



but only 3 hours to train the same model to the same accuracy on some other hardware/software architecture, I think we will all agree that the second architecture is twice as performant as the first one.

Image cc0: Turtle:

<https://pixabay.com/en/amphibian-turtle-animal-armor-blur-1850190/>



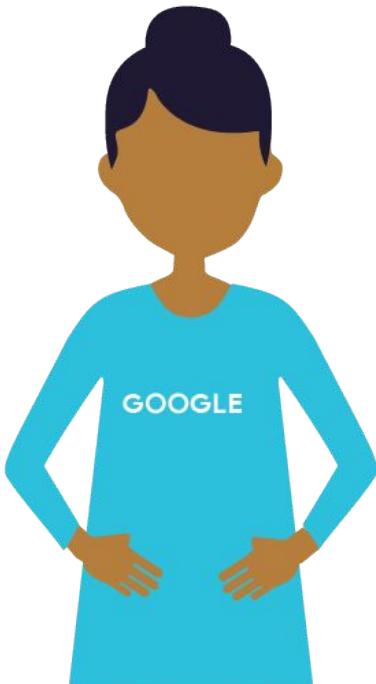
Model Training Time



Notice that I said “train the model to the same accuracy”. Throughout this module, we will assume that we are talking of models that have the same accuracy or RMSE or whatever your evaluation measure is. Obviously, when we talk about high-performance ML models, accuracy is important. We just aren’t going to consider that in this module. The rest of the courses in this specialization will look at how to build more accurate ML models, and there we will be looking at model architectures that will help us get to a desired accuracy. Here, in this course, we will look solely at infrastructure performance.

Image cc0: Turtle:

<https://pixabay.com/en/amphibian-turtle-animal-armor-blur-1850190/>



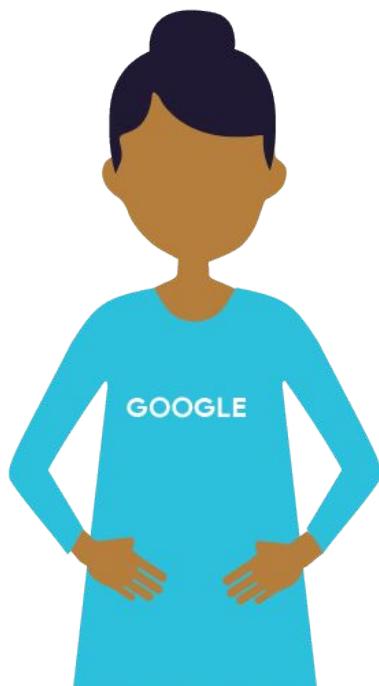
Optimizing your Training Budget



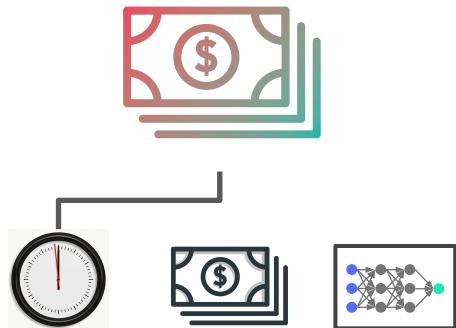
Besides time to train, there is another aspect. Budget. You often have a training budget. You might be able to train faster on better hardware, but the hardware might cost more, and so you might make the explicit choice to train on slightly slower infrastructure.

Image cc0, Budget icon:

<https://pixabay.com/en/money-icon-banking-financial-bag-2558681/>



Optimizing your Training Budget

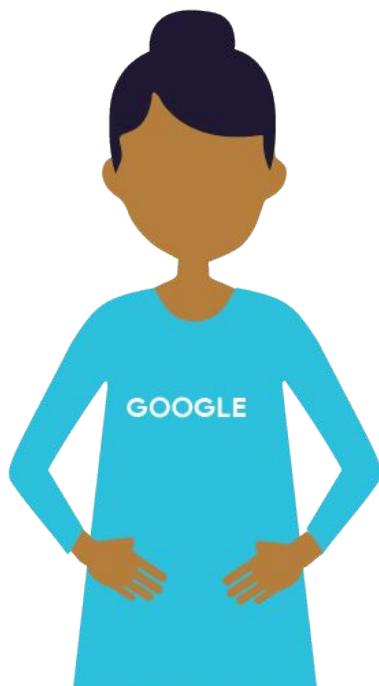


When it comes to your training budget, you have three considerations, three levers that you can adjust:

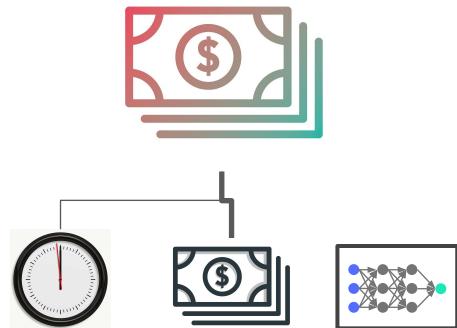
- **Time**
- Cost
- Scale

Image cc0, Budget icon:

<https://pixabay.com/en/money-icon-banking-financial-bag-2558681/>



Optimizing your Training Budget

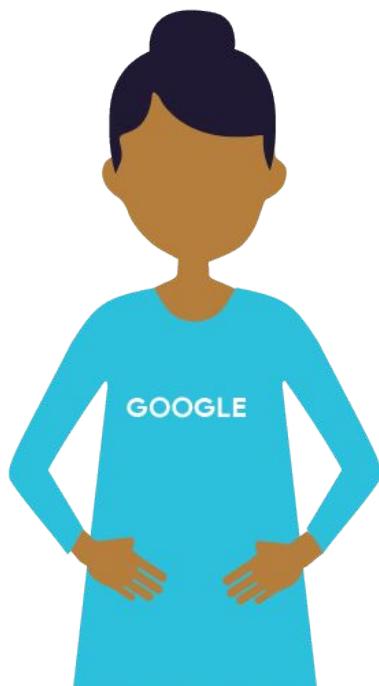


When it comes to your training budget, you have three considerations, three levers that you can adjust:

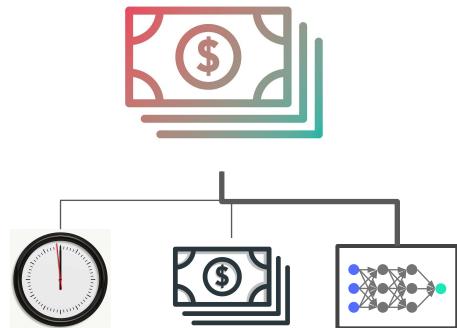
- Time
- Cost
- Scale

Image cc0, Budget icon:

<https://pixabay.com/en/money-icon-banking-financial-bag-2558681/>



Optimizing your Training Budget



When it comes to your training budget, you have three considerations, three levers that you can adjust:

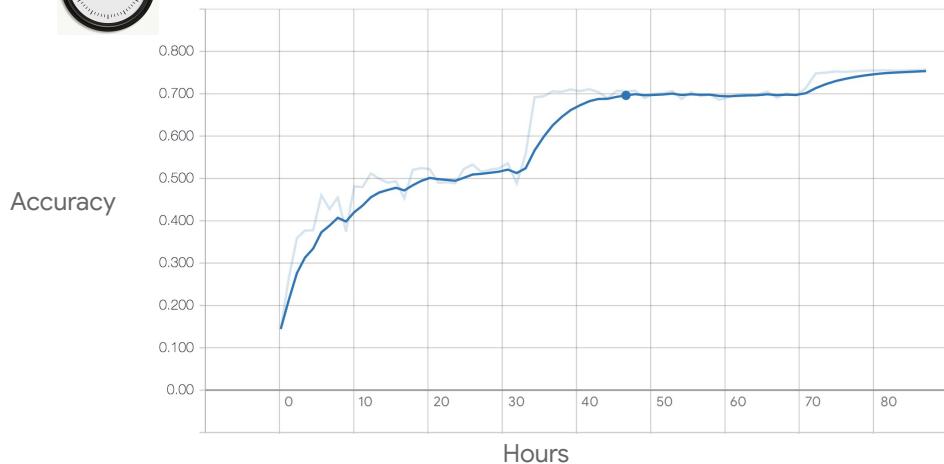
- Time
- Cost
- **Scale**

Image cc0, Budget icon:

<https://pixabay.com/en/money-icon-banking-financial-bag-2558681/>



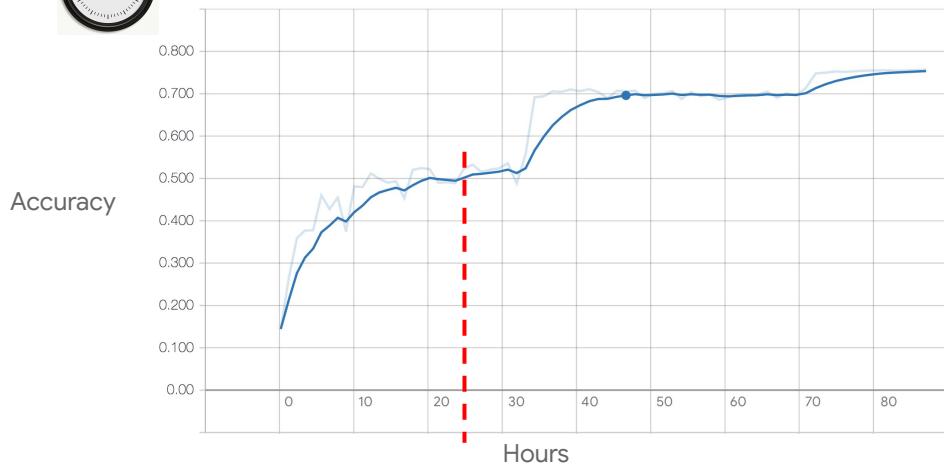
Model Training can take a long time



How long are you willing to spend on the model training? This might be driven by the business use case. If you are training a model everyday so as to recommend products to users the next day,



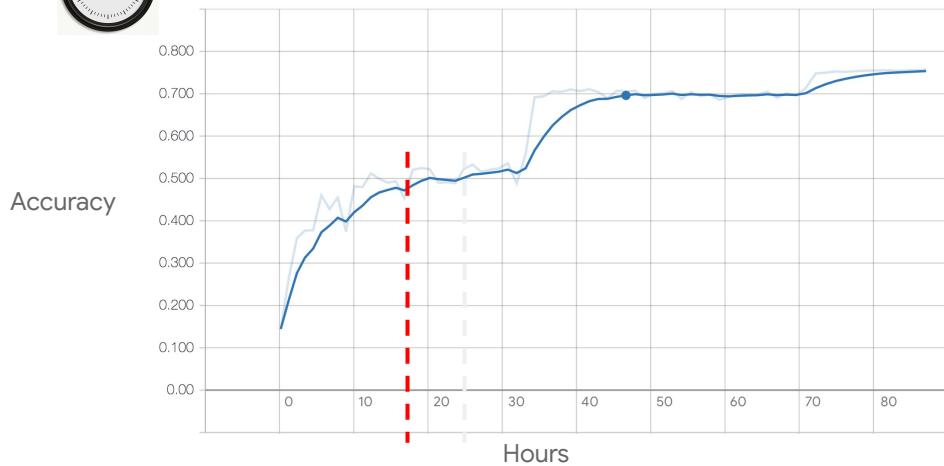
Model Training can take a long time



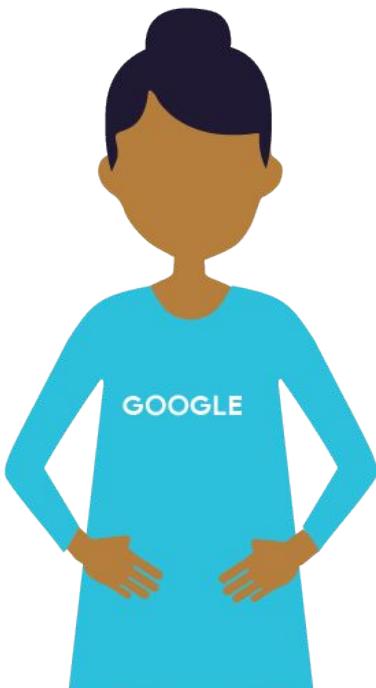
then your training has to finish within 24 hours. Realistically, you will need to time to deploy, to A/B test, etc.



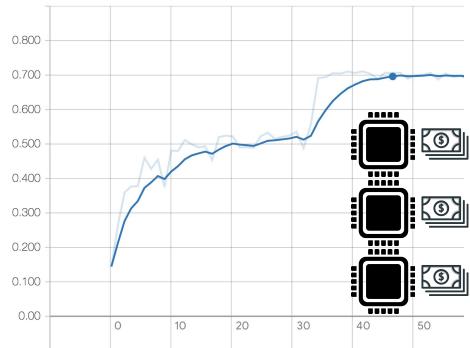
Model Training can take a long time



So, your actual budget might be only 18 hours.



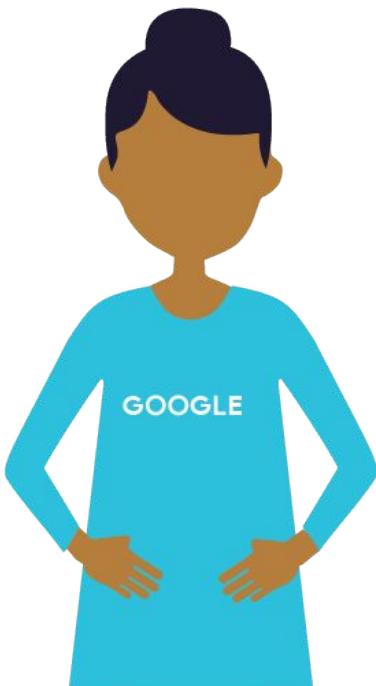
Analyze Benefit of Model vs Running Cost



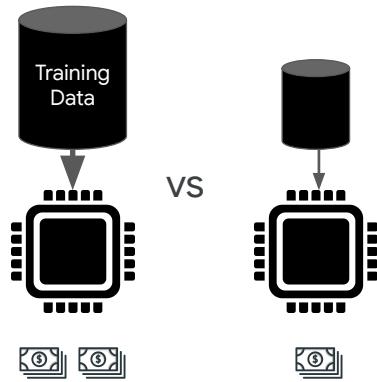
How much are you willing to spend on model training in terms of computing costs? This, too, is a business decision. You don't want to train for 18 hours every day if the incremental benefit is not sufficient.

Image cc0: CPU cost:

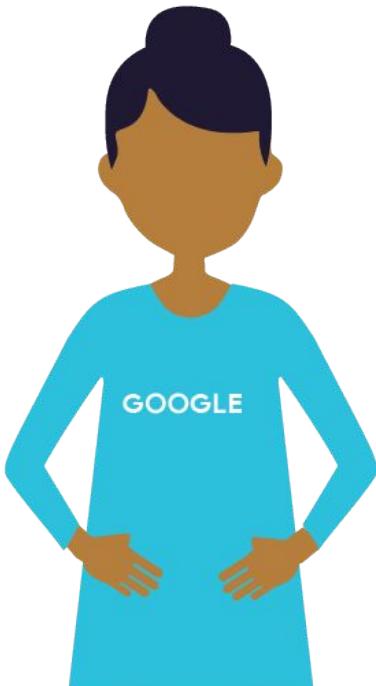
<https://pixabay.com/en/cpu-processor-computer-technology-2488091/>



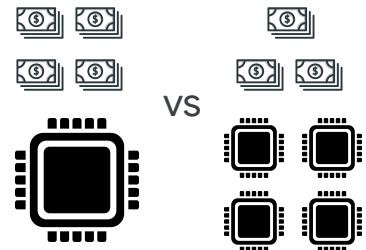
Optimize training
dataset size



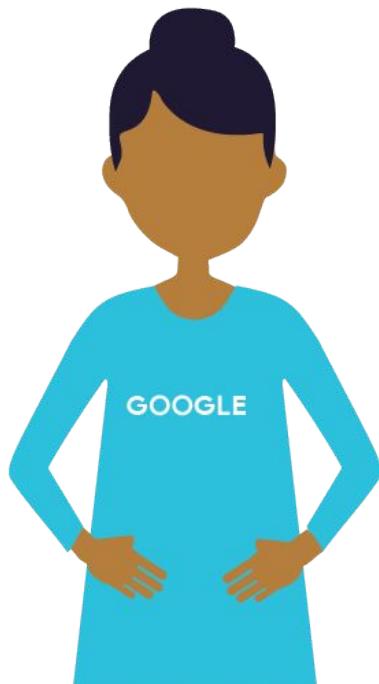
Scale is another aspect of your budget. Models differ in terms of how computationally expensive they are. Even keeping to the same model, you have a choice of how much data you are going to train on -- generally, the more data, the more accurate the model, but there are diminishing returns to larger and larger data sizes. So, your time and cost budget might dictate the data set size.



Choosing optimized infrastructure



Similarly, you often have a choice between training on a single, more expensive machine or multiple, cheaper machines. But to take advantage of this, you may have to write your code somewhat differently. That is another aspect of scale.



Use earlier model
checkpoints



Also, you have the choice of starting from an earlier model checkpoint, and training for just a few steps. Typically, this will converge faster than training from scratch each time. This compromise might allow you to reach the desired accuracy faster and cheaper.

<https://pixabay.com/en/mountaineering-rock-climbing-99093/>

<https://pixabay.com/en/mountains-distant-setting-infinite-378363/>

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	

In addition, there are ways to tune performance to reduce the time, reduce the cost, or increase the scale.

In order to understand what these are, it helps to understand that model training performance will be bound by one of three things:

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output

- **input/output** -- how fast can you get data into the model in each training step?
- Cpu -- how fast can you compute the gradient in each training step?
- Memory -- how many weights can you hold in memory, so that you can do the matrix multiplications in-memory on the GPU or TPU?

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU

- input/output -- how fast can you get data into the model in each training step?
- **Cpu** -- how fast can you compute the gradient in each training step?
- Memory -- how many weights can you hold in memory, so that you can do the matrix multiplications in-memory on the GPU or TPU?

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory

- input/output -- how fast can you get data into the model in each training step?
- Cpu -- how fast can you compute the gradient in each training step?
- **Memory** -- how many weights can you hold in memory, so that you can do the matrix multiplications in-memory on the GPU or TPU?

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory
Commonly Occurs	Large inputs Input requires parsing Small models		

Your ML training will be I/O bound if the number of inputs is large, heterogeneous (requires parsing), or if the model is so small that the compute requirements are trivial. This also tends to be the case if the input data is on a storage system with low throughput.

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory
Commonly Occurs	Large inputs Input requires parsing Small models	Expensive computations Underpowered Hardware	

Your ML training will be CPU bound if the I/O is simple, but the model involves lots of expensive computations. You will also encounter this situation if you are running a model on underpowered hardware.

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory
Commonly Occurs	Large inputs Input requires parsing Small models	Expensive computations Underpowered Hardware	Large number of inputs Complex model

Your ML training might be memory-bound if the number of inputs is large or if the model is complex and has lots of free parameters. You will also face memory limitations if your accelerator doesn't have enough memory.

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory
Commonly Occurs	Large inputs Input requires parsing Small models	Expensive computations Underpowered Hardware	Large number of inputs Complex model
Take Action	Store efficiently Parallelize reads Consider batch size		

So, knowing what you are bound by, you can look at how to improve performance. If you are I/O bound, look at storing the data more efficiently, on a storage system with higher throughput, or parallelizing the reads. Although it is not ideal, you might consider reducing the batch size so that you are reading less data in each step.

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory
Commonly Occurs	Large inputs Input requires parsing Small models	Expensive computations Underpowered Hardware	Large number of inputs Complex model
Take Action	Store efficiently Parallelize reads Consider batch size	Train on faster accel. Upgrade processor Run on TPUs Simplify model	

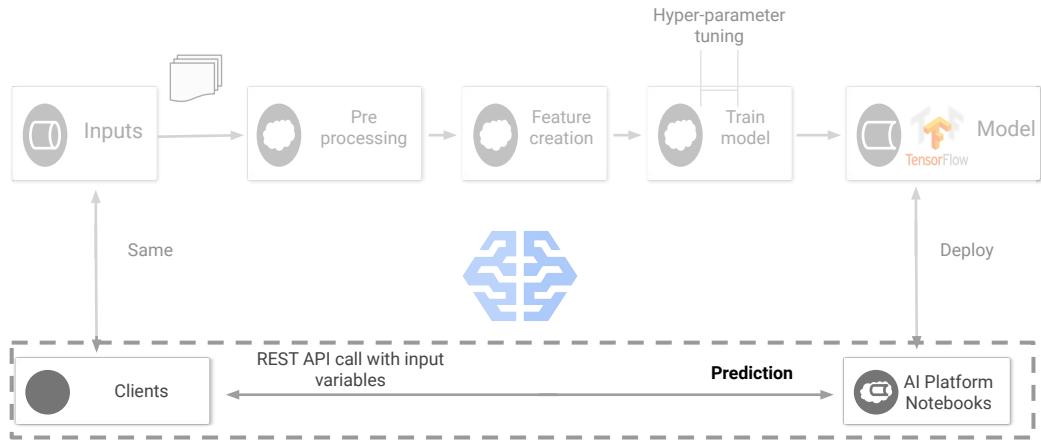
If you are CPU-bound, see if you can run the training on a faster accelerator. GPUs keep getting faster, so move to a newer generation processor. And on Google Cloud, you also have the option of running on TPUs. Even if it is not ideal, you might consider using a simpler model, a less computationally expensive activation function or simply just train for fewer steps.

Tuning Performance to reduce training time, reduce cost, and increase scale

Constraint	Input / Output	CPU	Memory
Commonly Occurs	Large inputs Input requires parsing Small models	Expensive computations Underpowered Hardware	Large number of inputs Complex model
Take Action	Store efficiently Parallelize reads Consider batch size	Train on faster accel. Upgrade processor Run on TPUs Simplify model	Add more memory Use fewer layers Reduce batch size

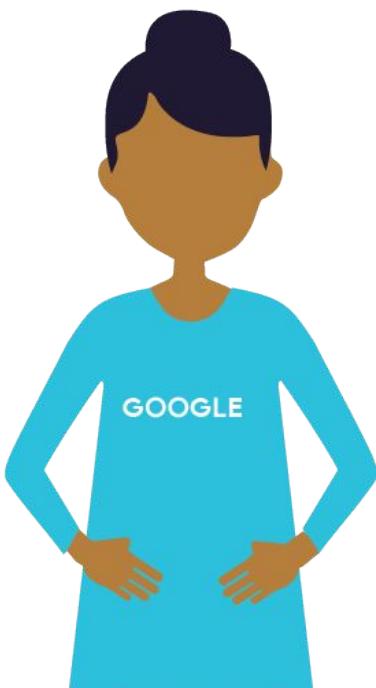
If you are memory-bound, see if you can add more memory to the individual workers. Again, not ideal, but you might consider using fewer layers in your model. Reducing the batch size can also help with memory-bound ML systems.

Performance must consider prediction-time, not just training

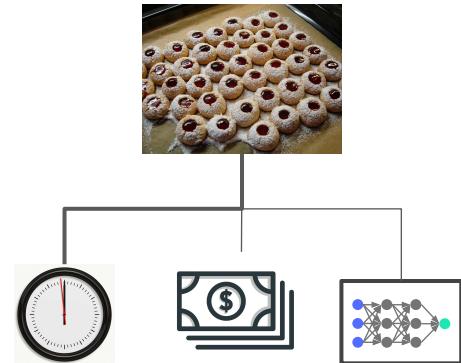


We have talked about time to *train*, but there is another aspect to performance.
Predictions.

During inference, you have performance considerations as well.



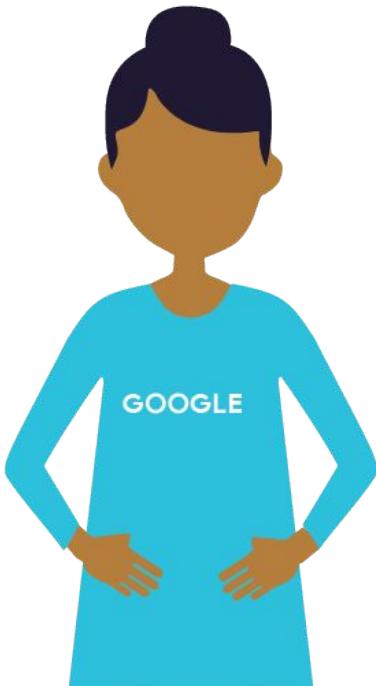
Optimizing your Batch Prediction



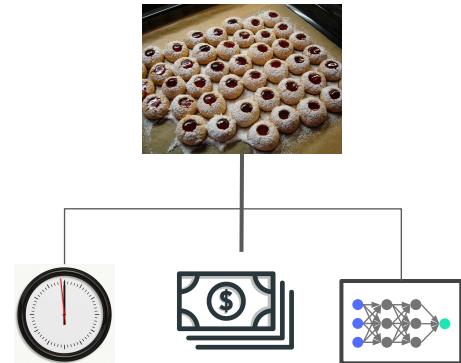
If you are doing batch prediction, the considerations are very similar to that of training. You are concerned with

- **Time:** How long does it take you to do all the predictions? This might be driven by a business need as well -- if you are doing product recommendations for the next day, you might want recommendations for the top 20% of users precomputed and available in about 5 hours, if it takes 18 hours to train ...
- Cost: What predictions you are doing, and how much you precompute is going to be driven by cost considerations
- Scale: do you have to do this all on one machine, or can you distribute it to multiple workers? What kind of hardware are on these workers? Do they have GPUs?

Batch: <https://pixabay.com/en/christmas-gingerbread-cookies-213645/>



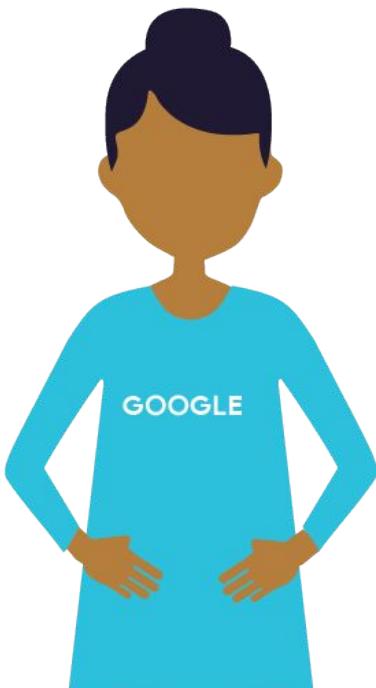
Optimizing your Batch Prediction



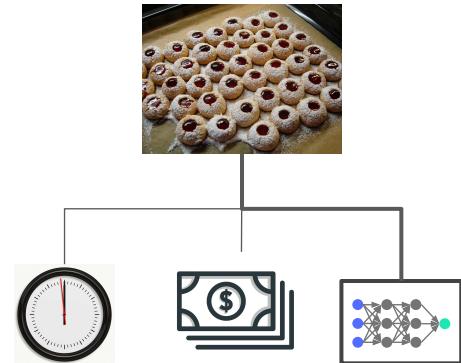
If you are doing batch prediction, the considerations are very similar to that of training. You are concerned with

- **Time:** How long does it take you to do all the predictions? This might be driven by a business need as well -- if you are doing product recommendations for the next day, you might want recommendations for the top 20% of users precomputed and available in about 5 hours, if it takes 18 hours to train ...
- **Cost:** What predictions you are doing, and how much you precompute is going to be driven by cost considerations
- **Scale:** do you have to do this all on one machine, or can you distribute it to multiple workers? What kind of hardware are on these workers? Do they have GPUs?

Batch: <https://pixabay.com/en/christmas-gingerbread-cookies-213645/>



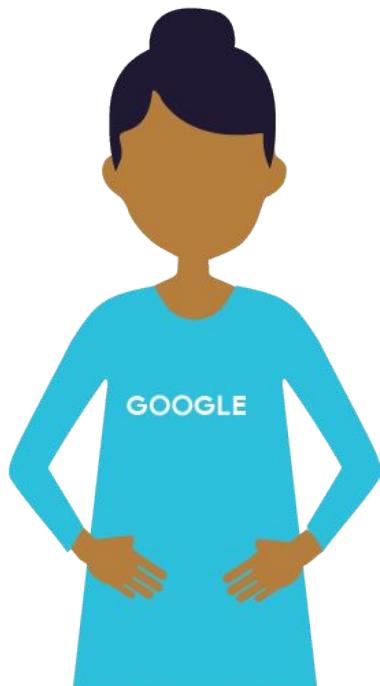
Optimizing your Batch Prediction



If you are doing batch prediction, the considerations are very similar to that of training. You are concerned with

- **Time:** How long does it take you to do all the predictions? This might be driven by a business need as well -- if you are doing product recommendations for the next day, you might want recommendations for the top 20% of users precomputed and available in about 5 hours, if it takes 18 hours to train ...
- **Cost:** What predictions you are doing, and how much you precompute is going to be driven by cost considerations
- **Scale:** do you have to do this all on one machine, or can you distribute it to multiple workers? What kind of hardware are on these workers? Do they have GPUs?

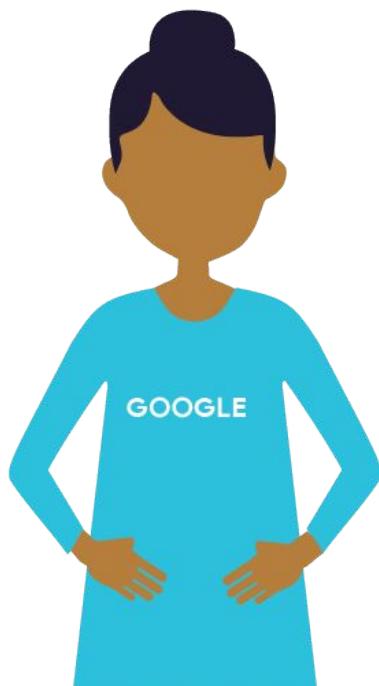
Batch: <https://pixabay.com/en/christmas-gingerbread-cookies-213645/>



Optimizing your Online Predictions



If you are doing online prediction, the performance considerations are quite different. This is because the end-user is waiting for the prediction. How is it different?

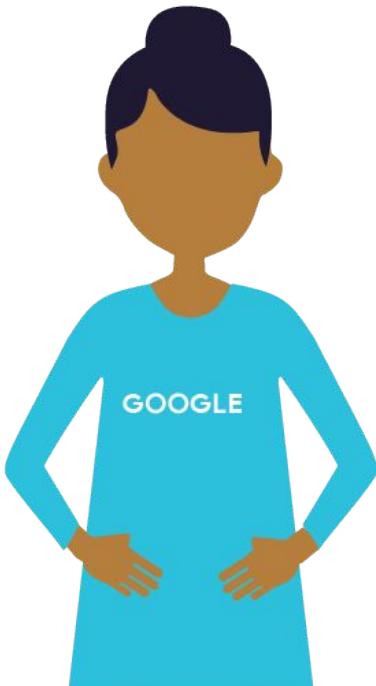


Optimizing your Online Predictions

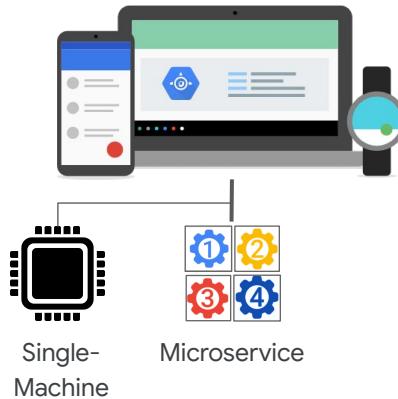


- You typically can not distribute the prediction graph; instead, you carry out the computation for one end-user on one machine
- However, you almost always scale out the predictions on to multiple workers. Essentially, each prediction is handled by a microservice, and you can replicate and scale out the predictions using Kubernetes or AppEngine -- Cloud AI Platform predictions are a higher-level abstraction, but they are equivalent to doing this.
- The performance consideration is not how many training steps can you carry out per minute, but how many queries you can handle per second. Queries Per Second or QPS. That's the performance target you need to hit.

When you design for high performance, you want consider training and prediction separately, especially if you will be doing online predictions.



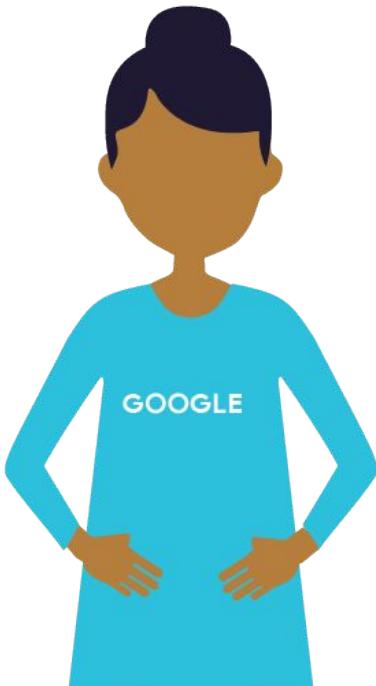
Optimizing your Online Predictions



[idea: triptych for single-machine, microservice, QPS]

- You typically can not distribute the prediction graph; instead, you carry out the computation for one end-user on one machine
- **However, you almost always scale out the predictions on to multiple workers. Essentially, each prediction is handled by a microservice, and you can replicate and scale out the predictions using Kubernetes or AppEngine -- Cloud ML Engine predictions are a higher-level abstraction, but they are equivalent to doing this.**
- The performance consideration is not how many training steps can you carry out per minute, but how many queries you can handle per second. Queries Per Second or QPS. That's the performance target you need to hit.

When you design for high performance, you want consider training and prediction separately, especially if you will be doing online predictions.



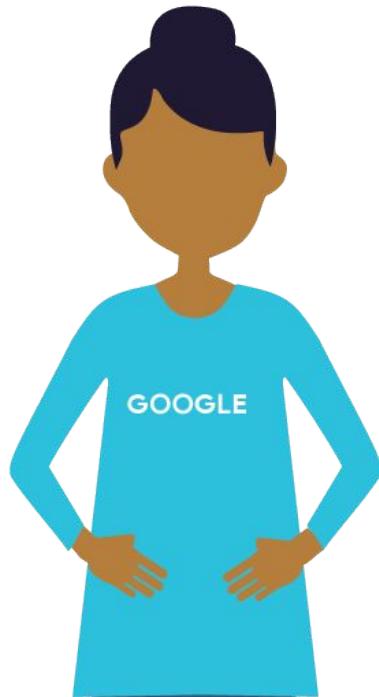
Optimizing your Online Predictions



[idea: triptych for single-machine, microservice, QPS]

- You typically can not distribute the prediction graph; instead, you carry out the computation for one end-user on one machine
- However, you almost always scale out the predictions on to multiple workers. Essentially, each prediction is handled by a microservice, and you can replicate and scale out the predictions using Kubernetes or AppEngine -- Cloud ML Engine predictions are a higher-level abstraction, but they are equivalent to doing this.
- **The performance consideration is not how many training steps can you carry out per minute, but how many queries you can handle per second. Queries Per Second or QPS. That's the performance target you need to hit.**

When you design for high performance, you want consider training and prediction separately, especially if you will be doing online predictions.

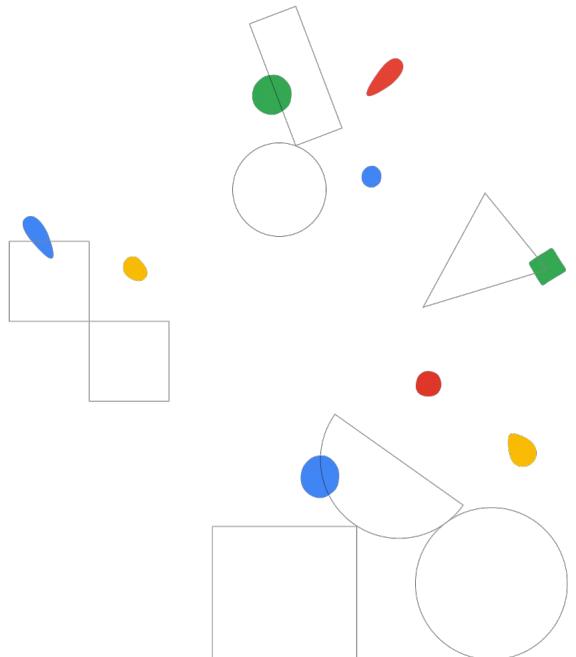


As I kind-of suggested in my line about precomputing batch predictions for the top 20% of users, and handling the rest of your users via online prediction, performance considerations will often involve striking the right balance. And ultimately, you will know the exact tradeoff (is it 20% or 10% or 25%, that you need to do) only after you build your system and measure things. However, unless you plan to be able to do both batch predictions and online predictions, you will be stuck with a solution that doesn't meet your needs.

The idea behind this module, and this course in general, is so that you are aware of the possibilities. Once you are aware that it can be done, it's not that difficult to accomplish -- the technical part is usually quite straightforward. Especially if you are using TensorFlow on a capable cloud platform.



Why distributed training is needed



Module 03

Designing high-performance ML systems



In this module, we'll explore how to run a distributed training job with TensorFlow.

Why distributed
training is needed

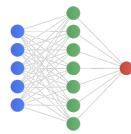


We'll begin with understanding why distributed training is needed.

Why distributed
training is needed

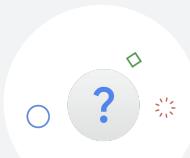


Distributed training
architectures

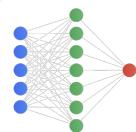


After that, we'll explore distributed training architectures.

Why distributed
training is needed



Distributed training
architectures



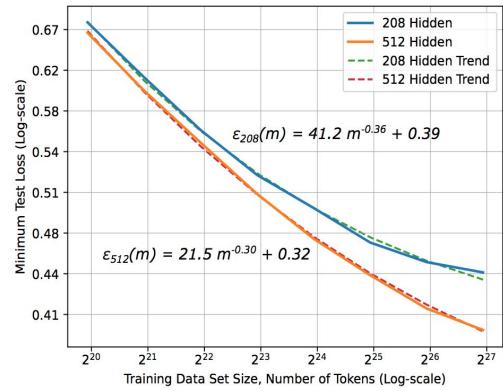
TensorFlow
distributed training
strategies



Then lastly, we'll provide an overview of TensorFlow distributed training strategies.



Deep learning works because datasets are large



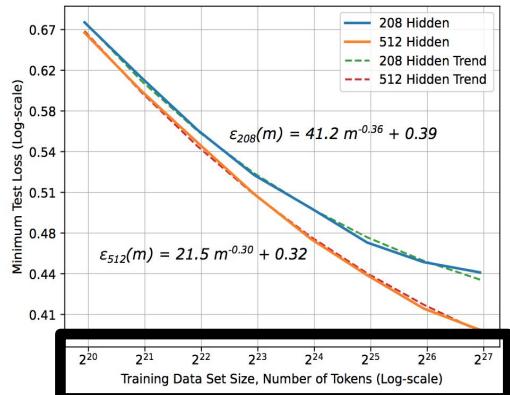
The unreasonable effectiveness of data
<https://static.googleusercontent.com/media/research.google.com/en/pubs/archive/35179.pdf>

Deep Learning scaling is predictable, empirically
<https://arxiv.org/abs/1712.00409>



Deep learning works because datasets are large.

Deep learning works because datasets are large



X-axis is logarithmic

For every doubling in the size of the data, the error rate falls linearly

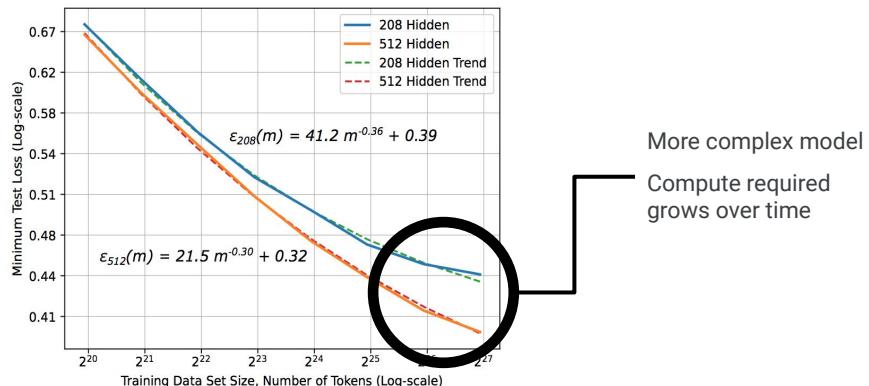
The unreasonable effectiveness of data
<https://static.googleusercontent.com/media/research.google.com/en/pubs/archive/35179.pdf>

Deep Learning scaling is predictable, empirically
<https://arxiv.org/abs/1712.00409>



Notice that the x-axis here is logarithmic. For every doubling in the size of the data, the error rate falls linearly.

Deep learning works because datasets are large



The unreasonable effectiveness of data
<https://static.googleusercontent.com/media/research.google.com/en/pubs/archive/35179.pdf>

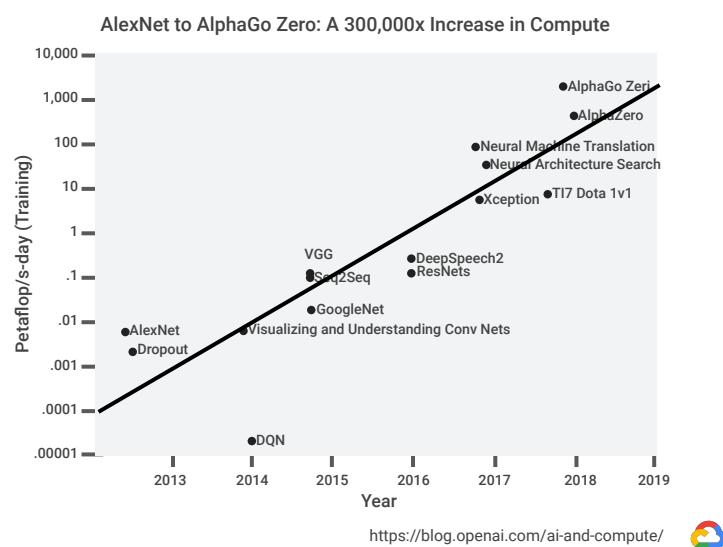
Deep Learning scaling is predictable, empirically
<https://arxiv.org/abs/1712.00409>



A more complex model also helps -- that is the jump from the blue line to the orange line -- but more data is even more helpful in this situation.

As a consequence of both of these trends, in terms of larger data sizes and more complex models, the compute required to build state-of-the-art models has grown over time. This growth is exponential as well.

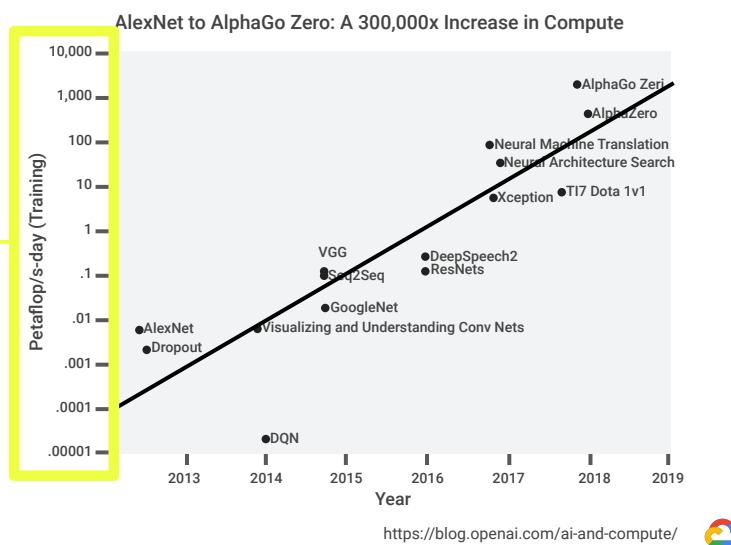
The compute required keeps increasing



Each y-axis tick

The compute required keeps increasing

10x increase in computational need

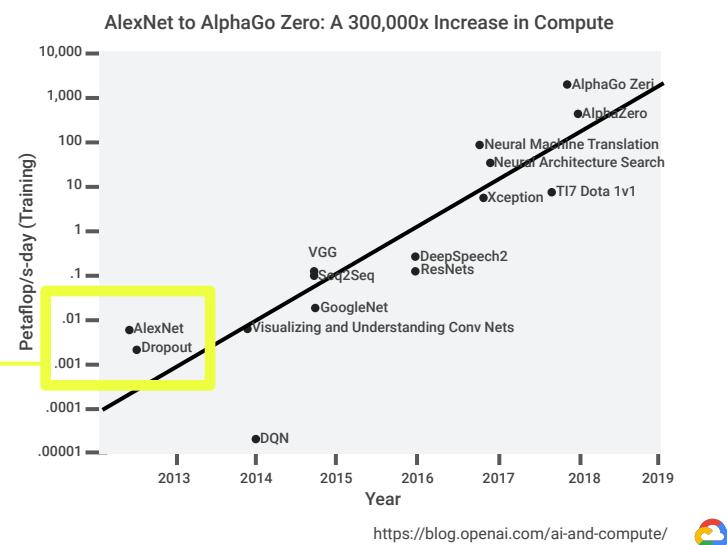


on this graph shows a 10x increase in computational need.

The compute required keeps increasing

AlexNet

<0.01 petaflops per second-day in compute per day for training

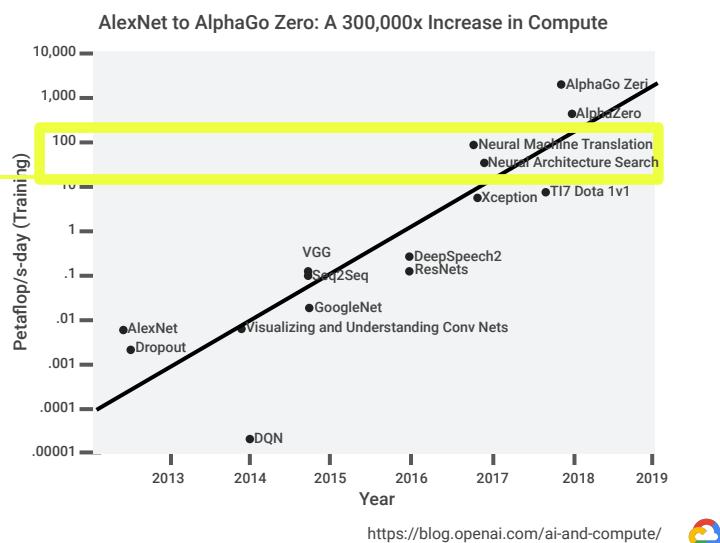


AlexNet, which started the deep learning revolution in 2013, required less than 0.01 petaflops per second-day in compute per day for training.

The compute required keeps increasing

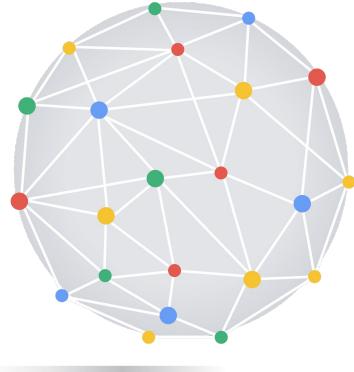
Neural
Architecture
Search

100 petaflops per second-day
(1000x more compute than you needed for AlexNet)

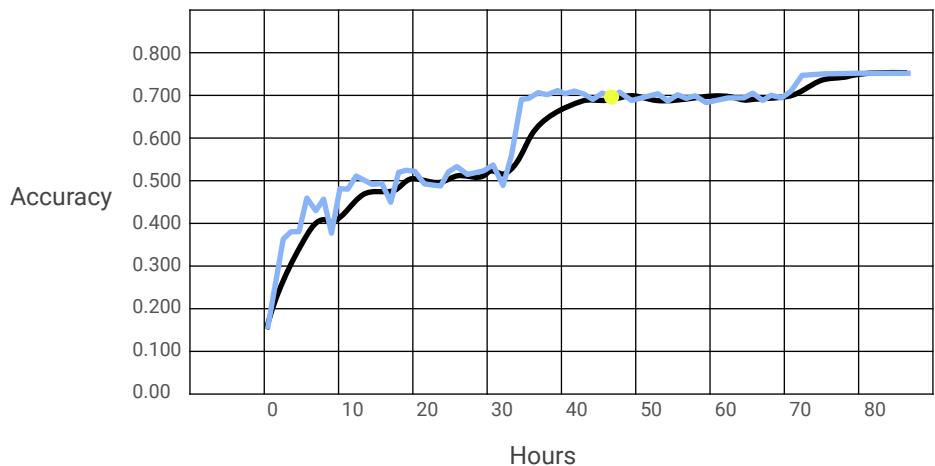


By the time you get to Neural Architecture Search, the learn-to-learn model published by Google in 2017, you need about 100 petaflops per second-day or 1000x more compute than you needed for AlexNet.

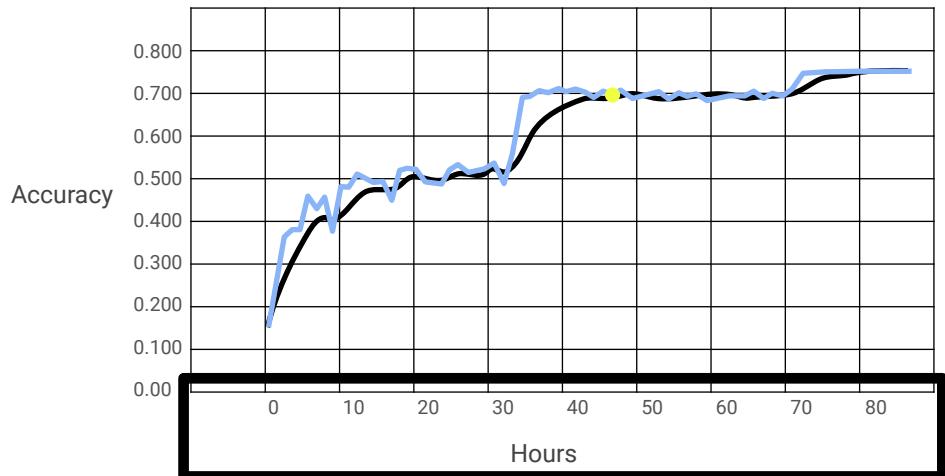
Distributed systems
are a necessity for
machine learning



This growth in algorithm complexity and data size means that, with complex models and large data volumes, distributed systems are pretty much a necessity when it comes to machine learning.

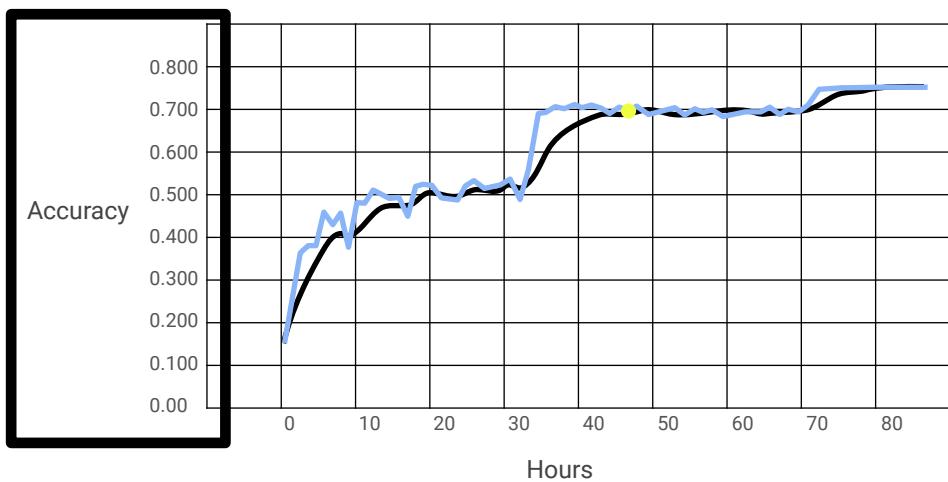


Training complex networks with large amounts of data can often take a long time.

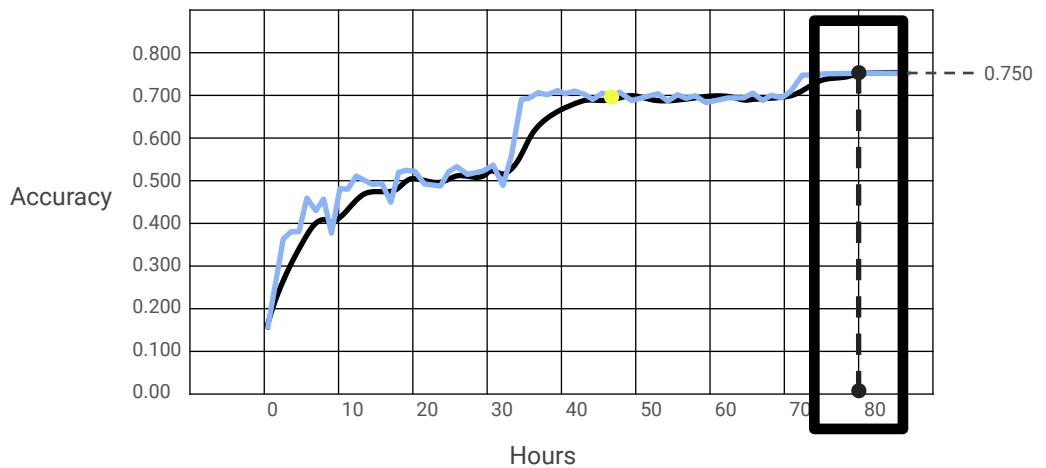


This graph shows training time on the x-axis





plotted against the accuracy of predictions on the y-axis, when training an image recognition model on a GPU.



As the dotted line shows, it took around 80 hours to reach 75% accuracy.



If your training takes a few minutes to a few hours, it will make you productive and happy, and you can try out different ideas fast.

 Minutes-hours

 1-4 days



If the training takes a few days, you could still deal with that by running a few ideas in parallel.

 Minutes-hours

 1-4 days

 1-4 weeks



If the training starts to take a week or more, your progress will slow down because you can't try out new ideas quickly.

 Minutes-hours

 1-4 days

 1-4 weeks

 > 1 month



And if it takes more than a month... Well that's probably not even worth thinking about!

And this is no exaggeration. Training deep neural networks such as ResNet50 can take up to a week on one GPU.

How can you make training faster?



Use a more powerful device



A natural question to ask is - how can you make training faster? You can use a more powerful device such as TPU or GPU (accelerator).

How can you make training faster?



Use a more powerful device



Optimize your input pipeline



You can optimize your input pipeline.

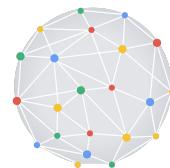
How can you make training faster?



Use a more powerful device



Optimize your input pipeline

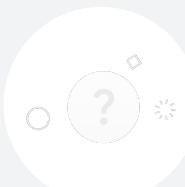


Try distributed training

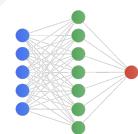


Or, you can try out distributed training.

Why distributed
training is needed



Distributed training
architectures



TensorFlow
distributed training
strategies



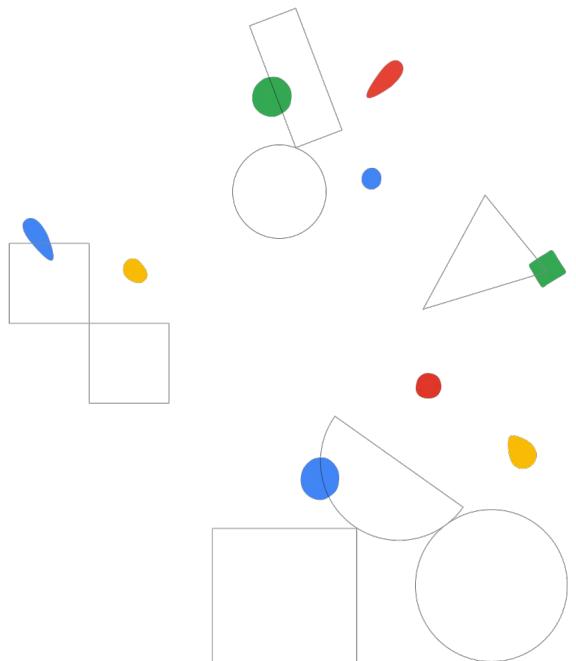
In the next video, we'll explore distributed training architectures.



Distributed training architectures

Module 03

Designing high-performance ML systems



Why distributed
training is needed



Distributed training
architectures

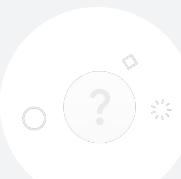


TensorFlow
distributed training
strategies

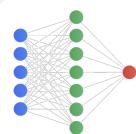


In the previous video we gave an overview of why distributed training is needed.

Why distributed training is needed



Distributed training architectures

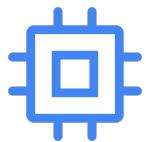


TensorFlow distributed training strategies



Let's now take a look at distributed training architectures.

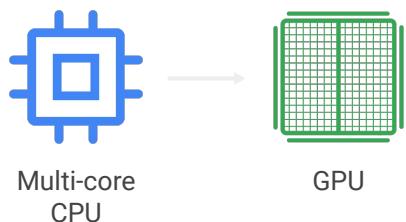
Before we get into the details of how to achieve this scaling in TensorFlow, let's step back and explore the high level concepts and architectures in distributed training.



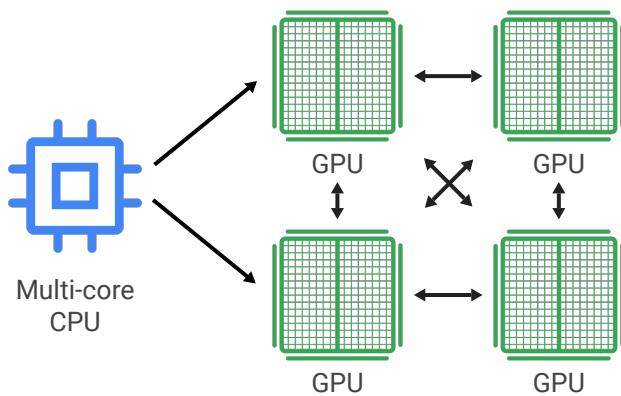
Multi-core
CPU



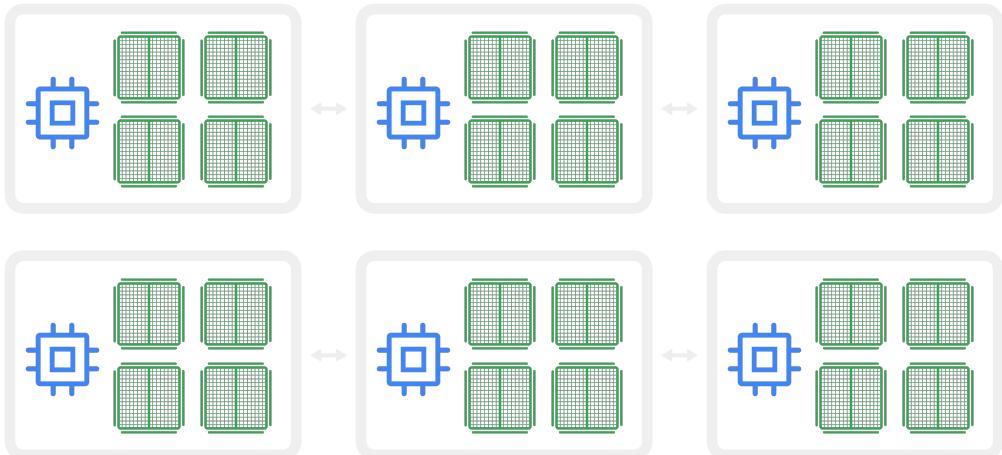
Let's say you start training on a machine with a multi-core CPU. TensorFlow automatically handles scaling on multiple cores.



You may speed up your training by adding an accelerator to your machine such as a GPU. Again, TensorFlow will use this accelerator to speed up model training with no extra work on your part.



But with distributed training, you can go further. You can go from using one machine with a single device, to a machine with multiple devices attached to it,



and finally to multiple machines, possibly with multiple devices each, connected over a network.

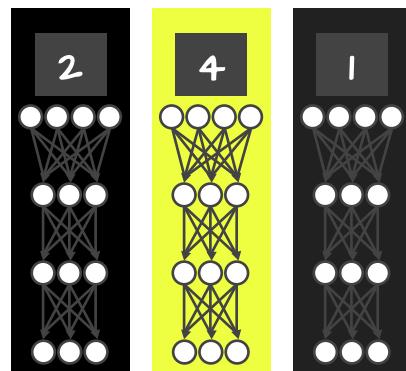
Eventually, with various approaches, you can scale up to hundreds of devices, and that is in fact what we do in several Google systems.

Simply stated, distributed training distributes training workloads across multiple mini-processors—or worker nodes.

These worker nodes work in parallel to accelerate the training process.

Types of distributed training architectures

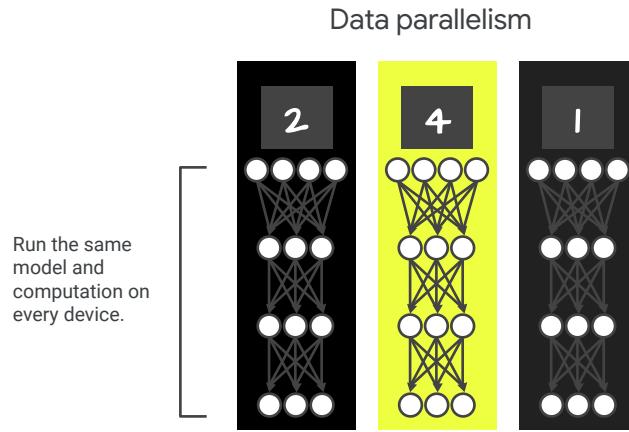
Data parallelism



Their parallelism can be achieved via two types of distributed training architecture. Let's explore both, starting with the most common, data parallelism.

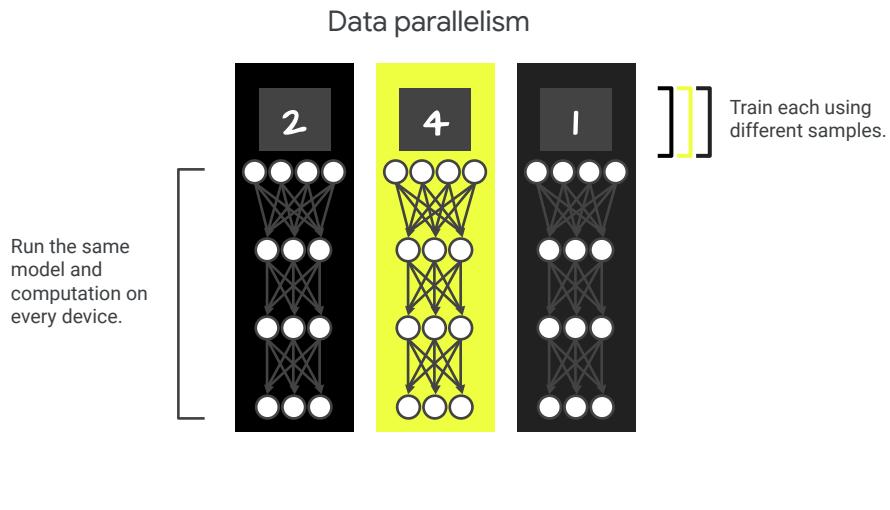
Data parallelism is model agnostic, making it the most widely used paradigm for parallelizing neural network training.

Types of distributed training architectures



In data parallelism, you run the same model and computation on every device,

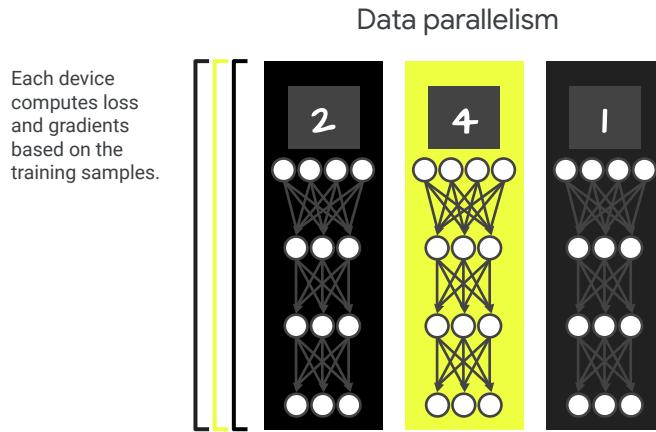
Types of distributed training architectures



but train each of them using different training samples.

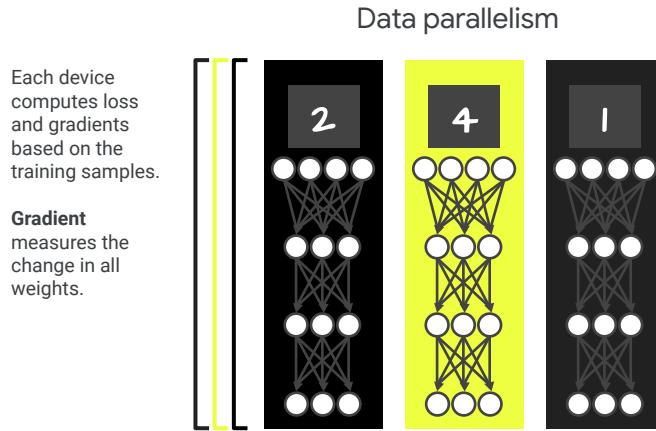


Types of distributed training architectures



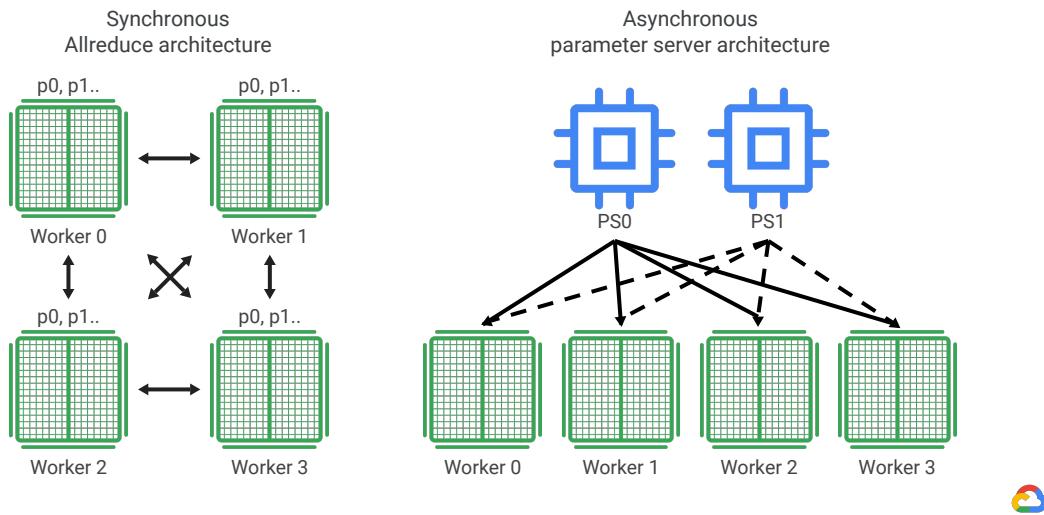
Each device computes loss and gradients based on the training samples it sees. Then we update the model's parameters using these gradients. The updated model is then used in the next round of computation.

Types of distributed training architectures



You'll recall that a gradient simply measures the change in all weights with regard to the change in error. You can also think of a gradient as the slope of a function. The higher the gradient, the steeper the slope and the faster a model can learn. But if the slope is zero, the model stops learning.

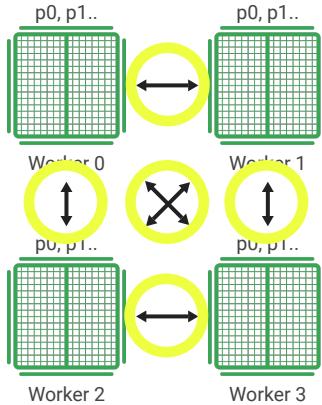
Data parallelism model approaches



There are currently two approaches used to update the model using gradients from various devices: **Synchronous** and **asynchronous**.

In synchronous training, all of the devices train their local model using different parts of data from a single, large mini-batch. They then communicate their locally calculated gradients, directly or indirectly, to all devices.

Synchronous Allreduce architecture

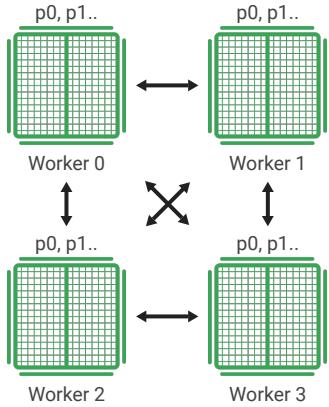


- Each worker device computes the forward and backward passes through the model on a different slice of the input data.



In this approach, each worker device computes the forward and backward passes through the model on a different slice of the input data.

Synchronous Allreduce architecture

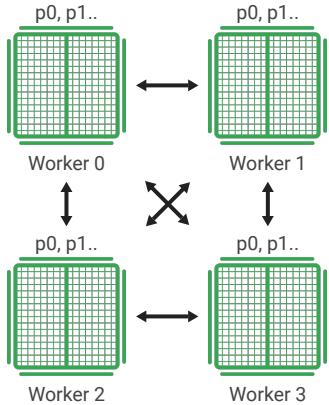


- Each worker device computes the forward and backward passes through the model on a different slice of the input data.
- Each training node exchanges the gradients via an AllReduce operation at the end of each training iteration.



The computed gradients from each of these slices are then aggregated across all of the devices and reduced, usually using an average, in a process known as AllReduce.

Synchronous Allreduce architecture



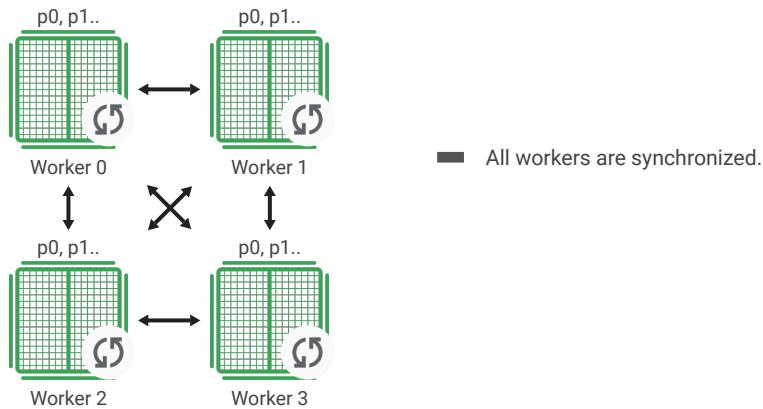
- Each worker device computes the forward and backward passes through the model on a different slice of the input data.
- Each training node exchanges the gradients via an AllReduce operation at the end of each training iteration.
- The optimizer performs the parameter updates with these reduced gradients, keeping the devices in sync.



The optimizer then performs the parameter updates with these reduced gradients thereby keeping the devices in sync.

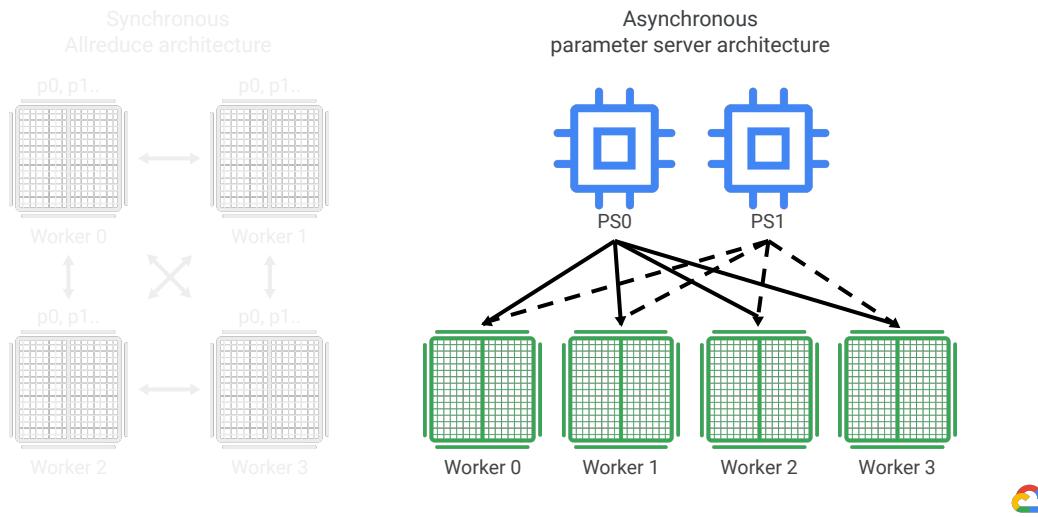
Because each worker cannot proceed to the next training step until all the other workers have finished the current step, this gradient calculation becomes the main overhead in distributed training for synchronous strategies.

Synchronous Allreduce architecture



Only after all devices have successfully computed and sent their gradients so that all workers are synchronized, is the model updated. The updated model is then sent to all nodes along with splits from the next mini-batch. That is, devices train on non-overlapping splits of the mini-batch.

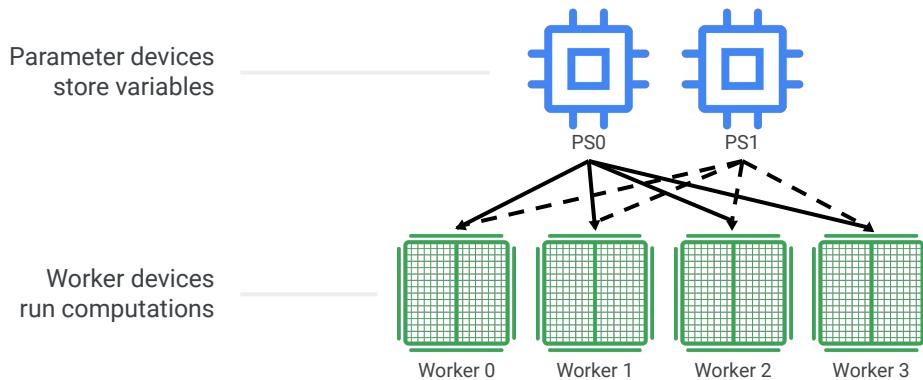
Asynchronous architecture



In asynchronous training, no device waits for updates to the model from any other device. The devices can run independently and share results as peers, or communicate through one or more central servers known as “parameter” servers.

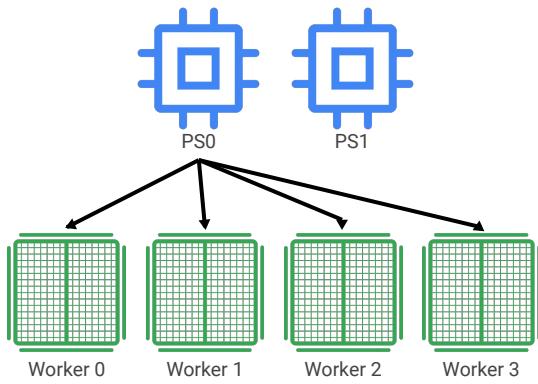
Thus, In an asynchronous parameter server architecture, some devices are designated to be parameter servers, and others as workers.

Asynchronous architecture



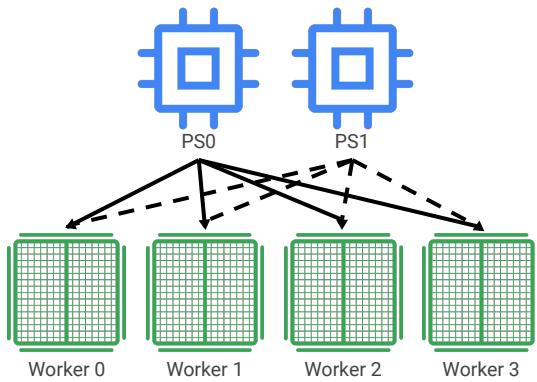
Devices used to store variables are parameter devices, whilst devices used to run computations are called worker devices.

Asynchronous architecture



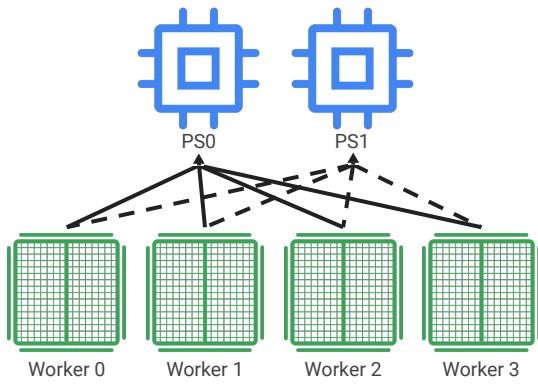
Each worker independently fetches

Asynchronous architecture



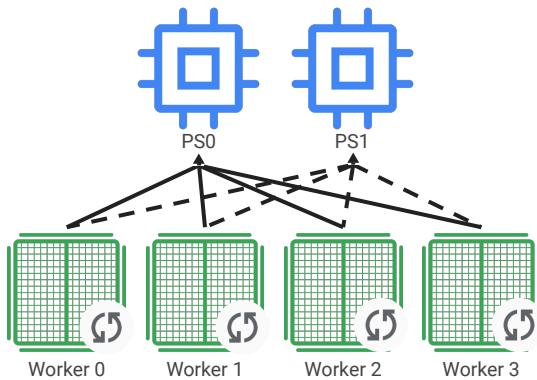
the latest parameters from the parameter servers and computes gradients based on a subset training samples.

Asynchronous architecture



It then sends the gradients back to the PS. Which then updates its copy of the parameters with those gradients.

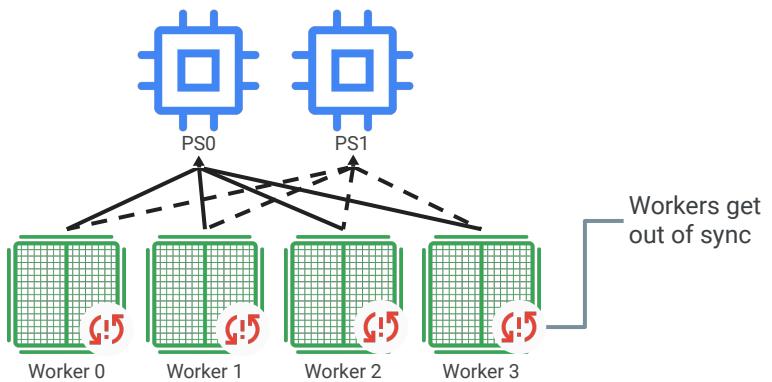
Asynchronous architecture



Each worker does this independently. This allows it to scale well to a large number of workers, where training workers might be preempted by higher priority production jobs, or a machine may go down for maintenance, or where there is asymmetry between the workers.

This does not hurt the scaling because workers are not waiting for each other.

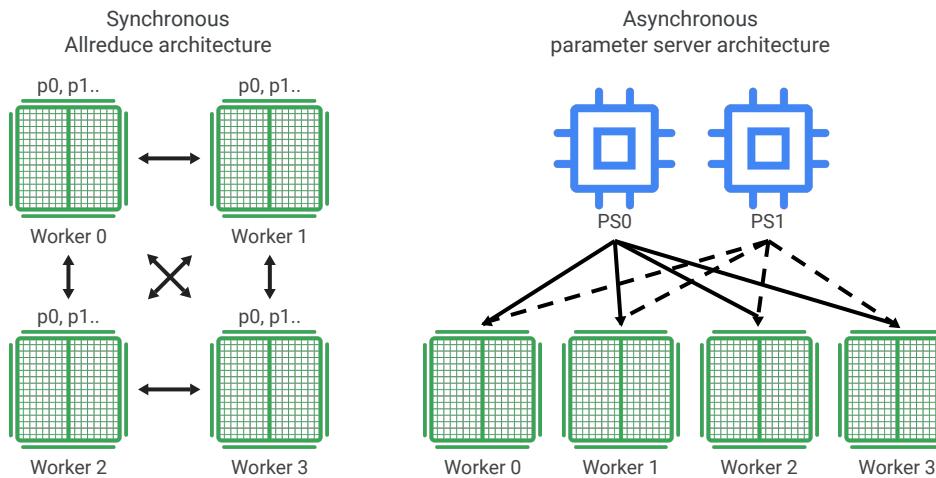
Asynchronous architecture



The downside of this approach, however, is that workers can get out of sync. They compute parameter updates based on stale values and this can delay convergence.



Which should you choose?



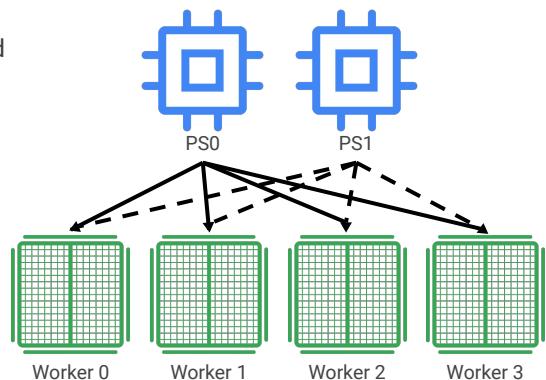
Given these two broad strategies, the asynchronous parameter server approach and the synchronous allreduce approach, which should you choose?

Well, there isn't one right answer, but here are some considerations.

Which should you choose?

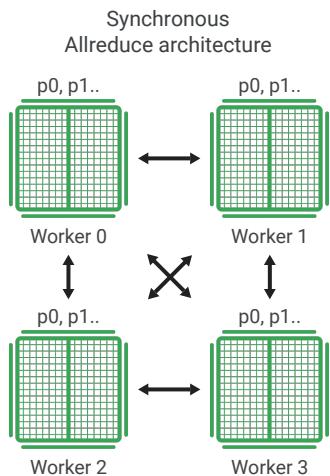
For models that use sparse data, contain fewer features, consume less memory, and can run just a cluster of CPUs.

Asynchronous parameter server architecture



The Asynchronous parameter server approach should be used for models that use sparse data (which contain fewer features, consume less memory, and can run just a cluster of CPUs).

Which should you choose?



Best for dense models, like BERT, or Bidirectional Encoder Representations from Transformers.

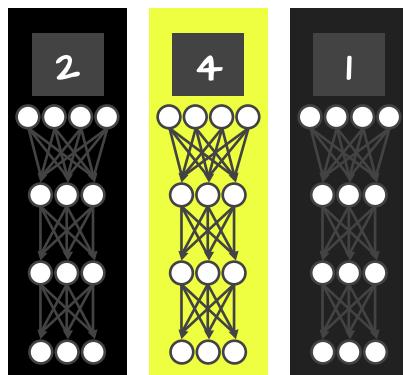


The Sync AllReduce approach should be considered for dense models which contain many features and thus consume more memory.

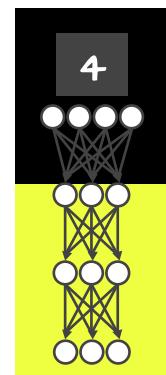
In this approach, all machines share the load of storing and maintaining the global parameters. This makes it the best option for dense models, like BERT (or Bidirectional Encoder Representations from Transformers).

Types of distributed training architectures

Data parallelism



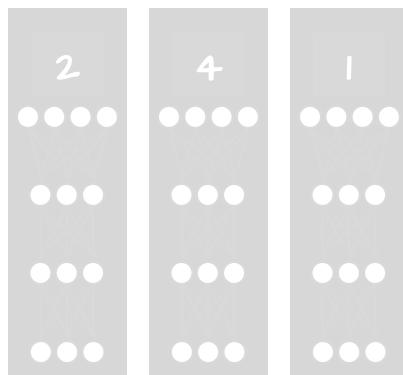
Model parallelism



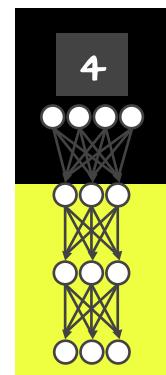
When a model is too big to fit on one device's memory, you divide it into smaller parts on multiple devices and then compute over the same training samples.

Types of distributed training architectures

Data parallelism



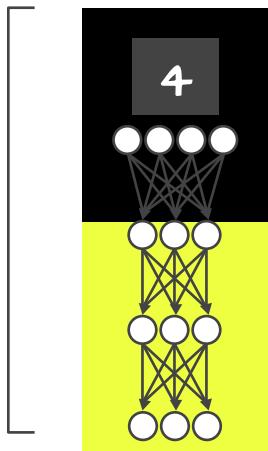
Model parallelism



This is called model parallelism.

Model parallelism

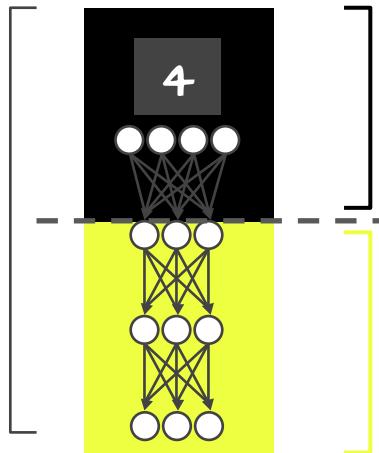
Each GPU has different parameters, and computation, of different parts of a model.



In this approach, each GPU has different parameters, and computation, of different parts of a model. In other words, multiple GPUs do not need to synchronize the values of the parameters.

Model parallelism

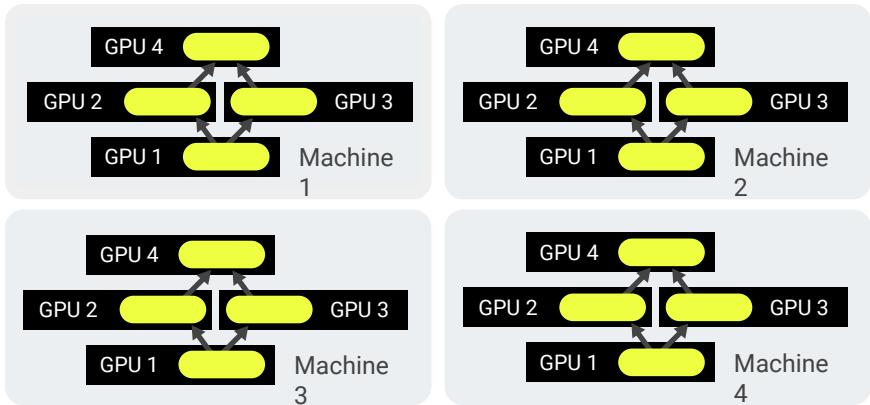
Needs special care
when assigning
different layers to
different GPUs.



Model parallelism needs special care when assigning different layers to different GPUs, which is more complicated than data parallelism.

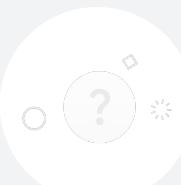
The gradients obtained from each model on each GPU are accumulated after a backward process, and the parameters are synchronized and updated.

Hybrid approach: Model and data parallelism

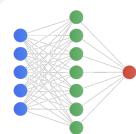


However, a hybrid of the data and model parallelism approaches is sometimes used together in the same architecture.

Why distributed
training is needed



Distributed training
architectures

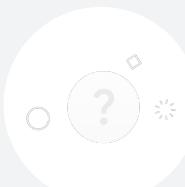


TensorFlow
distributed training
strategies



Now that you've been introduced to some of the different distributed training architectures,

Why distributed
training is needed



Distributed training
architectures



TensorFlow
distributed training
strategies



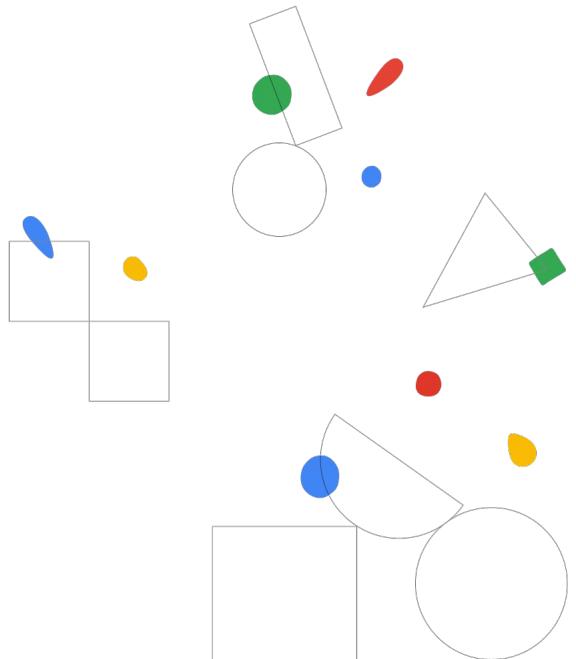
in the next video we'll take a look at four TensorFlow distributed training strategies.



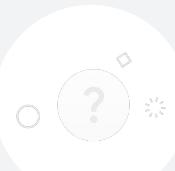
TensorFlow distributed training strategies

Module 03

Designing high-performance ML systems



Why distributed training is needed



Distributed training architectures



TensorFlow distributed training strategies



Distributed training is particularly useful for very large datasets, because it becomes very difficult, and often unrealistic to perform model training on only a single hardware accelerator, such as a GPU.

TensorFlow's distributed strategies make it easier to seamlessly scale up heavy training workloads across multiple hardware accelerators – be it GPUs or even TPUs.

But in doing so, you may face challenges

| How will you distribute the data across the different devices?



For example:

How will you distribute the model parameters across the different devices?

| How will you distribute the data across the different devices?

| How will you accumulate the gradients during backpropagation?



How will you accumulate the gradients during backpropagation?

| How will you distribute the data across the different devices?

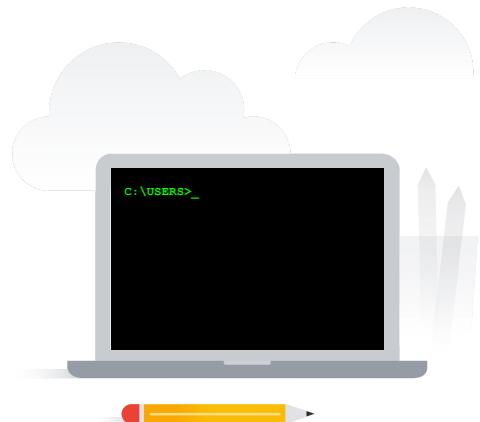
| How will you accumulate the gradients during backpropagation?

| How will the model parameters be updated?



And how will the model parameters be updated?

tf.distribute.Strategy API



tf.distribute.Strategy can help with these, and other, potential challenges. It is a TensorFlow API to distribute training across multiple GPUs, multiple machines or TPUs.

And there are four TensorFlow distributed training strategies. The list includes:

TensorFlow
distributed training
strategies



Mirrored strategy



the mirrored strategy

TensorFlow
distributed training
strategies



MirroredStrategy

MultiWorkerMirroredStrategy



the multi-worker mirrored strategy

TensorFlow
distributed training
strategies



MirroredStrategy

MultiWorkerMirroredStrategy

TPUStrategy



the TPU strategy

TensorFlow
distributed training
strategies



MirroredStrategy

MultiWorkerMirroredStrategy

TPUStrategy

ParameterServerStrategy



and finally, the parameter server strategy

We'll cover each strategy in more depth in the videos that follow.

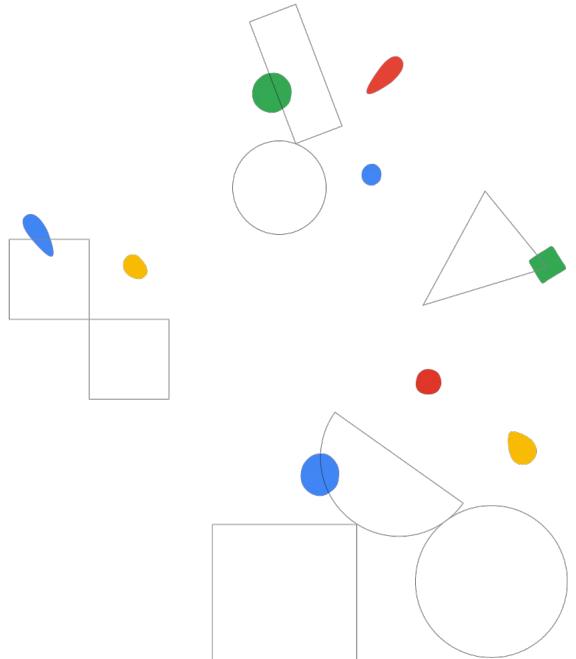
	Synchronous or Asynchronous	No. of nodes	No. of accelerators per node
MirroredStrategy	Synchronous	One	Many
TPUStrategy	Synchronous	One	Many
MultiWorkerMirroredStrategy	Synchronous	Many	Many
ParameterServerStrategy	Asynchronous	Many	Many



MirroredStrategy

Module 03

Designing high-performance ML systems



TensorFlow
distributed training
strategies



MirroredStrategy

MultiWorkerMirroredStrategy

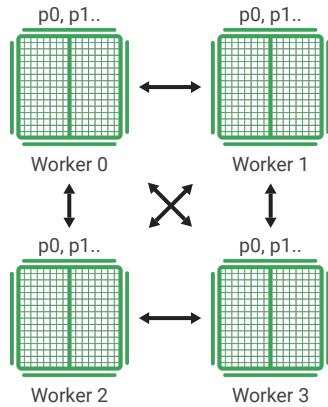
TPUStrategy

ParameterServerStrategy



The **mirrored strategy** is the simplest way to get started with distributed training.

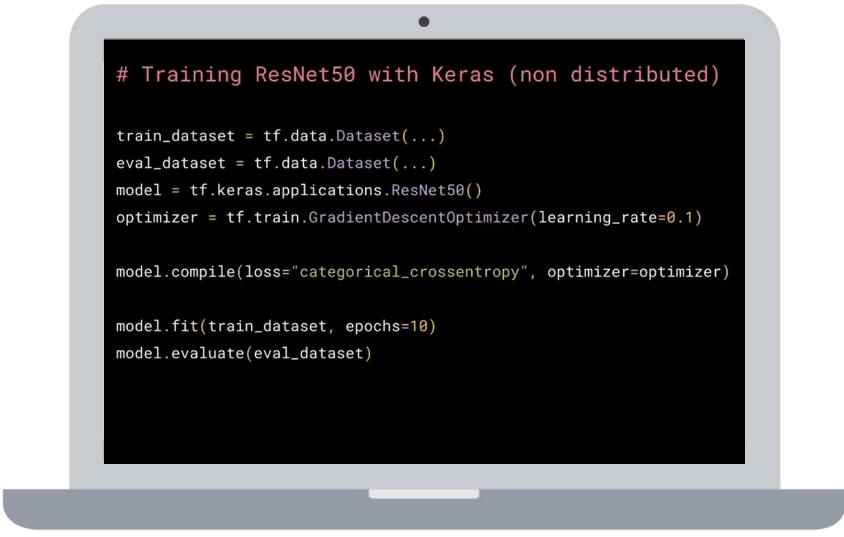
MirroredStrategy



It is a single machine with multiple GPU devices that creates one replica per GPU device.

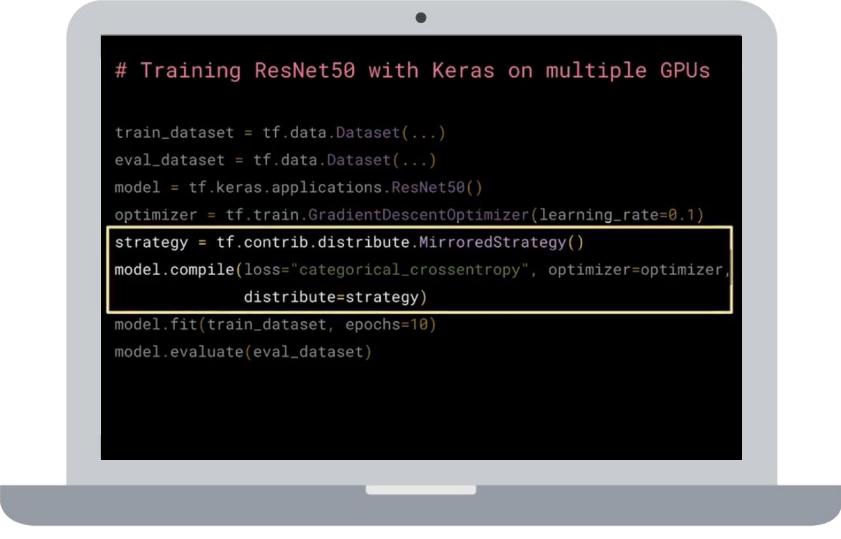
During training, one mini-batch is split into N parts, where N equals the number of GPUs, and each part feeds to one GPU device.

For this setup, the TensorFlow mirrored strategy manages the coordination of data distribution and gradient updates across all of the GPUs.



Here is an example of code showing a Non-distributed strategy.

Here we are training a Residual Network50 (or RESNET 50) with Keras. The code is standard -- we have our datasets, an optimizer, a model.compile, model.fit, and model.evaluation.



Now, to add a distribution strategy, we simply need to add the code here to show the mirrored strategy with multiple GPUs.

And here, you can see how easy it is to add code for a Distributed - Mirrored Strategy - single machines using multiple GPUs

Keras ResNet model with the functional API



Download the Cassava dataset from TensorFlow

```
data, info = tfds.load(name='cassava', as_supervised=True, with_info=True)
NUM_CLASSES = info.features['label'].num_classes
```



Add a preprocess_data function to scale the images

```
def preprocess_data(image, label):
    image = tf.image.resize(image, (300,300))
    return tf.cast(image, tf.float32) / 255., label
```



Let's look at an image classification example where a Keras ResNet model with the functional API is defined.

First, download the Cassava dataset from TensorFlow Datasets.

Then, add a preprocess_data function to scale the images.

Keras ResNet model with the functional API



Define the model

```
def create_model():
    base_model = tf.keras.applications.ResNet50(weights='imagenet',
        include_top=False)
    x = base_model.output
    x = tf.keras.layers.GlobalAveragePooling2D()(x)
    x = tf.keras.layers.Dense(1016, activation='relu')(x)
    predictions = tf.keras.layers.Dense(NUM_CLASSES, activation='softmax')(x)
    model = tf.keras.Model(inputs=base_model.input, outputs=predictions)
    return model
```



Then, define the model.

Keras ResNet model with the functional API



Create the strategy object

```
strategy = tf.distribute.MirroredStrategy()
```



Let's create the strategy object using `tf.distribute MirroredStrategy`.

Keras ResNet model with the functional API



Create your model variables within the strategy scope

```
with strategy.scope():
    model = create_model()
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=tf.keras.optimizers.Adam(0.0001),
        metrics=['accuracy'])
```



Next, let's create the model with variables within the strategy scope. These variables include the model, sparse_categorical_crossentropy for loss,, a Keras optimizer, and metrics variables to compute accuracy.

Keras ResNet model with the functional API



Change the batch size

```
batch_size = 64 * strategy.num_replicas_in_sync
```



The last change you will want to make is to the batch size.

When you carry out distributed training with the `tf.distribute.Strategy` API and `tf.data`, the batch size now refers to the global batch size.

Keras ResNet model with the functional API



Change the batch size

```
batch_size : 64 * strategy.num_replicas_in_sync
```



In other words, if you pass a batch size of 64, and you have two GPUs, then each machine will process 32 examples per step. In this case, 64 is known as the global batch size, and 32 as the per replica batch size.

To make the most out of your GPUs, you will want to scale the batch size by the number of replicas, which is two in this case because there is one replica on each GPU.

Keras ResNet model with the functional API



Map, shuffle, and prefetch the data

```
train_data = data['train'].map(preprocess_data)
train_data = train_data.shuffle(1000)
train_data = train_data.batch(batch_size)
train_data = train_data.prefetch(tf.data.experimental.AUTOTUNE)
```



From there, map, shuffle, and prefetch the data.

Keras ResNet model with the functional API



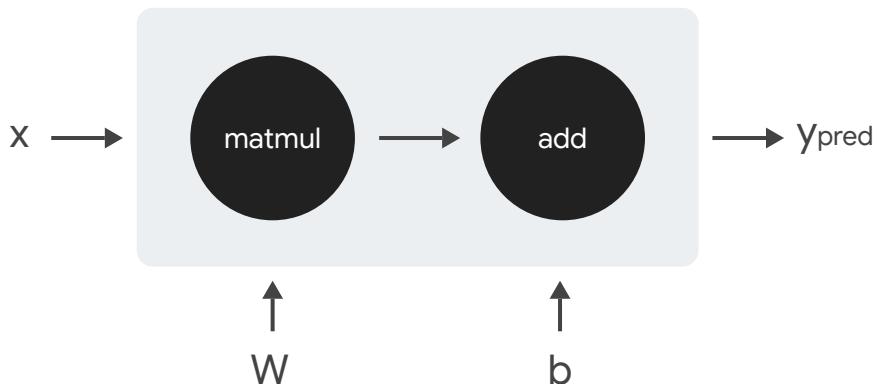
Call model.fit

```
model.fit(train_data, epochs = 5)
```



You then call model fit on the training data. Here we are going to run five passes of the entire training dataset.

DAG without MirroredStrategy

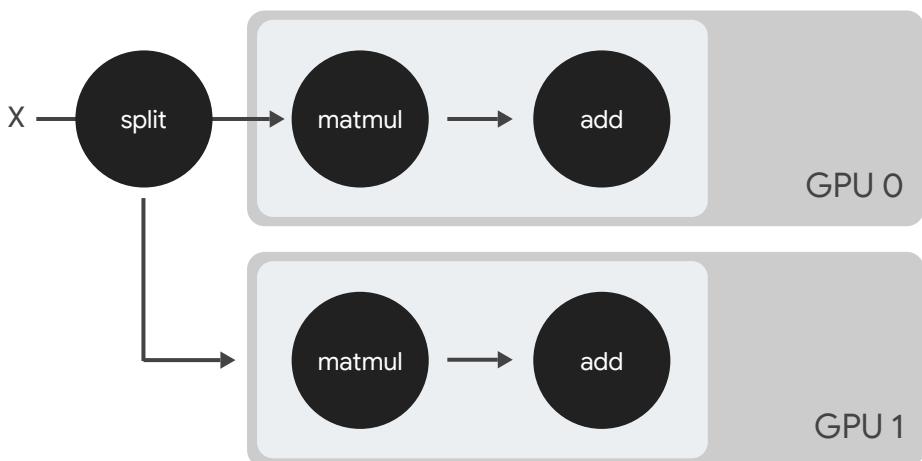


Let's take a brief look at what actually happens when we call `model.fit` before adding a strategy.

For simplicity, imagine you have a simple linear model instead of the ResNet50 architecture. In TensorFlow, you can think of this simple model in terms of its computational graph (or Directed Acyclic Graph - or DAG).

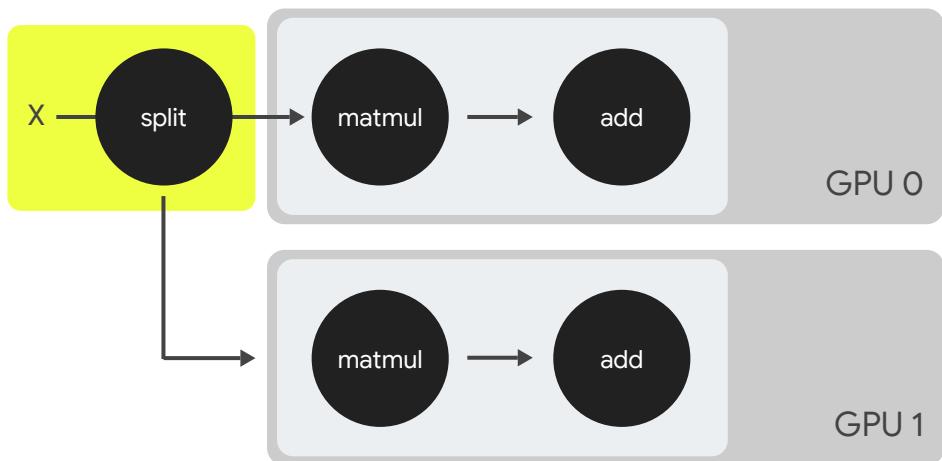
Here, the `matmul` op takes in the X and W tensors, which are the training batch and weights respectively. The resulting tensor is then passed to the `add` op with the tensor b , which is the model's bias terms. The result of this op is y_{pred} , which is the model's predictions.

Data parallelism with two GPUs



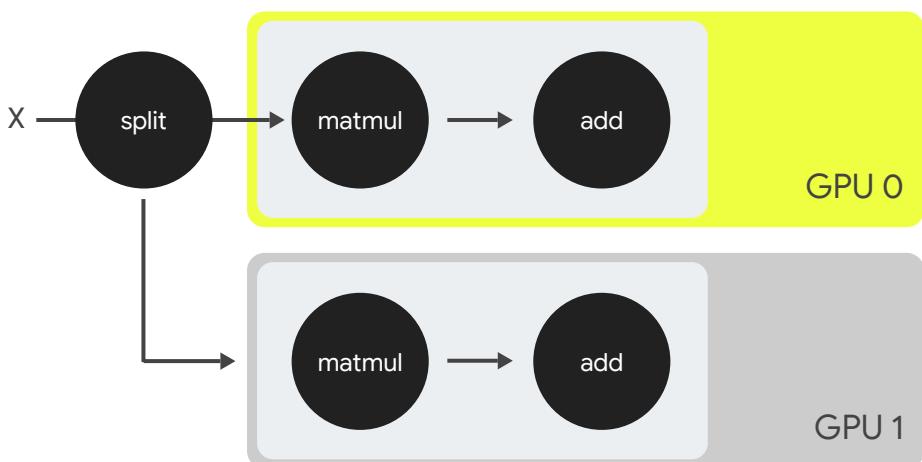
Here is an example of data parallelism with two GPUs.

Data parallelism with two GPUs



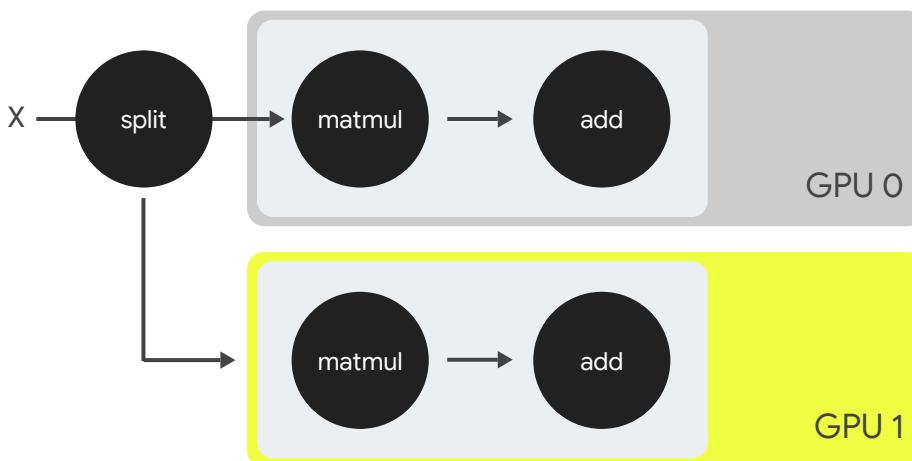
The input batch X is split in half,

Data parallelism with two GPUs



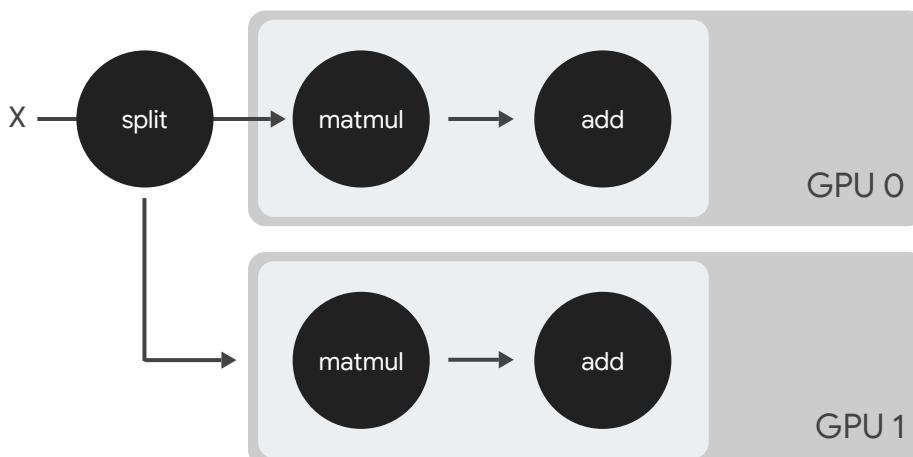
and one slice is sent to GPU 0

Data parallelism with two GPUs



and the other to GPU 1.

Data parallelism with two GPUs



In this case, each GPU calculates the same ops but on different slices of the data.

Optimize TensorFlow performance using the Profiler

tensorflow.org/guide/profiler



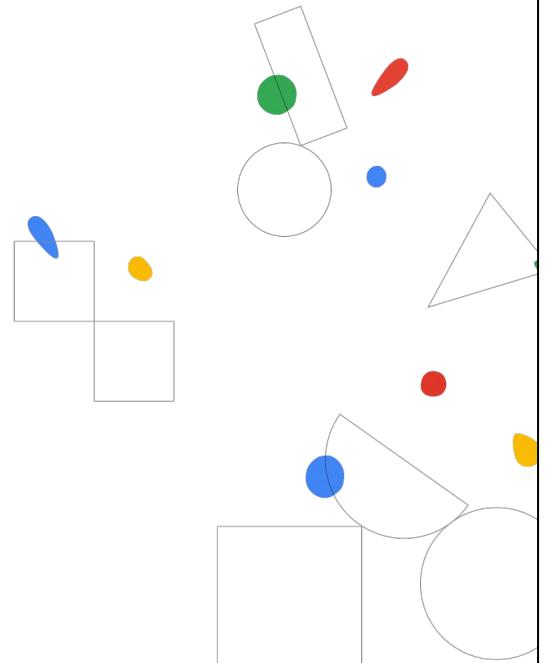
For more information on making the most of your GPUs, please refer to the guide titled, “Optimize TensorFlow GPU Performance with the TensorFlow Profiler,” found at tensorflow.org/guide/gpu_performance_analysis.



MultiWorkerMirroredStrategy

Module 03

Designing high-performance ML systems



TensorFlow
distributed training
strategies



MirroredStrategy

MultiWorkerMirroredStrategy

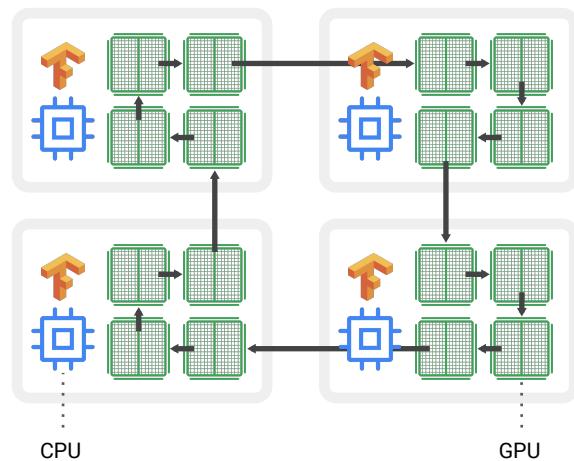
TPUStrategy

ParameterServerStrategy



The Multi Worker Mirrored Strategy is very similar to the MirroredStrategy. It implements synchronous distributed training across multiple workers, each with potentially multiple GPUs. Similar to MirroredStrategy, it creates copies of all variables in the model on each device across all workers.

Add machines to scale training



If you've mastered single host training and are looking to scale training even further, then adding multiple machines to your cluster can help you get an even greater performance boost. You can make use of a cluster of machines that are CPU only, or that each have one or more GPUs.

MultiWorker MirroredStrategy

```
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```



Like its single-worker counterpart, MirroredStrategy, MultiWorkerMirroredStrategy is a synchronous data parallelism strategy that can be used with only a few code changes.

Multi-worker mirrored strategy

```
Which machines are part of the cluster?  
  
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```



However, unlike MirroredStrategy, for a multi-worker setup TensorFlow needs to know which machines are part of the cluster. In most cases, this is specified with the environment variable TF_CONFIG.

MultiWorker MirroredStrategy

```
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```

Contains a dictionary with the internal IPs and ports of all the machines.



In this simple TF_CONFIG example, the “cluster” key contains a dictionary with the internal IPs and ports of all the machines.

MultiWorker MirroredStrategy

```
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```

Physical machines on which the replicated computation is executed.



In MultiWorkerMirroredStrategy, all machines are designated as workers, which are the physical machines on which the replicated computation is executed.

MultiWorker MirroredStrategy

```
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```

One worker that takes
on some extra work.



In addition to each machine being a worker, there needs to be one worker that takes on some extra work such as saving checkpoints and writing summary files to TensorBoard. This machine is known as the chief (or by its deprecated name master).

MultiWorker MirroredStrategy

With AI Platform Training, the TF_CONFIG environment variable is set on each machine in your cluster.

```
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```



Conveniently, when using AI Platform Training,

MultiWorker MirroredStrategy

With AI Platform Training, the TF_CONFIG environment variable is set on each machine in your cluster.

```
os.environ["TF_CONFIG"] = json.dumps({  
    "cluster": {  
        "chief": ["host1:port"],  
        "worker": ["host2:port", "host3:port"],  
    },  
    "task": {"type": "worker", "index": 1}  
})
```



the TF_CONFIG environment variable is set on each machine in your cluster so there's no need to worry about this set up!

`tf.distribute`

1. Create a strategy object.

```
strategy =  
tf.distribute.MultiWorkerMirroredStrategy()
```



As with any strategy in the `tf.distribute` module, step one is to create a strategy object.

tf.distribute

```
1. Create a      strategy =
    strategy object.      tf.distribute.MultiWorkerMirroredStrategy()

with strategy.scope():
    model = create_model()
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=tf.keras.optimizers.Adam(0.0001),
        metrics=['accuracy'])

2. Wrap the creation
   of the model
   parameters within the
   Scope of the strategy.
```

Step two is to wrap the creation of the model parameters within the scope of the strategy. This is crucial because it tells MirroredStrategy which variables to mirror across the GPU devices.



tf.distribute

1. Create a strategy object.

```
strategy =  
tf.distribute.MultiWorkerMirroredStrategy()
```

2. Wrap the creation of the model parameters within the Scope of the strategy.

```
with strategy.scope():  
    model = create_model()  
    model.compile(  
        loss='sparse_categorical_crossentropy',  
        optimizer=tf.keras.optimizers.Adam(0.0001),  
        metrics=['accuracy'])
```

3. Scale the batch size by the number of replicas in the cluster.

```
per_replica_batch_size = 64  
global_batch_size =  
per_replica_batch_size *  
strategy.num_replicas_in_sync
```



And the third and final step is to scale the batch size by the number of replicas in the cluster. This ensures that each replica processes the same number of examples on each step.

Since we've already covered training with MirroredStrategy, the previous steps should be familiar. The main difference when moving from synchronous data parallelism on one machine to many is that the gradients at the end of each step now need to be synchronized across all GPUs in a machine and across all machines in the cluster.

This additional step of synchronizing across the machines increases the overhead of distribution.

MultiWorkerMirroredStrategy



Large datasets that don't fit in a single file



When you are working with large datasets that don't fit in a single file, using a MultiWorkerMirroredStrategy

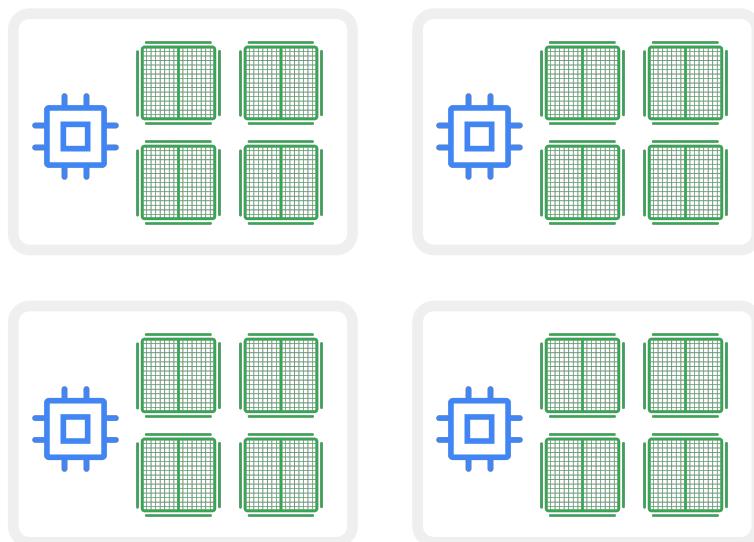
MultiWorkerMirroredStrategy



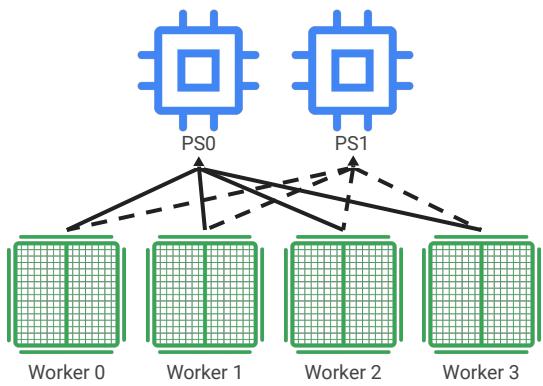
Large datasets that don't fit in a single file



is the best approach.



However, this option can lead to idle workers if the number of those multiple files is not divisible by the number of workers evenly, or if the length of some files are substantially longer than others.

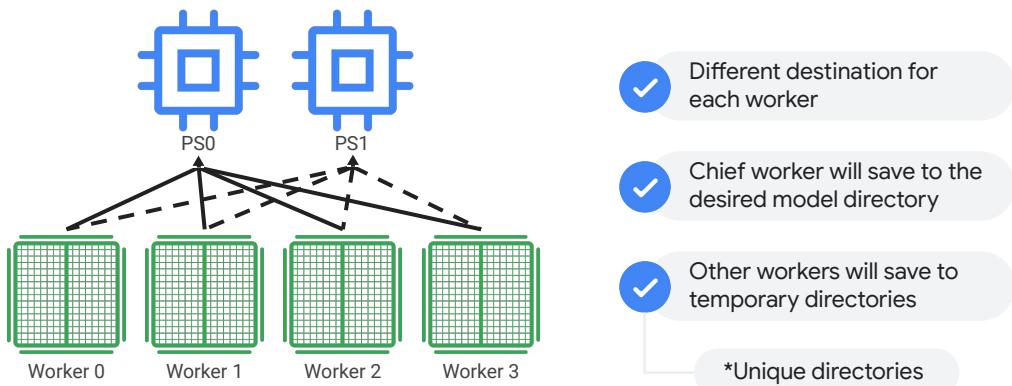


If the data is not stored in a single dataset, then TensorFlow's AutoShardPolicy will autoshard the elements across all the workers.



If the data is not stored in a single dataset, then TensorFlow will autoshard the elements across all the workers. This guards against the potential idle worker scenario, but the downside is that the entire dataset will be read on each worker.

Saving



Saving the model is slightly more complicated in the multi-worker case because there needs to be different destinations for each worker.

The chief worker will save to the desired model directory, while the other workers will save the model to temporary directories.

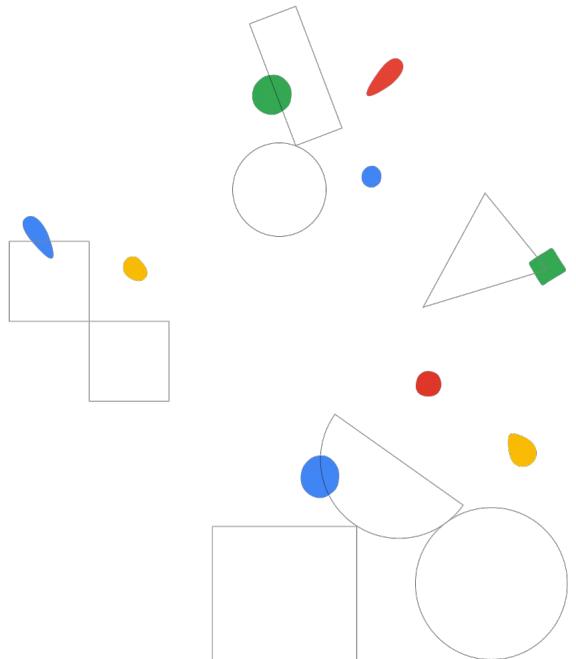
It's important that these temporary directories are unique in order to prevent multiple workers from writing to the same location. Saving can contain collective ops, so all workers must save and not just the chief.



TPUStrategy

Module 03

Designing high-performance ML systems



TensorFlow
distributed training
strategies



MirroredStrategy

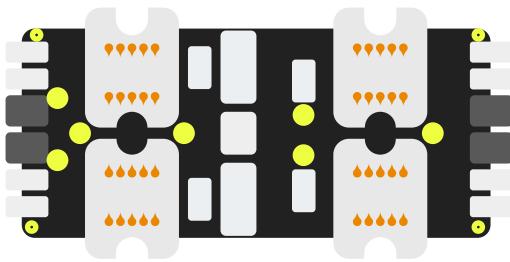
MultiWorkerMirroredStrategy

TPUStrategy

ParameterServerStrategy

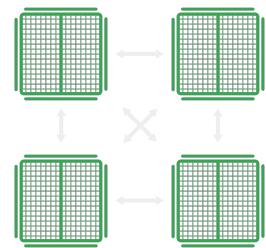


TPUStrategy



All-reduce across TPU cores

MirroredStrategy

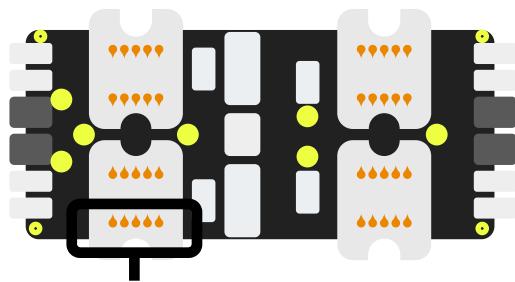


All-reduce across devices



The main difference, however, is that the TPU strategy will all-reduce across TPU cores, whereas the Mirrored Strategy will all-reduce across devices.

`tf.distribute.TPUStrategy`



`tf.distribute.TPUStrategy` lets you run your TensorFlow training on Tensor Processing Units (TPUs). TPUs are Google's specialized ASICs designed to dramatically accelerate machine learning workloads.

TPUs provide their own implementation of efficient all-reduce and other collective operations across multiple TPU cores, which are used in `TPUStrategy`.

Call the `tf.distribute.TPUStrategy()` method

```
strategy = tf.distribute.TPUStrategy()

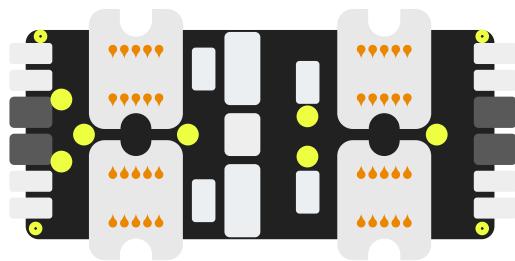
with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=tf.keras.optimizers.Adam(0.0001),
        metrics=['accuracy'])
```



Data considerations



Many models ported to the TPU end up with a data bottleneck

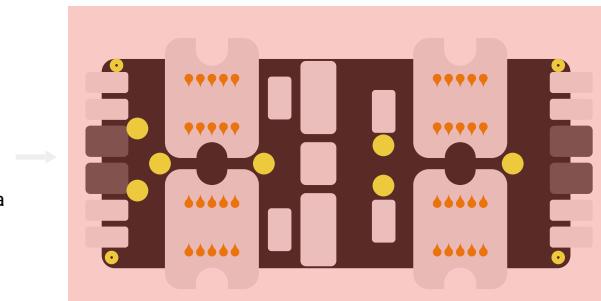


Because TPUs are very fast, many models ported to the TPU end up with a data bottleneck.

Data considerations



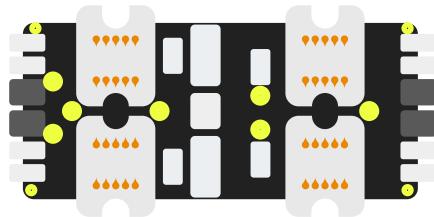
Many models ported to the TPU end up with a data bottleneck



The TPU is sitting idle, waiting for data for the most part of each training epoch.

Data considerations

- ✓ TPUs read training data from GCS
- ✓ GCS can sustain a large throughput
- ✓ Too few files: GCS will not have enough streams to get max throughput
- ✓ Too many files: Time will be wasted



TPUs read training data exclusively from Google Cloud Storage (GCS). And GCS can sustain a pretty large throughput if it is continuously streaming from multiple files in parallel.

Following best practices will optimize the throughput. With too few files, GCS will not have enough streams to get max throughput. With too many files, time will be wasted accessing each individual file.

```
model = tf.keras.Sequential(...)  
model.compile(loss='mse', optimizer='sgd')  
model.fit(dataset, epochs=2)  
model.evaluate(dataset)
```



Let's summarize the distribution strategies using code.

Our base scope is a Keras sequential model.

```
strategy = tf.distribute.MirroredStrategy()

with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(loss='mse', optimizer='sgd')
```



Now, to improve training, we can use the mirrored strategy.

```
strategy = tf.distribute.MultiWorkerMirroredStrategy()

with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(loss='mse', optimizer='sgd')
    model.fit(dataset, epochs=2)
    model.evaluate(dataset)
```



Or for faster training, the multi-worker mirrored strategy.

```
strategy = tf.distribute.TPUStrategy()

with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(loss='mse', optimizer='sgd')
    model.fit(dataset, epochs=2)
    model.evaluate(dataset)
```



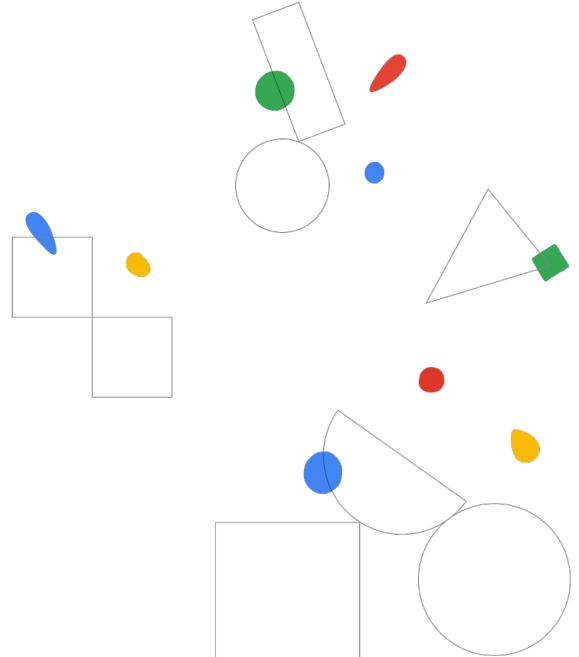
And for really fast training, the TPU strategy.



ParameterServerStrategy

Module 03

Designing high-performance ML systems



TensorFlow
distributed training
strategies



MirroredStrategy

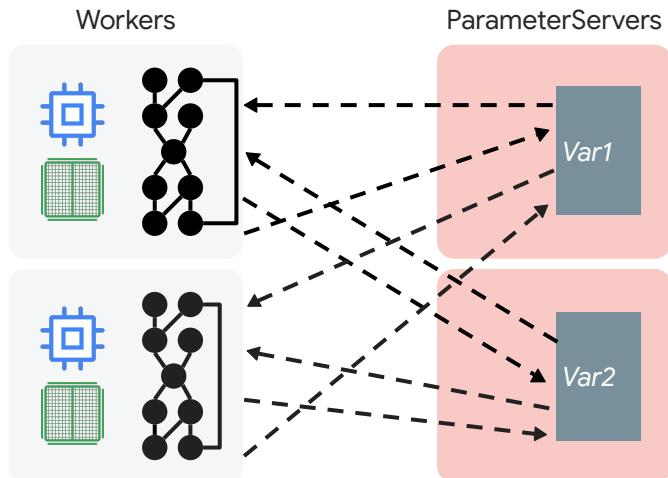
MultiWorkerMirroredStrategy

TPUStrategy

ParameterServerStrategy

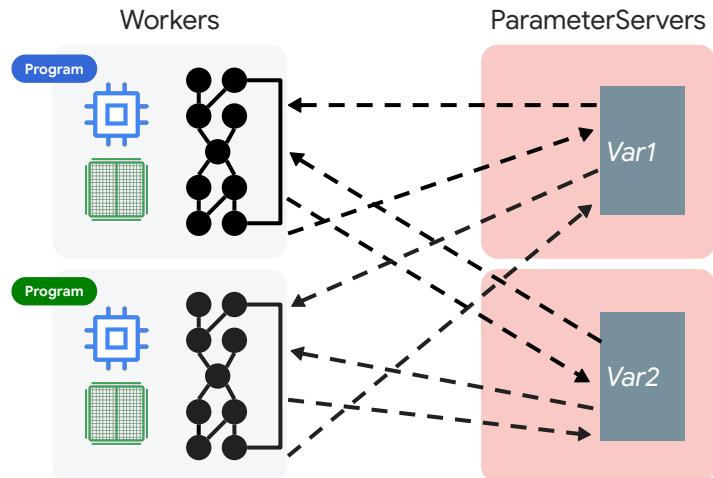


Earlier we explored the asynchronous parameter server architecture earlier. A parameter server training cluster consists of workers and parameter servers.

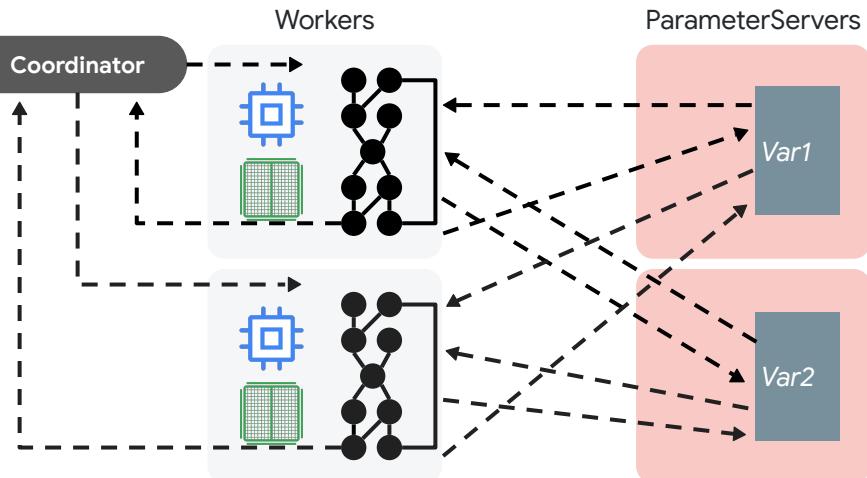


Earlier we explored the asynchronous parameter server architecture earlier. A parameter server training cluster consists of workers and parameter servers.

Variables are created on parameter servers and they are read and updated by workers in each step. By default, workers read and update these variables independently without synchronizing with each other.



If you used Parameter Server Strategy in TensorFlow 1, you might recall that each worker ran its own training program.



In TensorFlow 2, we introduce a central coordinator.

The coordinator is a special task type that creates resources, dispatches training tasks, writes checkpoints, and deals with task failures.

```
strategy = tf.distribute.experimental.ParameterServerStrategy(
    tf.distribute.cluster_resolver.TFConfigClusterResolver())

def dataset_fn(input_context):
    ...
    return dataset

dc = tf.keras.utils.experimental.DatasetCreator(dataset_fn)

with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(loss='mse', optimizer='sgd')

model.fit(dc, epochs=5, steps_per_epoch=20, callbacks=callbacks)
```



```
strategy = tf.distribute.experimental.ParameterServerStrategy(
    tf.distribute.cluster_resolver.TFConfigClusterResolver())

def dataset_fn(input_context):
    ...
    return dataset

dc = tf.keras.utils.experimental.DatasetCreator(dataset_fn)

with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(loss='mse', optimizer='sgd')

model.fit(dc, epochs=5, steps_per_epoch=20, callbacks=callbacks)
```



```
strategy = tf.distribute.experimental.ParameterServerStrategy(
    tf.distribute.cluster_resolver.TFConfigClusterResolver())

def dataset_fn(input_context):
    ...
    return dataset

dc = tf.keras.utils.experimental.DatasetCreator(dataset_fn)

with strategy.scope():
    model = tf.keras.Sequential(...)
    model.compile(loss='mse', optimizer='sgd')

model.fit(dc, epochs=5, steps_per_epoch=20, callbacks=callbacks)
```



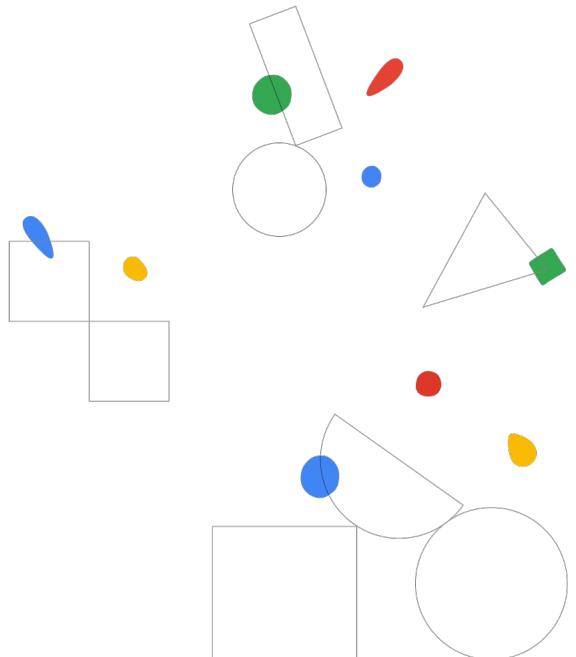


Distributed training with Keras

Lab

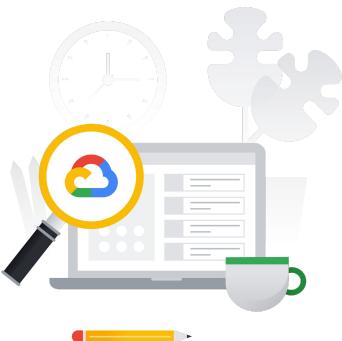
Module 03

Designing high-performance ML systems



This lab provides guidance on how to use distributed training with Keras.

Lab objectives

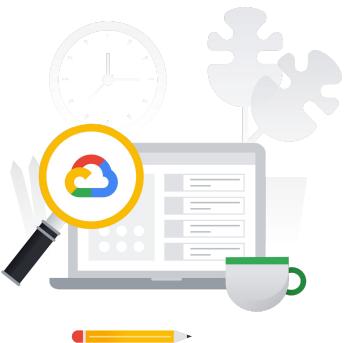


Define distribution strategy and set input pipelines.



You'll begin by defining a distribution strategy and setting an input pipeline,

Lab objectives



Define distribution strategy and set input pipelines.

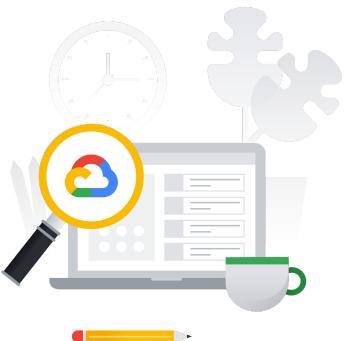


Create a Keras model.



then continue on to create a Keras model,

Lab objectives



- Define distribution strategy and set input pipelines.
- Create a Keras model.
- Define callbacks.



define callbacks,

Lab objectives



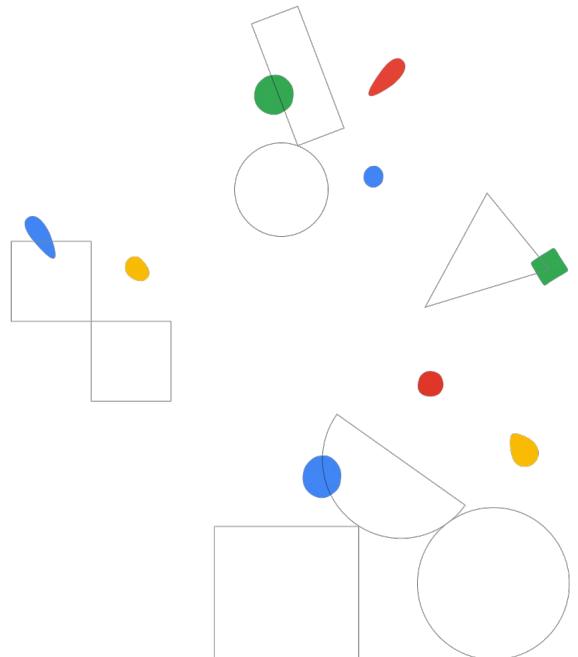
- ✓ Define distribution strategy and set input pipelines.
- ✓ Create a Keras model.
- ✓ Define callbacks.
- ✓ Train and evaluate a model.



and finally, train and evaluate a model.

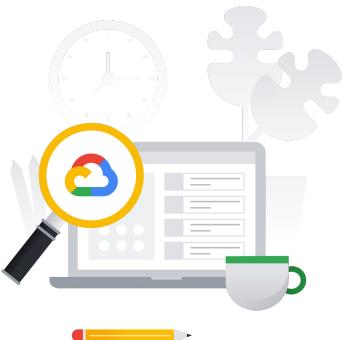
Distributed training using GPUs on Google Cloud's AI Platform

Module 03
Designing high-performance ML systems



This lab provides hands-on practice using Google Cloud's AI Platform to perform distributed training using the MirroredStrategy found within tf.keras. This strategy allows the use of the synchronous AllReduce strategy on a virtual machine with multiple GPUs attached.

Lab objectives

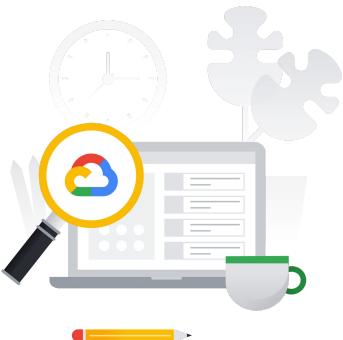


Set up the environment.



You'll start by setting up the environment,

Lab objectives



Set up the environment.

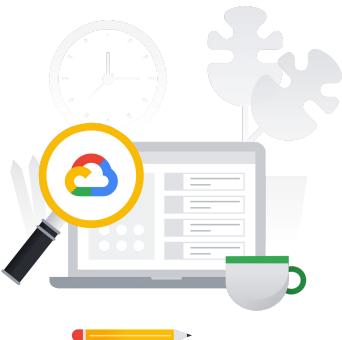


Create a deep neural network model
using the Fashion MNIST dataset.



then continue on to create a deep neural network model using the Fashion MNIST dataset,

Lab objectives



- Set up the environment.
- Create a deep neural network model using the Fashion MNIST dataset.
- Train that model using a MultiWorkerMirroredStrategy running on multiple GPUs.



and then, finally, you'll train that model using a `MultiWorkerMirroredStrategy` running on multiple GPUs.



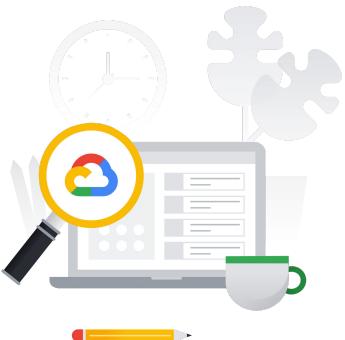
Lab

TPU-Speed data pipelines: tf.data.Dataset and TFRecords

Production Machine Learning Systems

In this lab, you'll get practice loading data from GCS with the `tf.data.Dataset` API to feed a TPU.

Lab objectives

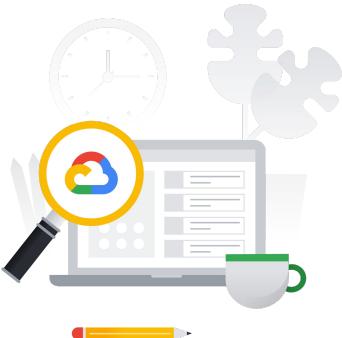


Use the `tf.data.Dataset API` to load training data.



You'll start by using the `tf.data.Dataset API` to load training data,

Lab objectives



Use the `tf.data.Dataset API` to load training data.



Use TFRecord format to load training data from Google Cloud Storage.



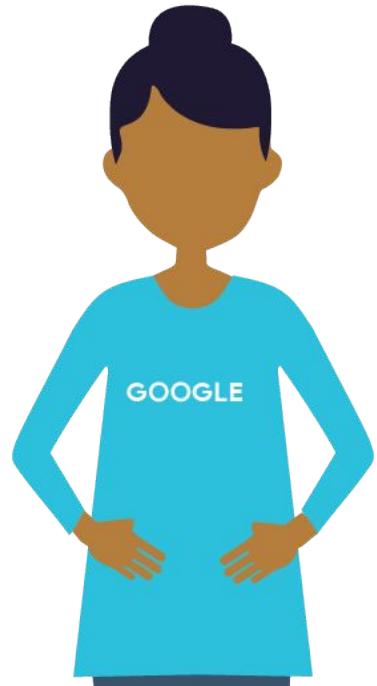
then use the TFRecord format to load training data from Google Cloud Storage.

Agenda

Distributed training

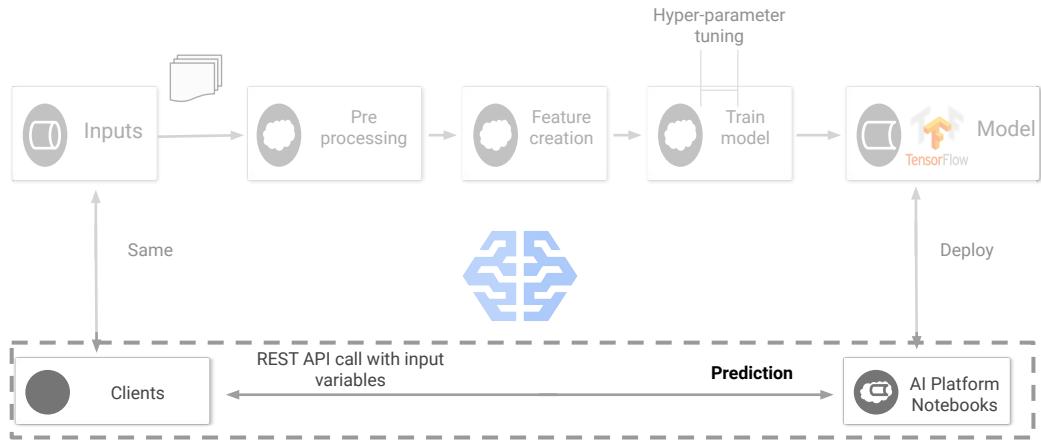
Faster input pipelines

Inference

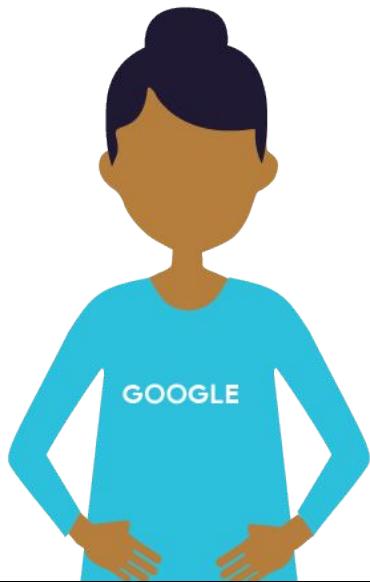


So far, we have looked at training performance. Now, we will look at performance when it comes to predictions.

Performance must consider prediction-time, not just training



How do you obtain high-performance inference?



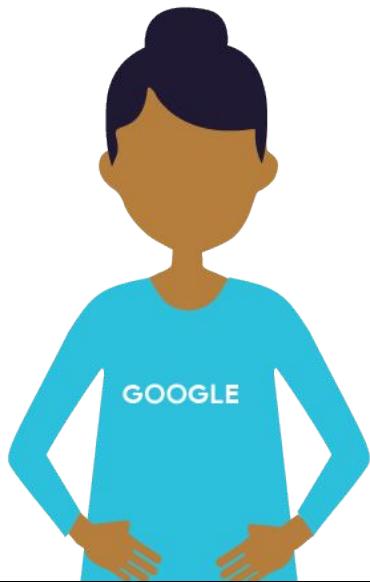
Aspects of performance during inference



You need to consider several aspects:

- * Throughputs requirements -- how many queries per second do you need to process?
- * Latency requirements -- how long can a query take?
- * Cost -- in terms of infrastructure **and** in terms of maintenance

Cost: <https://pixabay.com/en/tools-settings-options-system-98391/>



Implementation Options



REST/HTTP
API

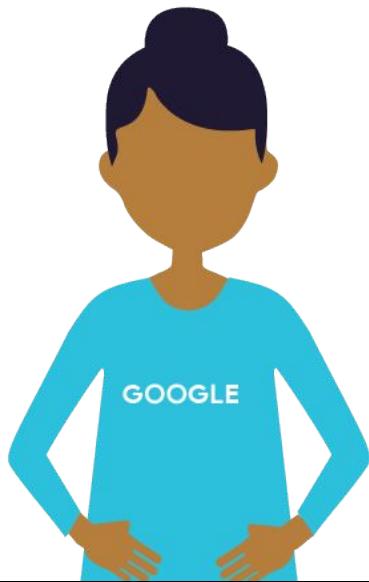
For Streaming
Pipelines

There are essentially three approaches to implementing this:

- Using a deployed model REST/HTTP API (for streaming pipelines)
- Using Cloud ML Engine batch prediction jobs (for batch pipelines.)
- Using Cloud Dataflow direct model prediction (for both batch and streaming pipelines)

Let's take the third option and delve into it a bit -- this will help clarify our terminology as well

Gear icon: <https://pixabay.com/en/gear-settings-options-icon-1077550/>



Implementation Options



REST/HTTP
API



Cloud Machine
Learning Engine

For Streaming
Pipelines

For Batch
Pipelines

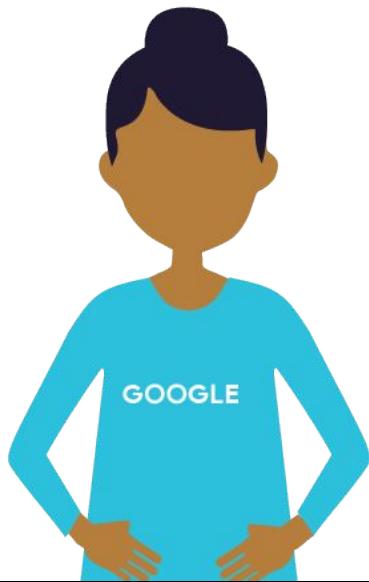
[idea: triptych]

There are essentially three approaches to implementing this:

- Using a deployed model REST/HTTP API (for streaming pipelines)
- **Using Cloud ML Engine** batch prediction jobs (for batch pipelines.)
- Using Cloud Dataflow direct model prediction (for both batch and streaming pipelines)

Let's take the third option and delve into it a bit -- this will help clarify our terminology as well

Gear icon: <https://pixabay.com/en/gear-settings-options-icon-1077550/>



Implementation Options



REST/HTTP API



Cloud Machine Learning Engine



Cloud Dataflow

For Streaming Pipelines

For Batch Pipelines

For Batch and Streaming Pipelines

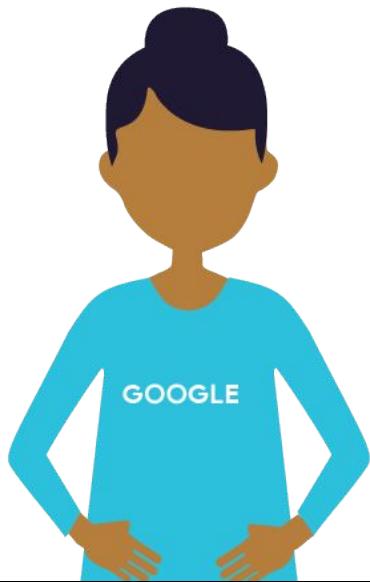
[idea: triptych]

There are essentially three approaches to implementing this:

- Using a deployed model REST/HTTP API (for streaming pipelines)
- Using Cloud ML Engine batch prediction jobs (for batch pipelines.)
- **Using Cloud Dataflow** direct model prediction (for both batch and streaming pipelines)

Let's take the third option and delve into it a bit -- this will help clarify our terminology as well

Gear icon: <https://pixabay.com/en/gear-settings-options-icon-1077550/>

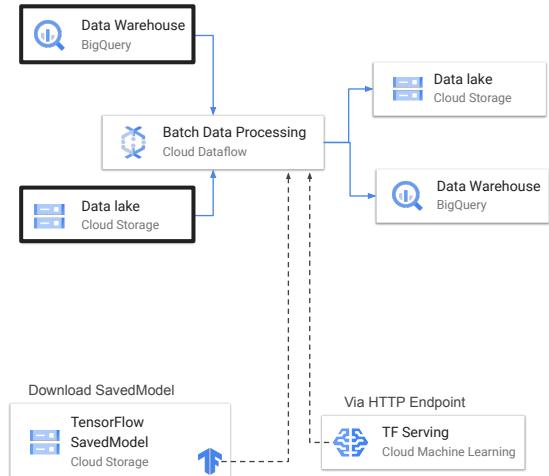
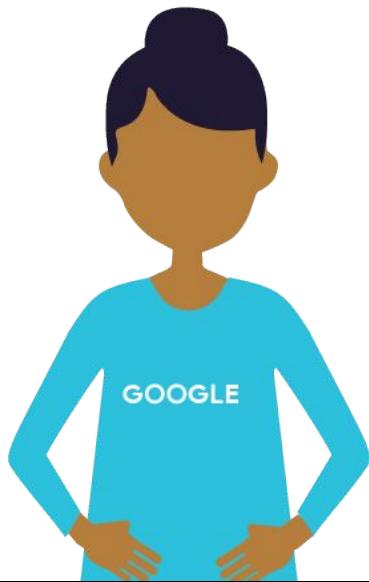


Batch = Bounded Dataset

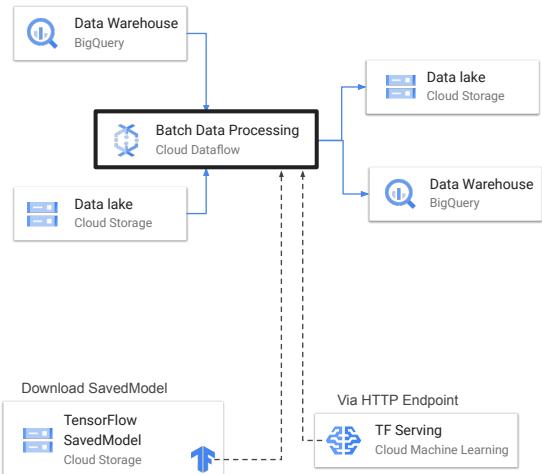
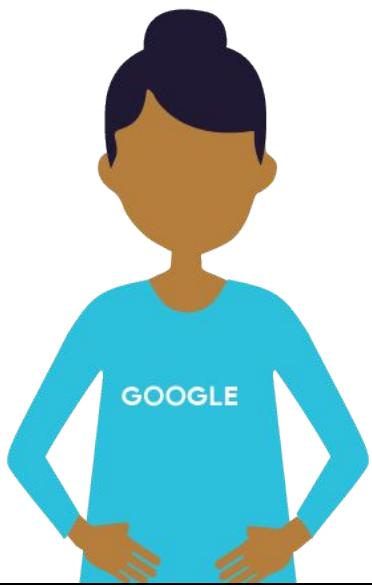


```
SELECT * FROM sales  
WHERE date = '2018-01-01'
```

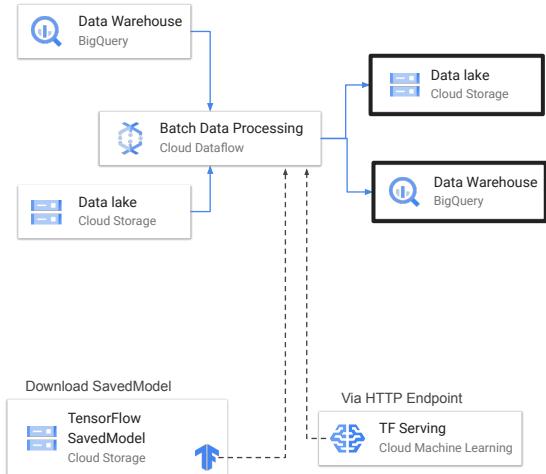
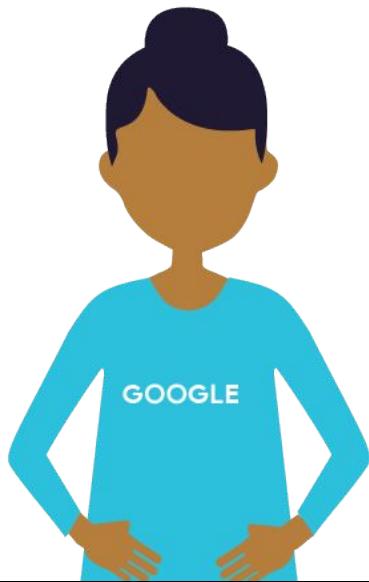
We are using the word “batch” differently from the word “batch” in ML training. Here, we are using batch to refer to a bounded dataset.



A typical batch data pipeline reads data from some persistent storage, either a data lake like Google Cloud Storage or a data warehouse like BigQuery,

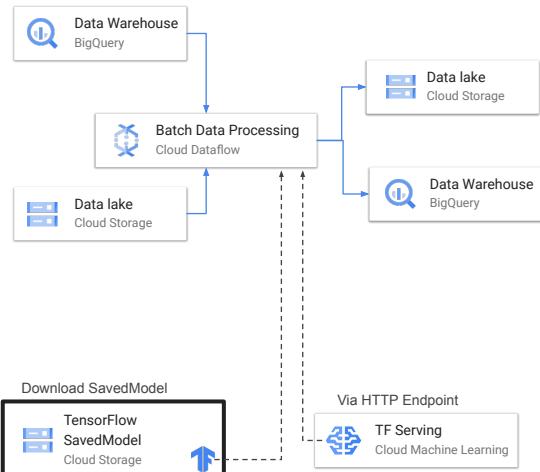
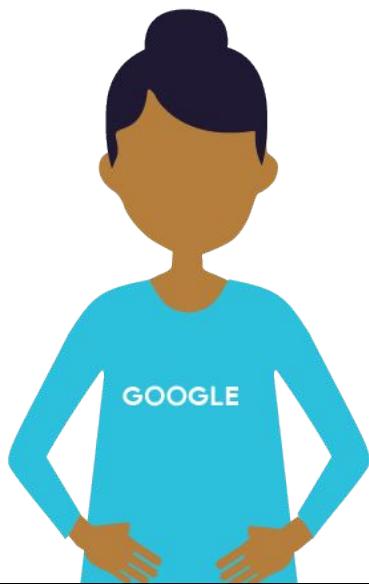


does some processing and

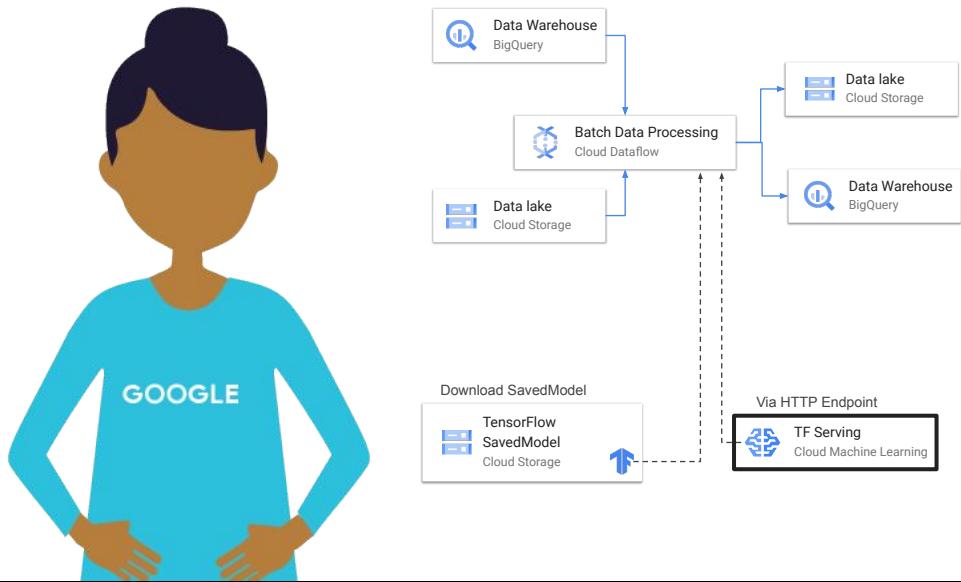


writes it out to the same or different format.

The processing, carried by Cloud Dataflow, typically enriches the data with the predictions of a ML model.



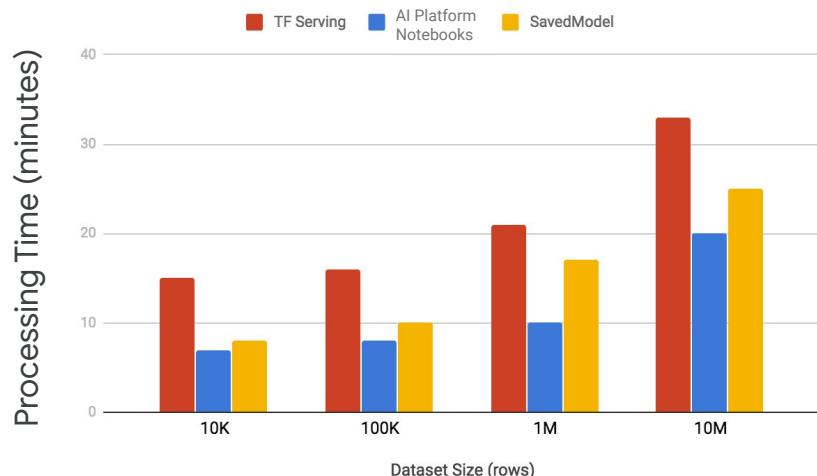
There are two options to do this, either by using a TensorFlow SavedModel and loading it directly into the Dataflow pipeline from Cloud Storage,



Or by using TF Serving and accessing it via a HTTP end-point as a microservice, either from Cloud ML Engine as shown or using Kubeflow, running on Kubernetes Engine.

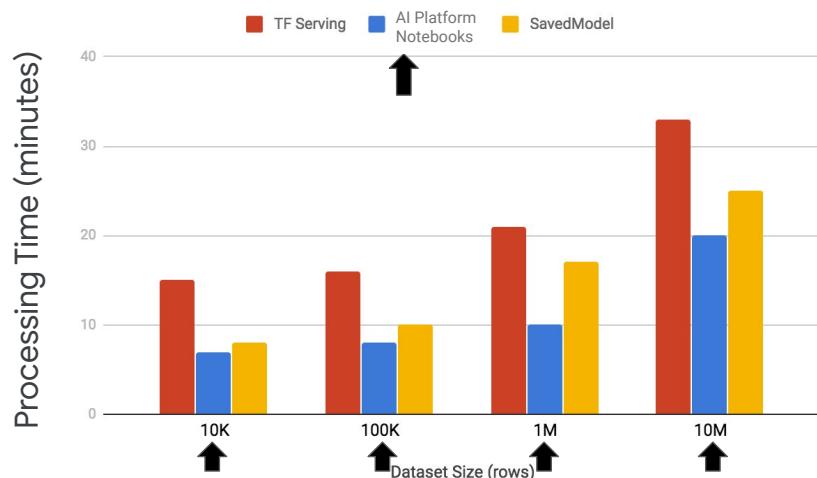
So far, we have used the HTTP end-point approach, but for performance reasons you might want to consider the SavedModel approach as well.

Performance for Batch Pipelines



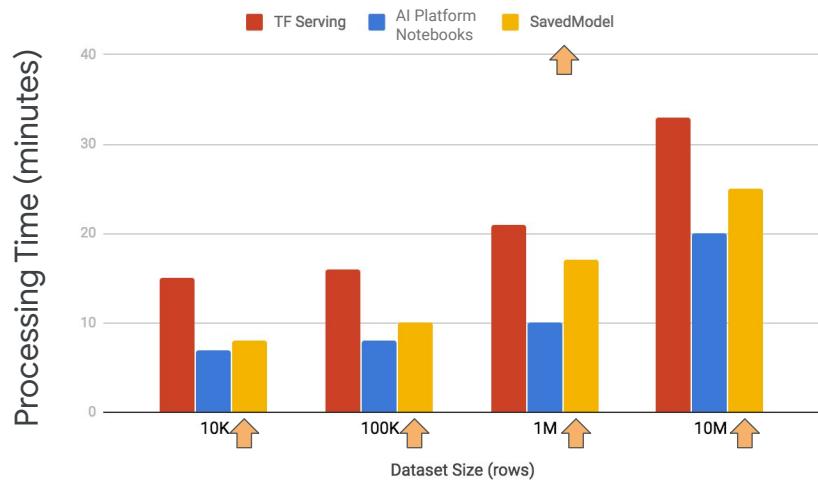
So, what option gives the best performance for batch pipelines?
As usual, this depends on which aspect is most important to you.

Performance for Batch Pipelines



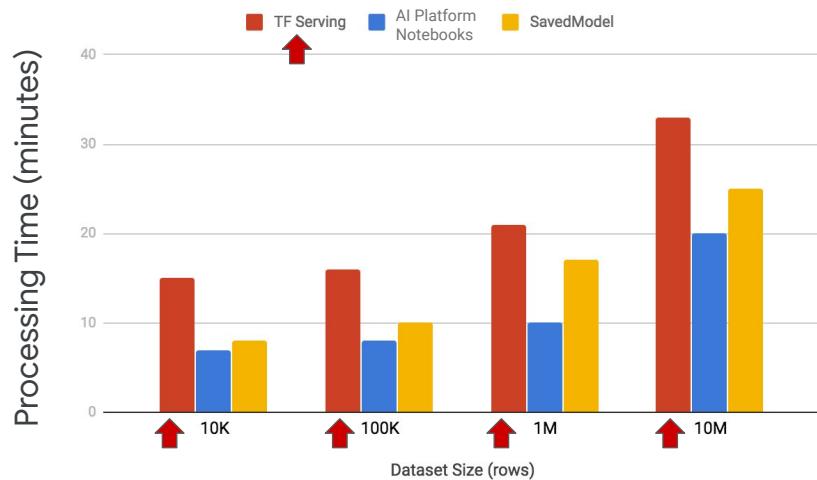
In terms of raw processing speed, you want to use Cloud ML Engine batch predictions.

Performance for Batch Pipelines



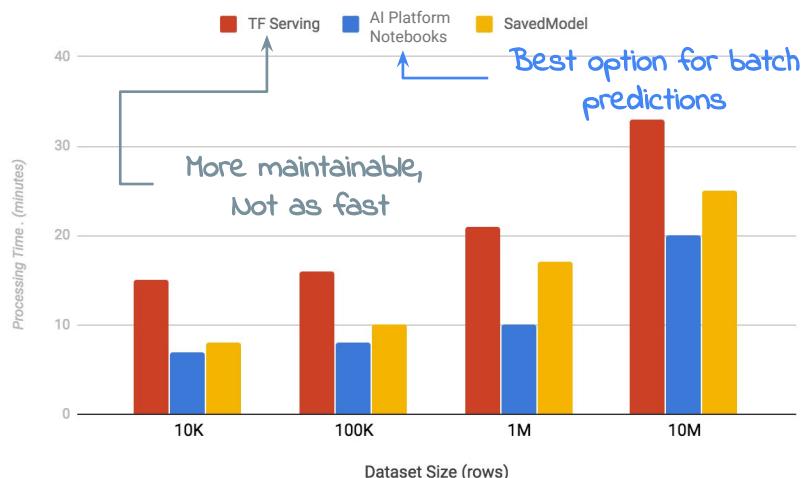
Next fastest is to directly load the SavedModel into your Dataflow job and invoke it.

Performance for Batch Pipelines



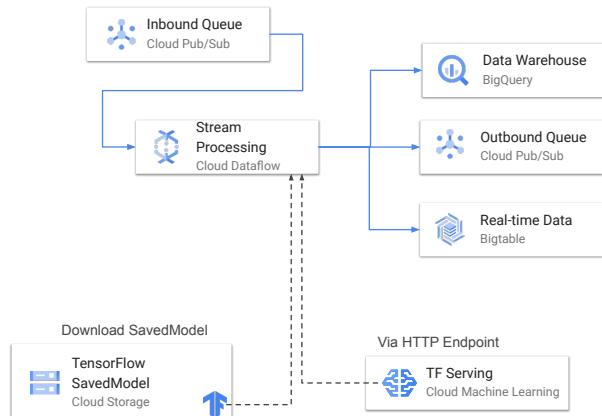
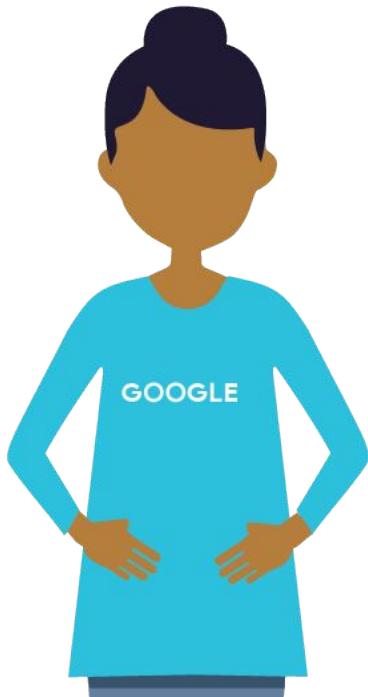
The third option in terms of speed is to use TensorFlow Serving on Cloud ML Engine.

Performance for Batch Pipelines



But if you want maintainability, the 2nd and 3rd options reverse. The batch prediction is still the best -- what's not to love about a fully managed service? But using online predictions as a microservice allows for easier upgradability and dependency management than loading up the current version into the Dataflow job.

This graph is from an upcoming solution -- see <https://cloud.google.com/solutions> -- by the time this video is available on Coursera, the solution might already have been published.

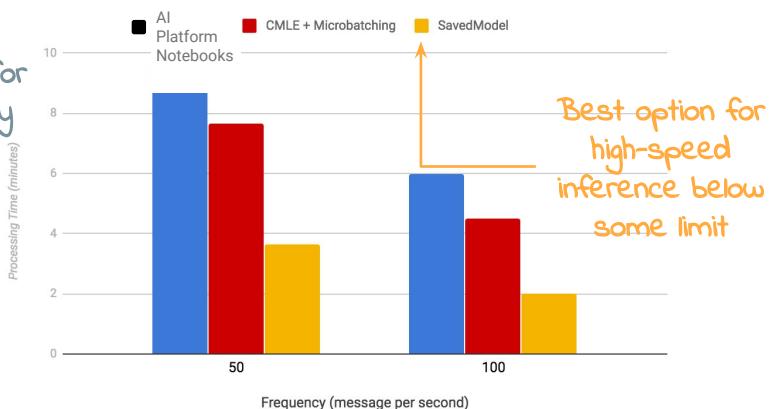


A streaming pipeline is similar, except that the input dataset is not bounded. So, we read it from an unbounded source like Pub/Sub and process it with Dataflow.

You have two options, of SavedModel or TensorFlow serving here as well, with TF serving hosted on Cloud ML Engine.

Performance for Streaming Pipelines

Best option for maintainability and speed



For streaming pipelines, the SavedModel approach is the fastest.

Using minibatching as we recommended earlier in the module on implementing serving helps reduce the gap between the TF Serving http end point approach supported by Cloud ML Engine and directly loading the model into the client. However, the CMLE approach is much more maintainable especially when the model will be used from multiple clients.

Another thing to keep in mind is that as the number of queries of second keeps increasing, at some point, the SavedModel approach will become infeasible, but the Cloud ML Engine approach will scale indefinitely.



Google Cloud

Hybrid ML Systems

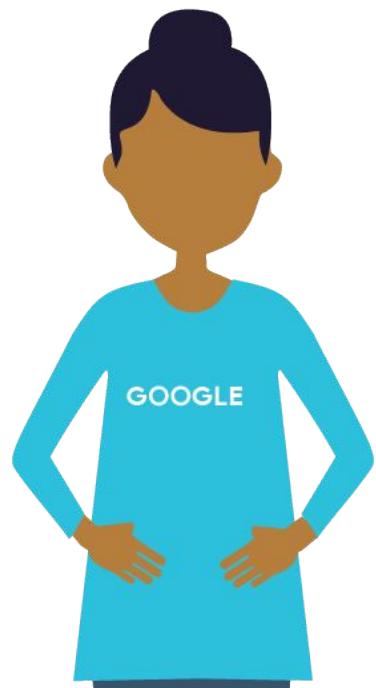
Lak Lakshmanan

Hi, I'm Lak and I lead the team that is putting together this course and specialization. In this module, we will look at building hybrid machine learning models.

Learn how to...

Build hybrid cloud machine
learning models

Optimize TensorFlow graphs for
mobile

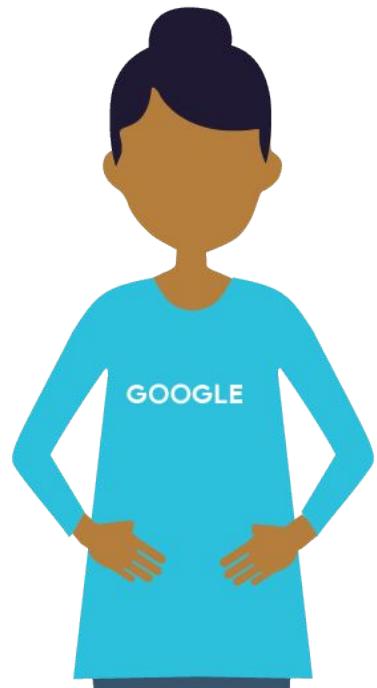


You will learn how to
Build hybrid cloud machine learning models
And how to
Optimize TensorFlow graphs for mobile

Agenda

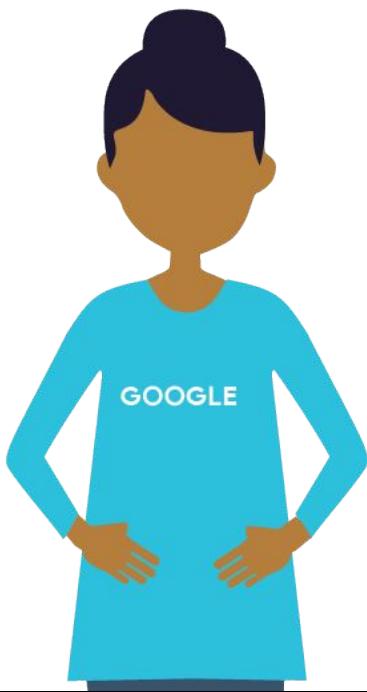
Kubeflow for hybrid cloud

Optimizing TensorFlow for
mobile



Let's start by discussing a technology called Kubeflow which helps us build hybrid cloud machine learning models.

But why are we discussing hybrid in the first place? Why would you need anything other than Google Cloud?



Choose from
ready-made ML models

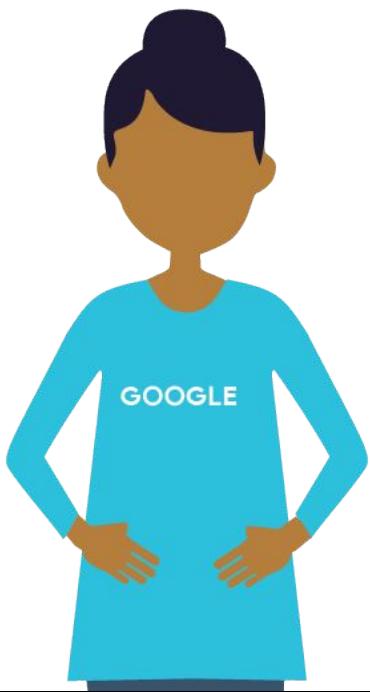


Vision Translation Speech Natural
Language

Google Cloud is a great place to do machine learning.

You have access to ready-made models like the Vision API, Natural Language API etc.

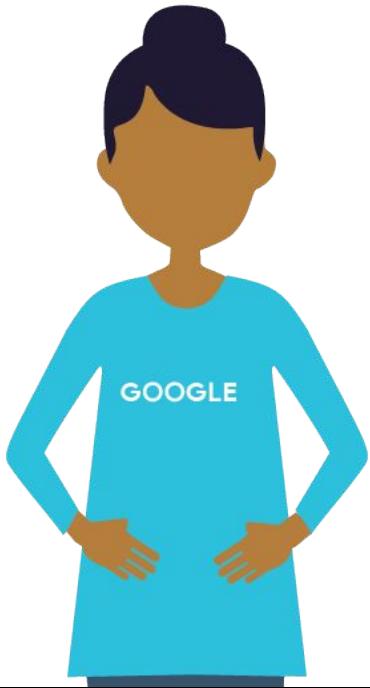
The key aspect of these models is that they are trained on Google's massive datasets.



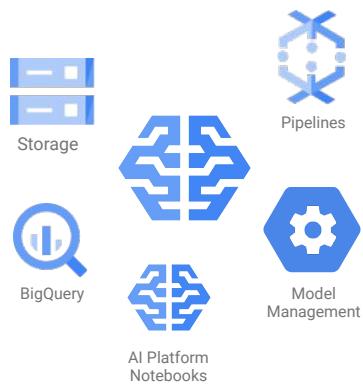
Customize ready-made
ML models



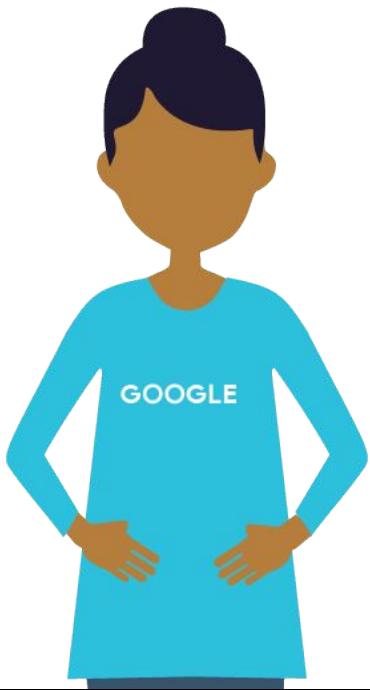
Sometimes, ready-to-run models like the Vision API don't quite fit. In that case, you might want to train a family of models using your images and labels to customize and add to the Vision API. That's called Auto-ML and is possible only on the Cloud.



Build, train, and serve, your
own custom ML Models

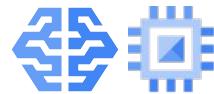


Even if you are building custom machine learning models, you have reason to do ML on GCP -- while TensorFlow is open-source, a serverless execution environment like Cloud ML Engine allows your data scientists to not have to worry about infrastructure. Plus, of course, the integration with distributed cloud storage and serverless and BigQuery make the overall development experience a lot better than if you had to provision and manage all that infrastructure yourself.

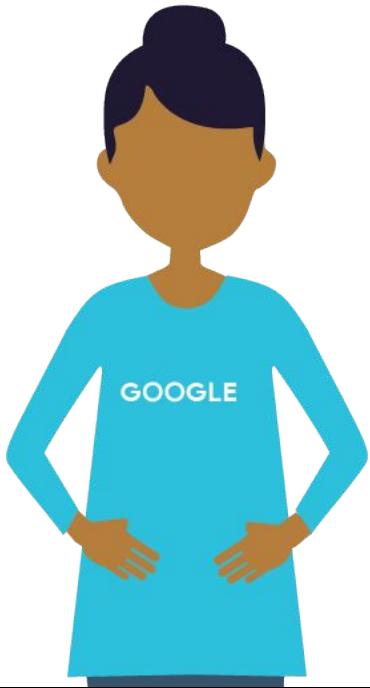


ML runtimes in a cloud-native environment

1. Prototype with AI
Platform Notebooks
or Deep Learning
Image



So far in this series of courses, we have assumed that you are in a cloud-native environment. So, we prototyped and developed our code using Cloud Datalab and once we had the code working on a small sample of data, ...



ML runtimes in a cloud-native environment

1. Prototype with AI Platform Notebooks or Deep Learning Image



2. Distribute and autoscale training and predictions with Cloud ML Engine

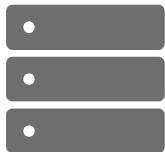


...we then submitted the training job to Cloud ML Engine to operate on the full dataset. We also served out model using Cloud ML Engine, so that we didn't have to worry about infrastructure.

There are times, however, when you can not be fully cloud-native.
What kinds of situations?

You may not be able to do machine learning solely on Google Cloud

Tied to On-Premise
Infrastructure



[idea: diagrams to illustrate the various scenarios]

You may not be able to do machine learning solely on the cloud

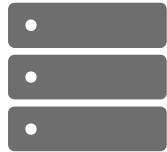
Perhaps, you are tied to on-premises infrastructure and your ultimate goal is to move to the public cloud perhaps in a few years.

Perhaps there are constraints about being able to move your training data off your on-prem cluster or data center.

So you have to make do with the system you have.

You may not be able to do machine learning solely on Google Cloud

Tied to On-Premise
Infrastructure



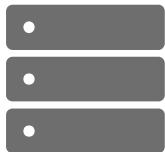
Multi Cloud System
Architecture



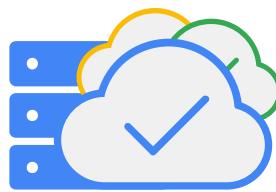
Or maybe the data is being produced by a system that is running on a different cloud. Or the model predictions need to be consumed from another cloud. So, you need a multi-cloud solution, not a solution that is solely GCP.

You may not be able to do machine learning solely on Google Cloud

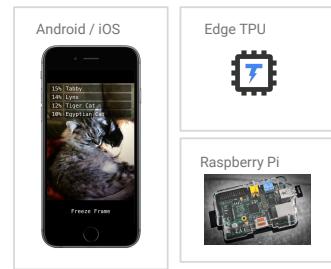
Tied to On-Premise
Infrastructure



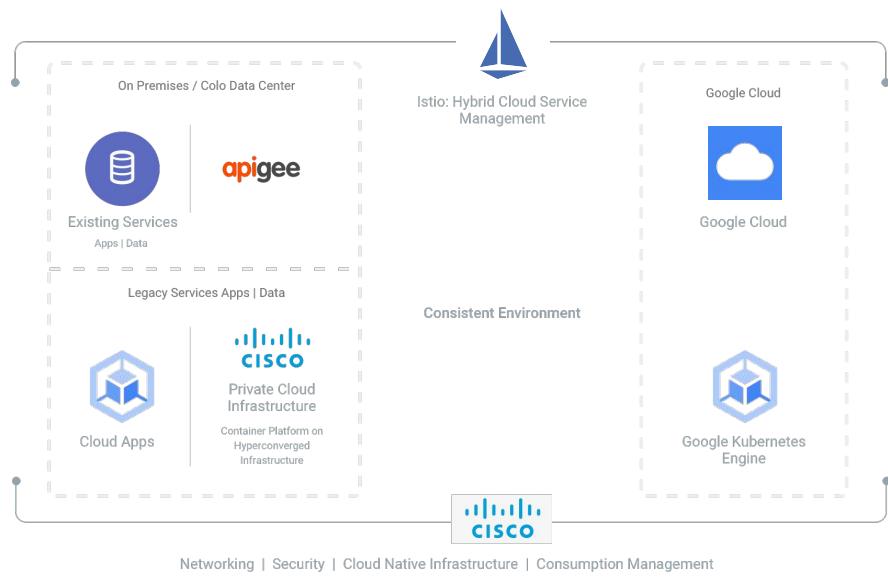
Multi Cloud System
Architecture



Running ML on the edge



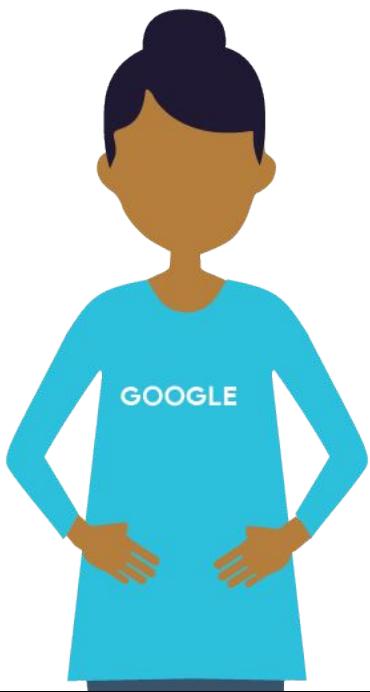
Or maybe you are running machine learning on the edge, and connectivity constraints force you to have to do your predictions on the edge, on the device itself. So you have to do inference-on-the-edge. This is a common situation in internet-of-things.



Here's an example of Cisco's hybrid cloud architecture -- Cisco partnered with Google Cloud to bridge their private cloud infrastructure and existing applications with Google Cloud Platform. Note the use of Google Kubernetes Engine to manage their container deployments.

Kubernetes is a container-orchestration system that was designed and then open-sourced by Google.

<https://cloud.google.com/cisco/>



Kubernetes minimizes
infrastructure management



Using Kubernetes, it is possible to orchestrate containers whether they are running on-prem, or on the cloud. Any cloud. So, one possible solution to retain the ability to move fast, minimize your infrastructure management needs, and still retain the ability to move or burst to GCP is to use Kubernetes.

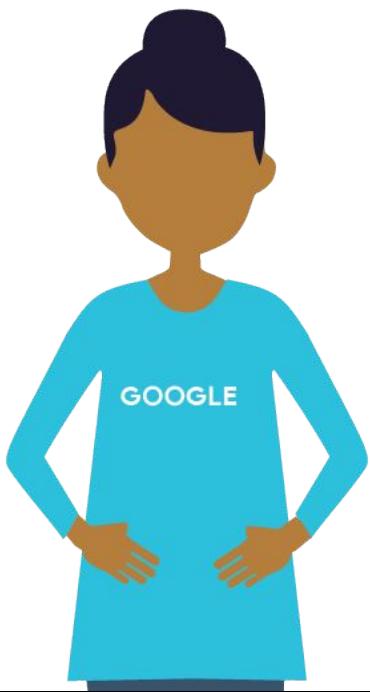
Logos: <https://cloud.google.com/kubernetes-monitoring/>



Kubeflow enables hybrid
machine learning



Specifically, a project called Kubeflow



Kubeflow enables hybrid
machine learning

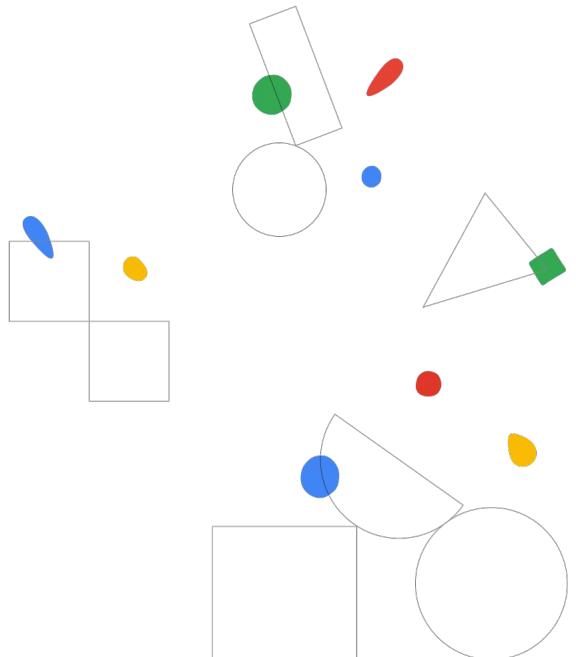


Kubeflow

helps you migrate between cloud and on-prem environments. Kubeflow is an open source machine learning stack built on K8s.



Kubeflow

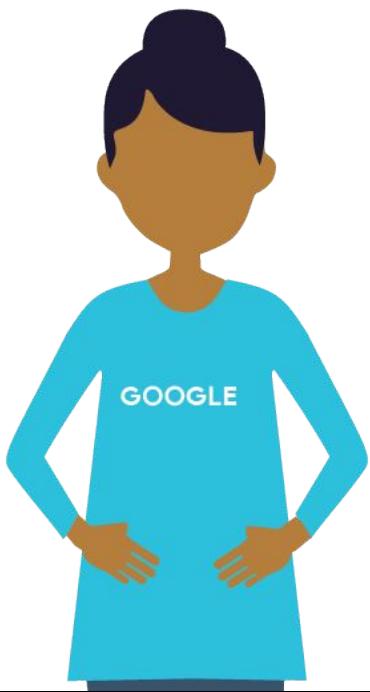


Module 04

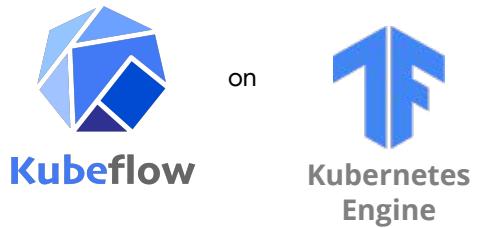
Building Hybrid ML systems

Welcome back.

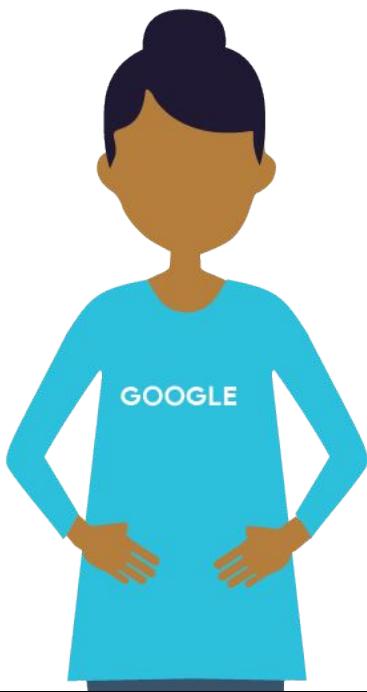
In this module you will learn how to



Kubeflow enables hybrid
machine learning



On Google Cloud, you can run Kubeflow on Google Kubernetes Engine, GKE.

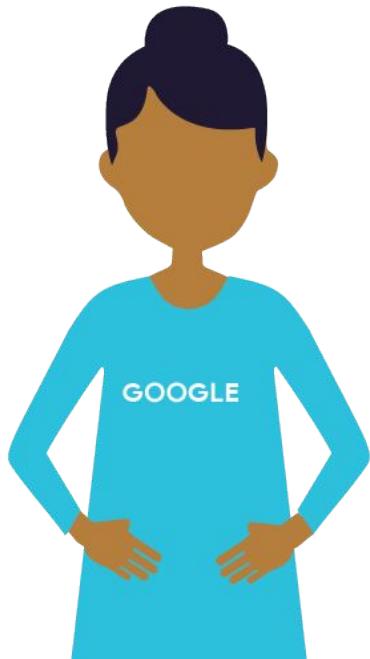


Kubeflow enables hybrid
machine learning



However, you can run Kubeflow on anything from a phone to a laptop to an on-prem cluster.

Your code remains the same. Some of the configuration settings change. That's it.



 Composability

 Portability

 Scalability

In order to build hybrid machine learning systems, that work well both on-prem and in the cloud, your ML framework has to support three things:
Composability
Portability.
and
Scalability.

Composability

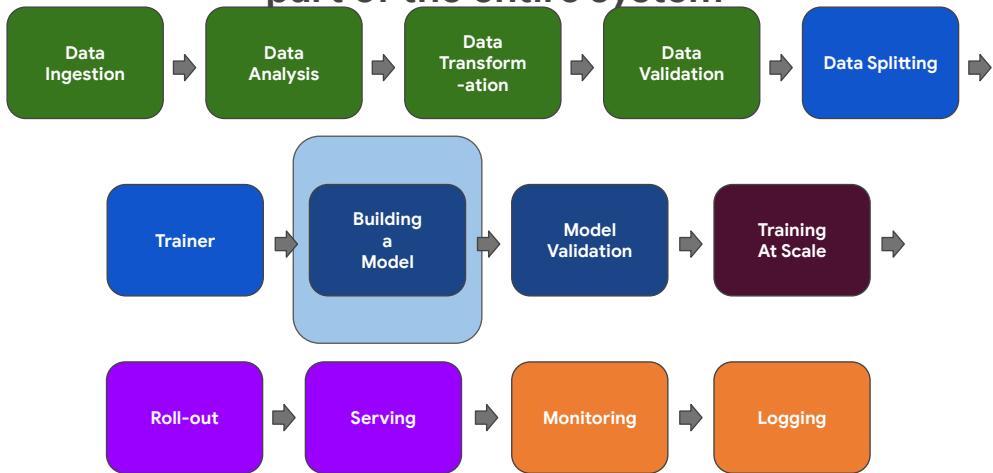
Building
a
Model

First composability.

When people think about ML, they think about building a model.

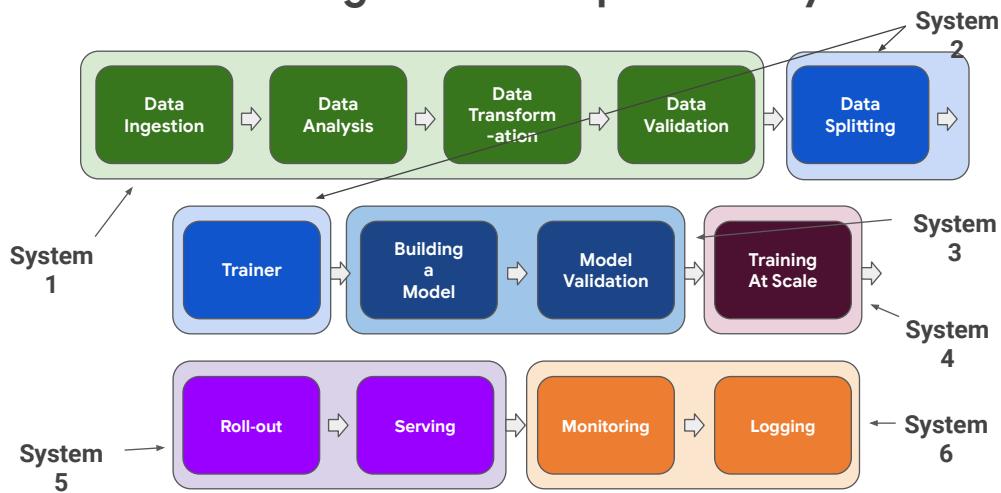
Tensorflow, pytorch, numpy, etc.

Building a model is only one part of the entire system



But the reality is 95% of the time is spent NOT building a model... it's all the other stuff.

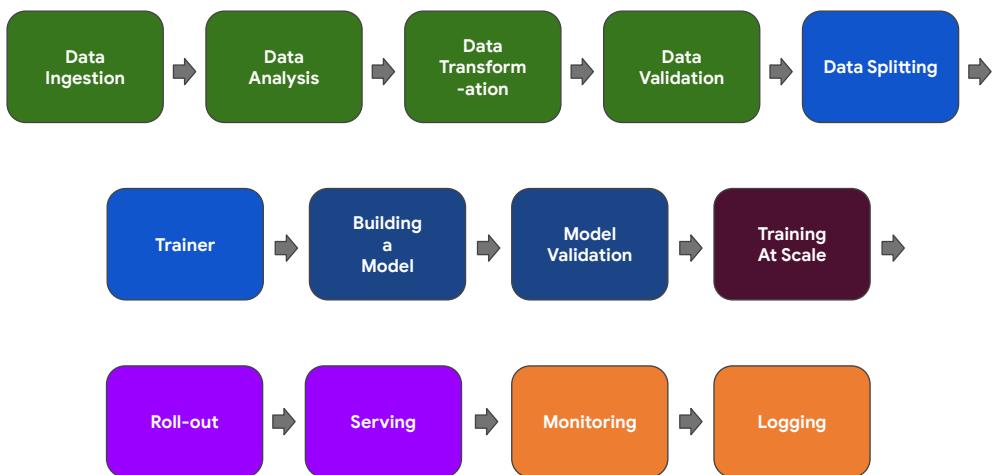
Each ML Stage is an Independent System



Each ML stage - data analysis, training, model validation, monitoring, they are all independent systems.

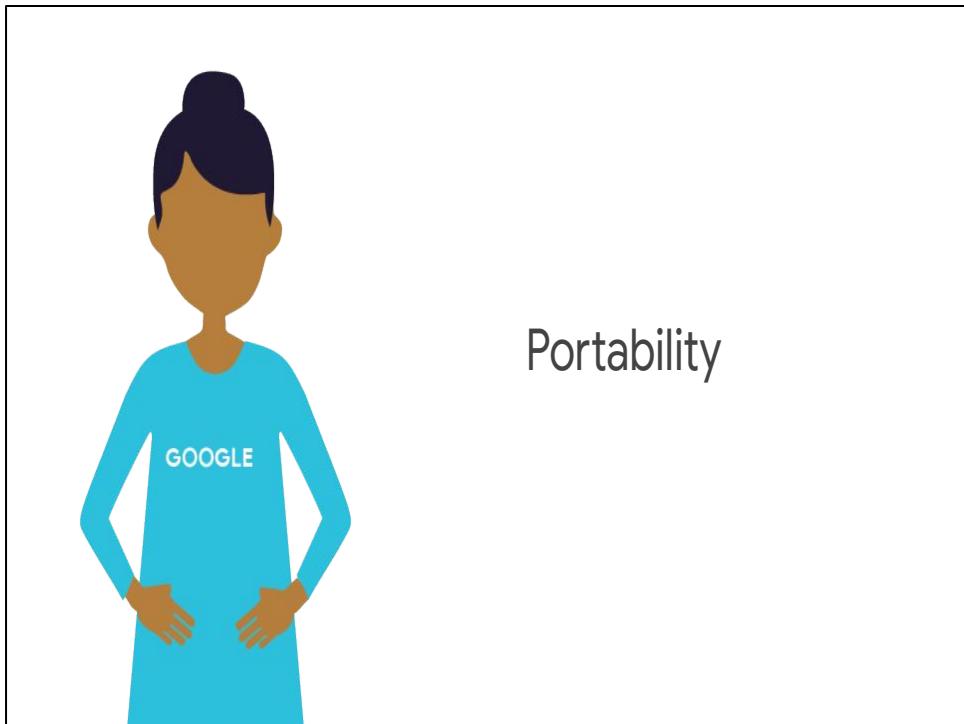
EVERYONE has a different way to handle all these boxes.

Composability is about microservices



So when we say composability, it's about the ability to compose a BUNCH of microservices together.

And the option to use what makes sense for your problem.



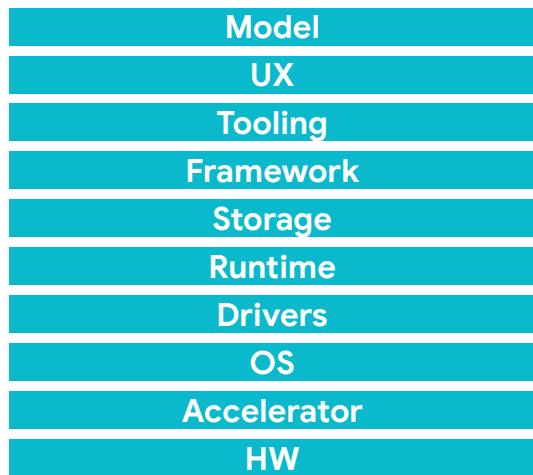
Portability

Now you've built your specific framework, you want to move it around.

This is where we get into portability.

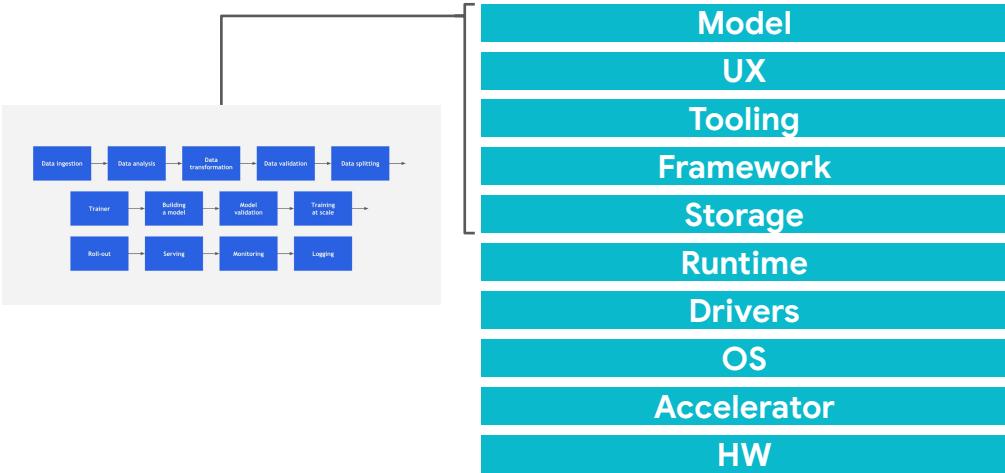
Portability

Experimentation



The stack you use is likely made up of all these components (and lots more)!

Portability



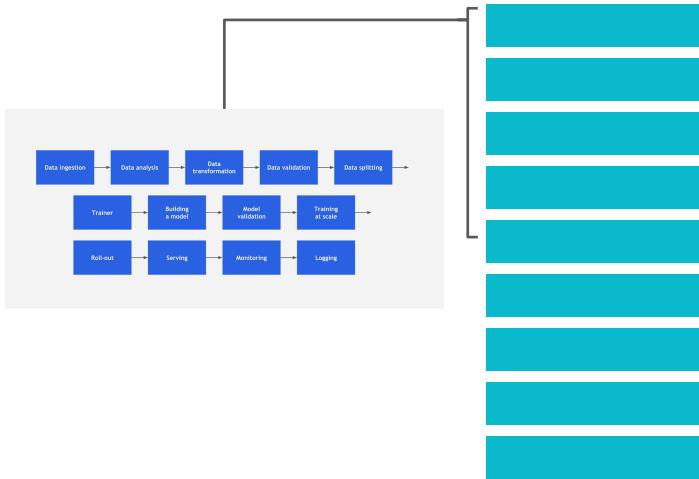
And all those microservices I detailed earlier only touch a small number of them.

But you do it, you configure every stage in the stack and it's FINALLY running.

What is this good for? What happens next? You need to think about the ML workflow.

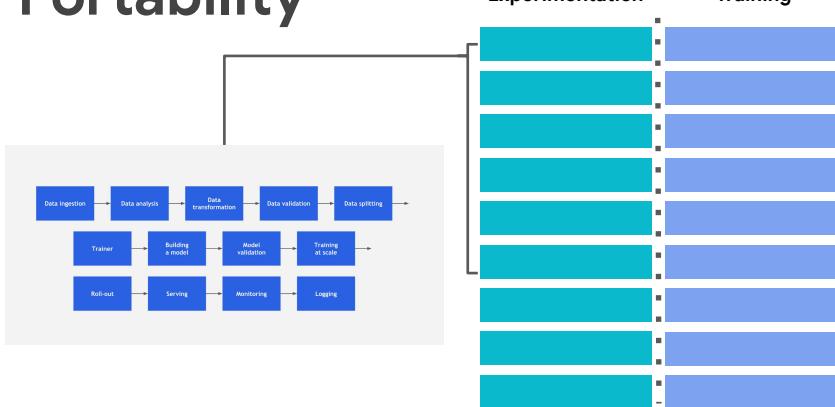
Portability

Experimentation



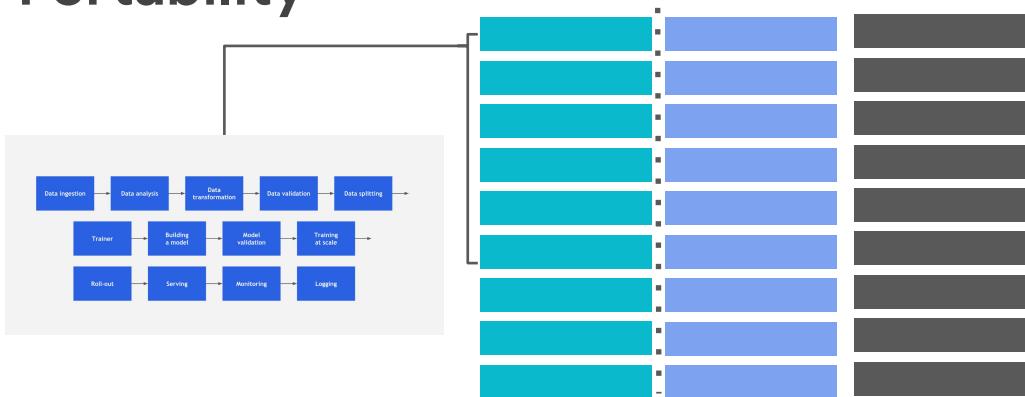
Remember that you did all this just so that you could develop the model. We'll call that experimentation. But once you have the code running, what do you need to do? That's right, you need to train the model on the full dataset. You probably can't do it on the small setup on which you did all your initial development.

Portability



So, you start up a training cluster.... And you've got to do it all over again.
All the configuration, all the libraries, all the testing. You've got to repeat it for the new environment.

Portability



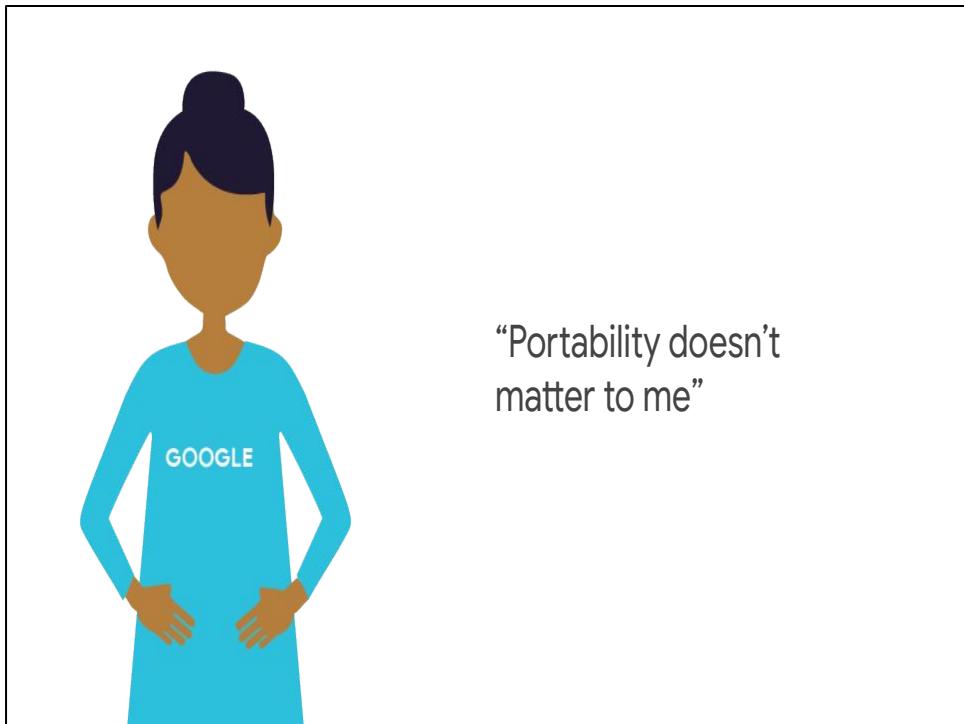
And then, chances are you've got to do it AGAIN to move to the cloud.

Because, remember, we said, we want a hybrid environment.

An ML model that helps you train on the cloud, and predict on the edge.

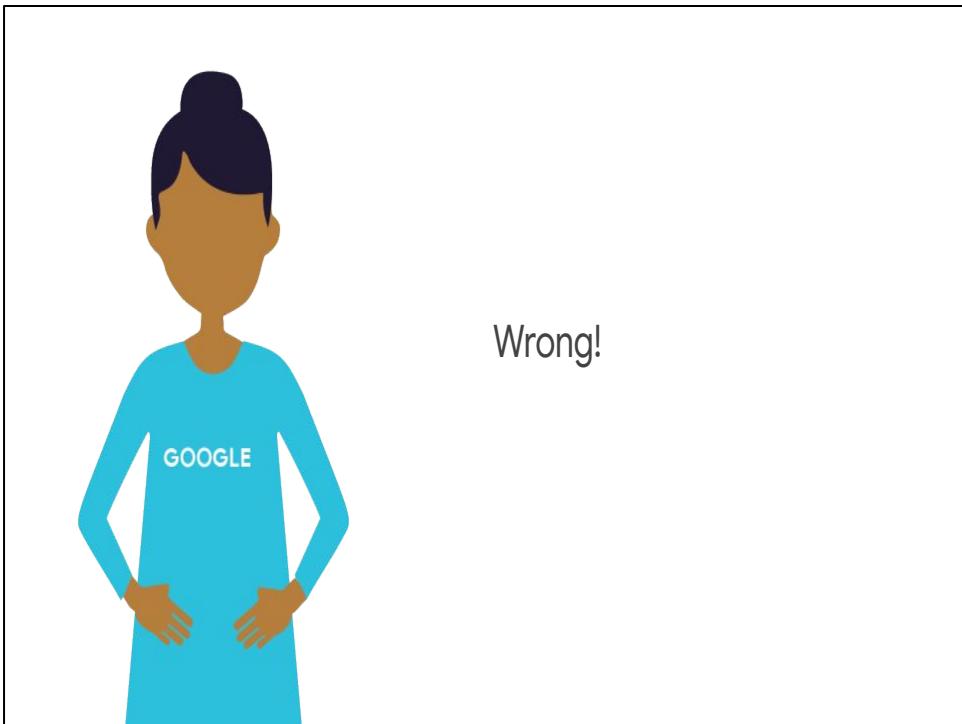
Or train on-the-cloud, but predict on-premises.

The point is that you have to configure the stack over and over again for each environment you need to support.



“Portability doesn’t
matter to me”

Maybe at this point, you are thinking “pfft ... that doesn’t matter to me.” I *never* have to change environments. I will use only one environment.



Wrong!

Wrong.

Joe Beda
@jbeda

The way I think about it: every difference between dev/staging/prod will eventually result in an outage.

6:25 PM - 19 Oct 2017

54 Retweets 107 Likes

3 replies 54 retweets 107 likes

Joe Beda is the CTO of Heptio, a startup focused on bringing Kubernetes to everyone. Before that, he was at Google. He co-founded Kubernetes and started Google Compute Engine.

His vast experience building production systems shows in this quote, which is not even about machine learning –

“The way I think about it: every difference between dev/staging/prod will eventually result in an outage.”



Joe Beda

@jbeda

Following



The way I think about it: every difference between dev/staging/prod will eventually result in an outage.

6:25 PM - 19 Oct 2017

54 Retweets 107 Likes



3



54



107

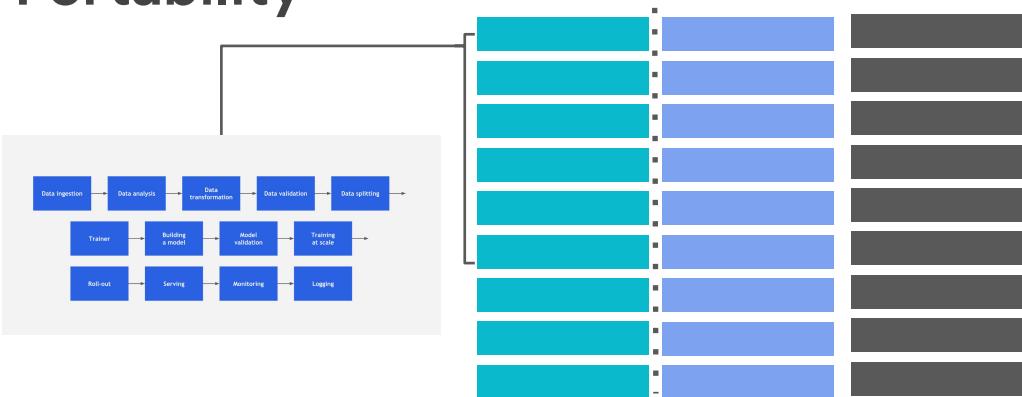


Notice the first environment that he mentions. It's dev.



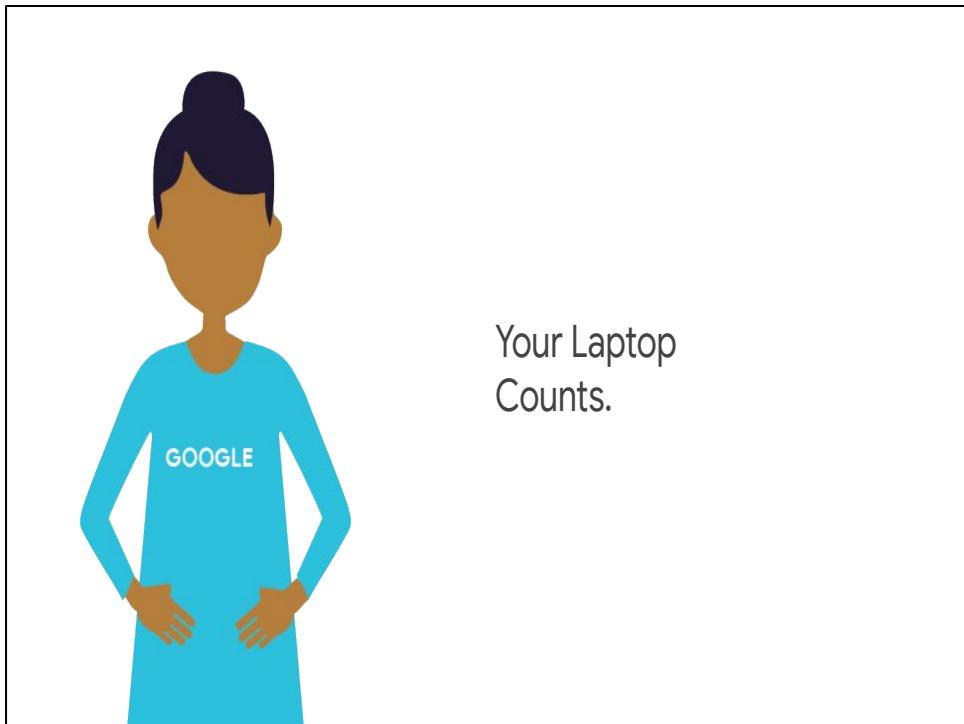
Your development environment is an environment.

Portability



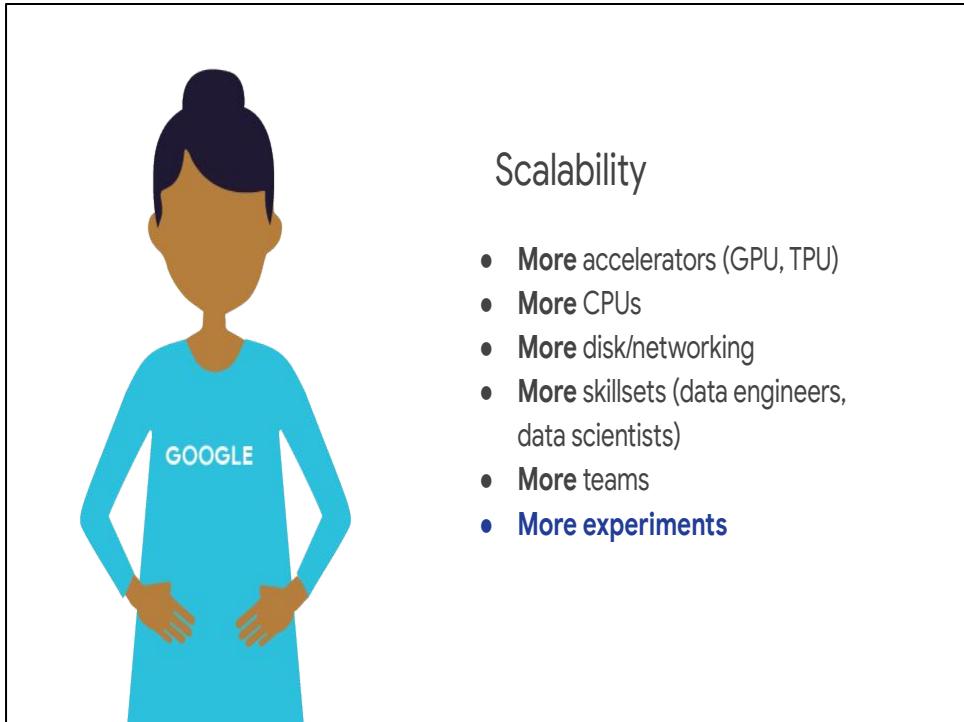
Portability? It is essential.

And then, of course, you've got to do it AGAIN when your inputs change, or your boss tells you to train faster by training on more machines. You find that you have to change environments over and over again.



Your Laptop
Counts.

Your laptop counts as environment #1. And you don't do production services on your laptop. So ...



- **More** accelerators (GPU, TPU)
- **More** CPUs
- **More** disk/networking
- **More** skillsets (data engineers, data scientists)
- **More** teams
- **More experiments**

Finally scalability.

You always hear about Kubernetes being able to scale, and that's true!

But scalability in ML means so many more things

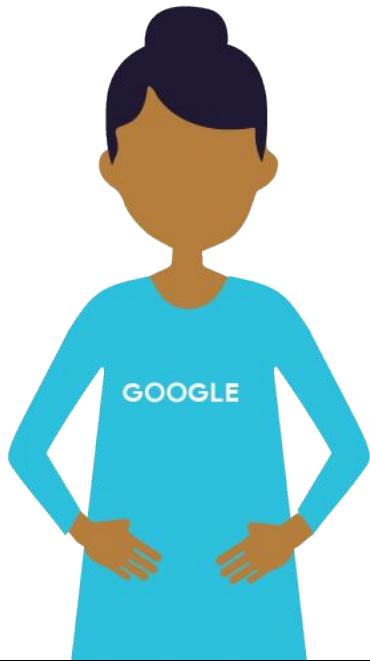
Accelerators

Disk

Skillsets (software engineers vs. researchers)

Teams

Experiments



≡🎵 Composability

💼 Portability

↗ Scalability

So that's what we think of when we think of machine learning in a hybrid cloud environment.

Composability. Portability. Scalability.

Build hybrid cloud ML
models with Kubeflow

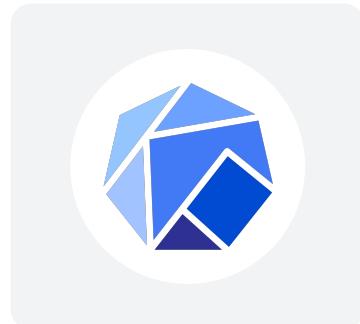


Optimize TensorFlow
graphs for mobile

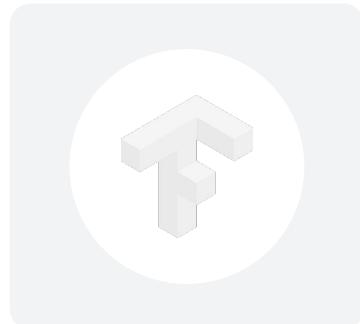


build hybrid cloud machine learning models with Kubeflow and how to optimize
TensorFlow graphs for mobile.

Build hybrid cloud ML
models with Kubeflow



Optimize TensorFlow
graphs for mobile



To begin, let's explore Kubeflow, an open-source machine learning platform designed to enable the use of machine learning pipelines to orchestrate complicated workflows running on Kubernetes.

Kubeflow helps build hybrid cloud machine learning models. But why are we discussing hybrid in the first place? Why would you need anything other than Google Cloud?

Cloud-native environment

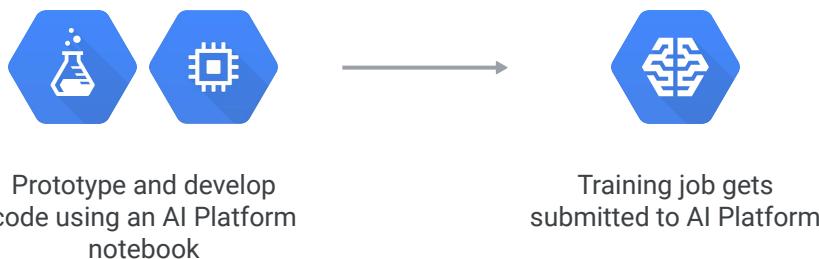


Prototype and develop
code using an AI Platform
notebook



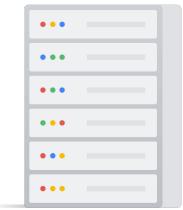
So far in this course we've focused on cloud-native environments, which involve prototyping and developing code using an AI Platform notebook. Once that code is working on a small sample of data...

Cloud-native environment



...the training job gets submitted to AI Platform to operate on the full dataset.

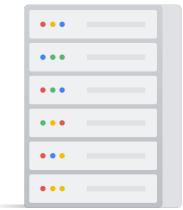
There are, however, scenarios when cloud-native, or conducting machine learning solely on Google Cloud, is not an option. Let's look at some of these scenarios now.



Tied to on-premises
architecture



Maybe you're tied to on-premises infrastructure, or there are other constraints preventing the option to move your training data off an on-prem cluster or data center.



Tied to on-premises
architecture

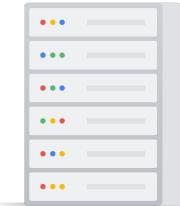


Multi-cloud solution
architecture



Alternatively, perhaps you require a multi-cloud solution architecture, one that does not rely solely on Google Cloud.

This could be because you're working with data that is being produced by a system that is running on a different cloud provider, or because model predictions need to be consumed from another cloud.



Tied to on-premises
architecture



Multi-cloud solution
architecture

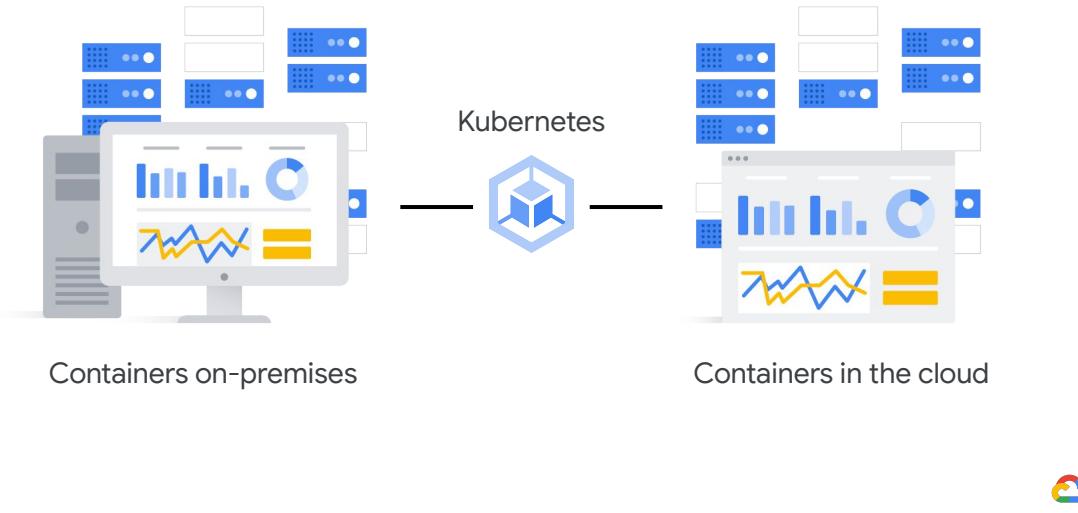


Running ML
on the edge



Or maybe you are in an initial phase of an ML project running machine learning on the edge, where you're working on a local developer workstation.

Training at scale typically happens in a cloud environment; however, inference and distributed training can happen at the edge. The edge means that predictions happen on a smart device, which is common in internet-of-things.



Using Kubernetes, it's possible to orchestrate containers that run either on-premises or in the cloud. And that can be any cloud.

Using Kubernetes allows for speed, the ability to minimize infrastructure management needs, all while being able to move or burst to Google Cloud.



Kubeflow



Kubeflow is the machine learning toolkit for Kubernetes, and it brings a number of benefits.



Kubeflow

Makes deploying machine learning workflows on Kubernetes simple, portable, and scalable



It makes deploying machine learning workflows on Kubernetes simple, portable, and scalable.



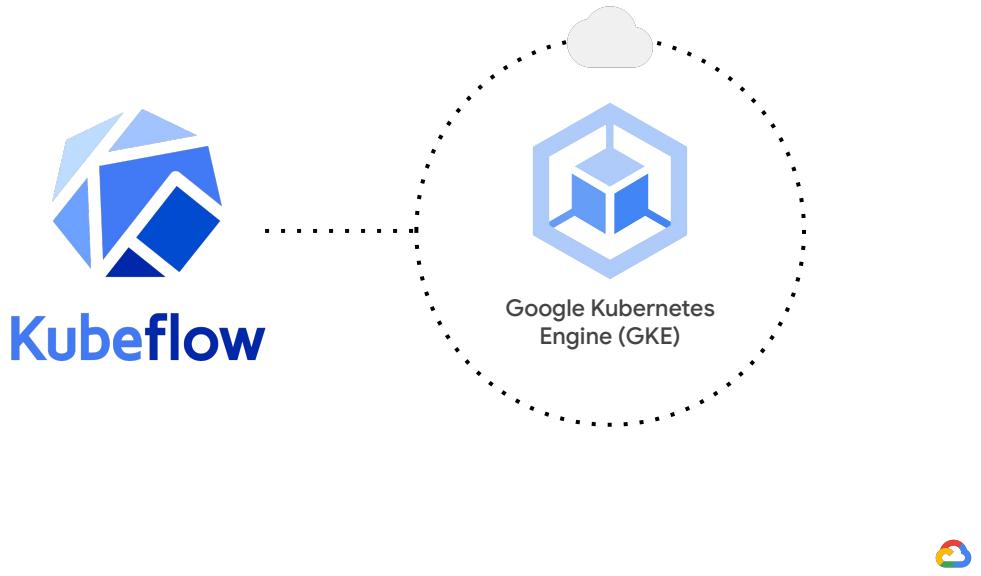
Kubeflow

Makes deploying machine learning workflows on Kubernetes simple, portable, and scalable

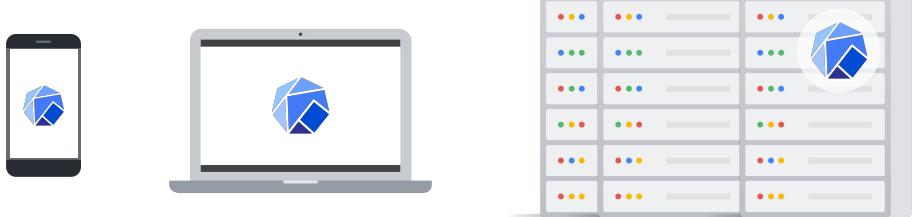
Extends Kubernetes' ability to run independent and configurable steps



It also extends Kubernetes' ability to run independent and configurable steps, with machine learning specific frameworks and libraries.



And since Kubeflow is open source, it can run Google Kubernetes Engine, which is part of Google Cloud.



However, Kubeflow can actually run on anything—whether it's a phone, a laptop, or an on-premise cluster.

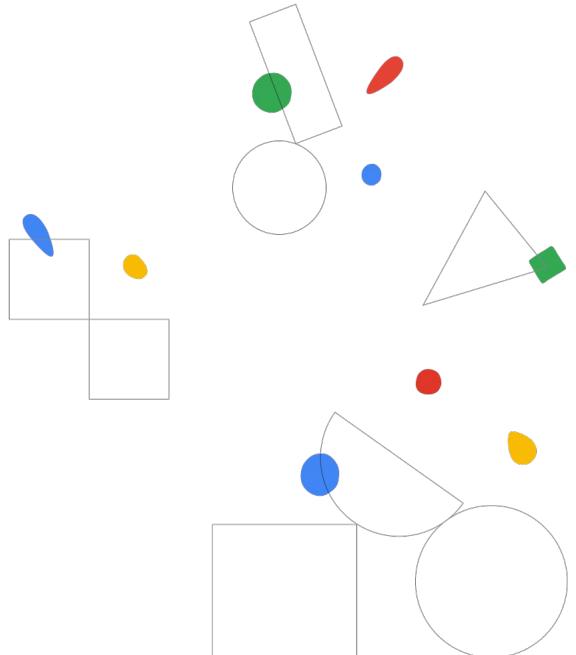
Regardless of where it's run, the code remains the same. Some of the configuration settings just change.



Lab

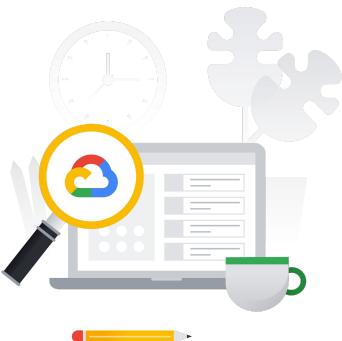
Kubeflow Pipelines with Google Cloud's AI Platform

Module 04
Building Hybrid ML systems



This lab provides hands-on practice installing and using Kubeflow Pipelines to orchestrate Google Cloud services in an end-to-end ML pipeline.

Lab objectives

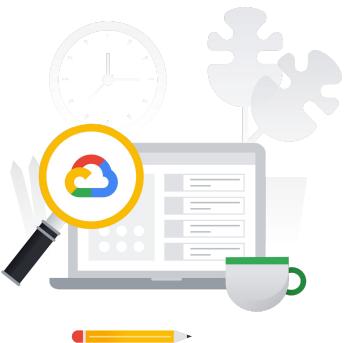


Create a Kubernetes cluster and install Kubeflow Pipelines.



To begin, you'll create a Kubernetes cluster and install Kubeflow Pipelines.

Lab objectives



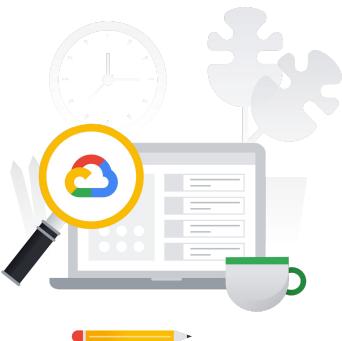
Create a Kubernetes cluster and install Kubeflow Pipelines.

Launch an AI Platform notebook.



Next, you'll launch an AI Platform notebook.

Lab objectives

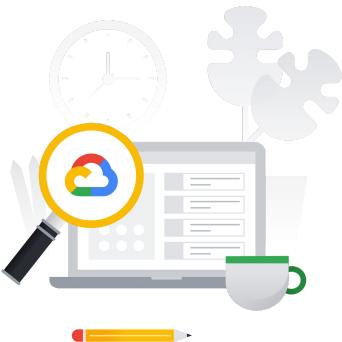


- ✓ Create a Kubernetes cluster and install Kubeflow Pipelines
- ✓ Launch an AI Platform notebook
- ✓ Create and run an AI Platform Pipelines instance.



From there, you'll create and run an AI Platform Pipelines instance.

Lab objectives



- ✓ Create a Kubernetes cluster and install Kubeflow Pipelines.
- ✓ Launch an AI Platform notebook.
- ✓ Create and run an AI Platform Pipelines instance.
- ✓ Run a Python function-based pipeline.



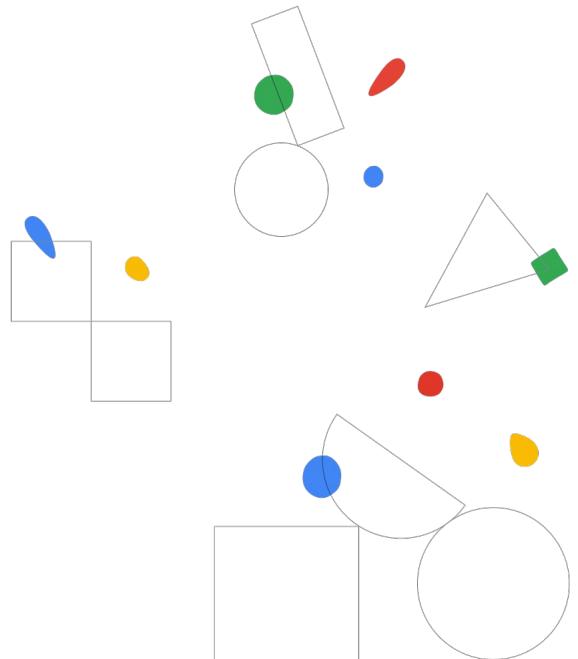
And finally, you'll run a Python function-based pipeline.



Optimizing TensorFlow for mobile

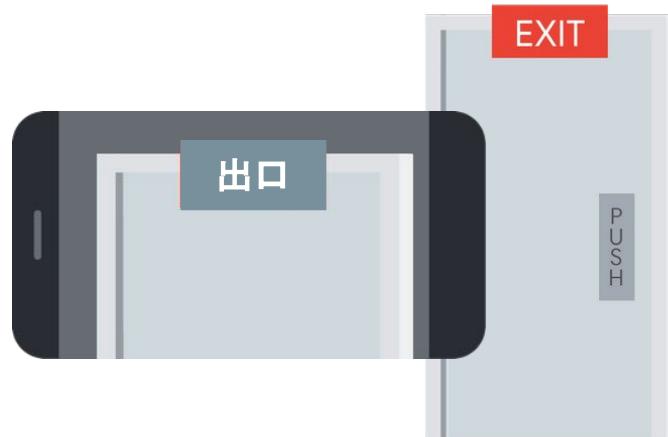
Module 04

Building Hybrid ML systems



Welcome back.

In this module you will learn how to



Take Google Translate, for example, which is composed of several models.

It uses one model to find a sign, another model to read the sign using optical character recognition, a third model to translate the sign, a fourth model to superimpose the translated text, and possibly even a fifth model to select the best font to use.

Add some intelligence

- Image and voice recognition
- Translation
- Natural language processing



ML allows you to add some "intelligence" to your mobile apps, such as image and voice recognition, translation and natural language processing.

Add some intelligence

- Image and voice recognition
- Translation
- Natural language processing
- Smarter analytics



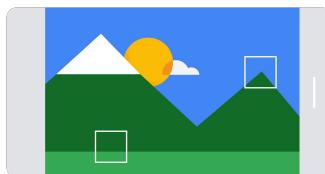
You can also apply machine learning to gain smarter analytics on mobile-specific data. For example, to detect certain patterns from motion sensor data, or GPS tracking data.

ML can extract meaning from **raw data**



This is all thanks to the fact that ML can extract meaning from raw data.

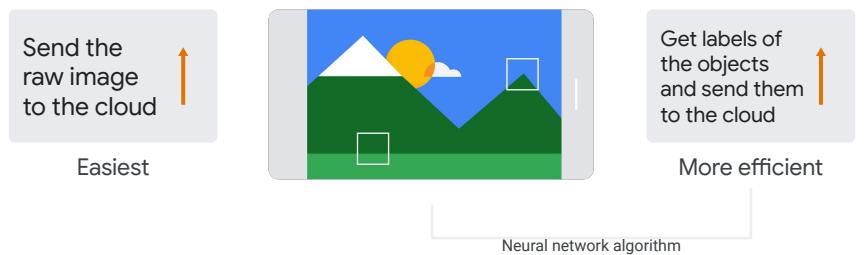
Send the
raw image
to the cloud



Easiest

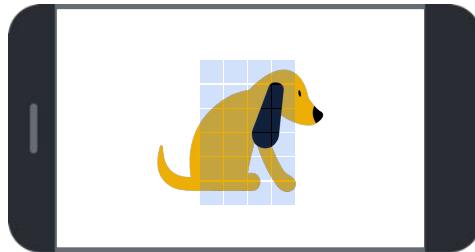


So, if you want to perform image recognition with your mobile app, the easiest way is to send the raw image to the cloud, and let the cloud service recognize the objects in the image.



However, if you have a neural network algorithm running on your mobile app, you can get labels of the objects and send them to the cloud. It's a more efficient way to collect the object labels on the cloud service.

Motion detection

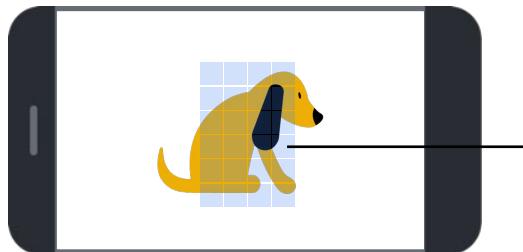


Run a neural network algorithm to extract a feature vector from the sensor data



Now let's say you perform motion detection with your mobile app. In this case, you can run a neural network algorithm to extract a feature vector from the sensor data.

Motion detection



Numbers in the feature vector represent the signatures of each motion

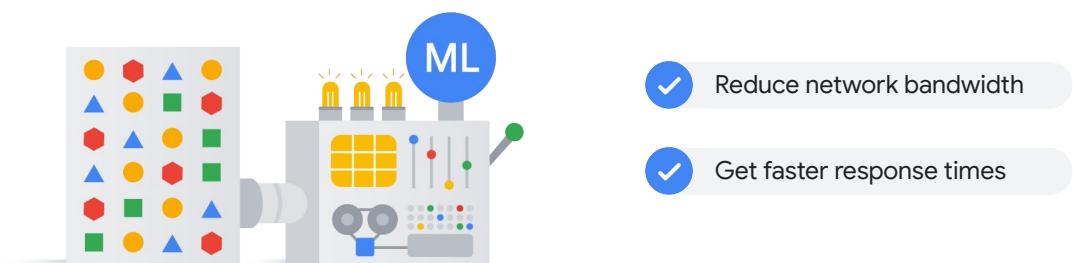
Send raw data

Run a neural network algorithm to extract a feature vector from the sensor data



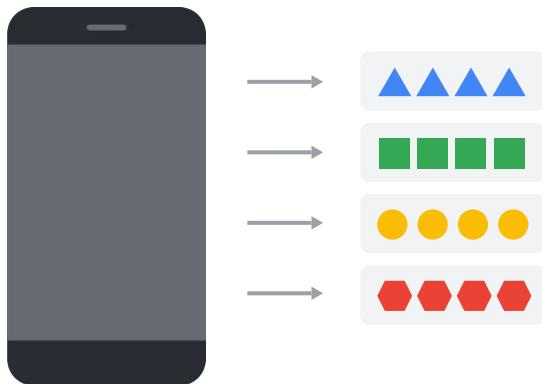
The numbers in the feature vector represent the "signatures" of each motion. This means you don't have to send the raw motion data to a cloud service.

By applying ML to mobile apps



Also, by applying machine learning to mobile apps, you can reduce network bandwidth and get faster response times when communicating with cloud services.

Microservices



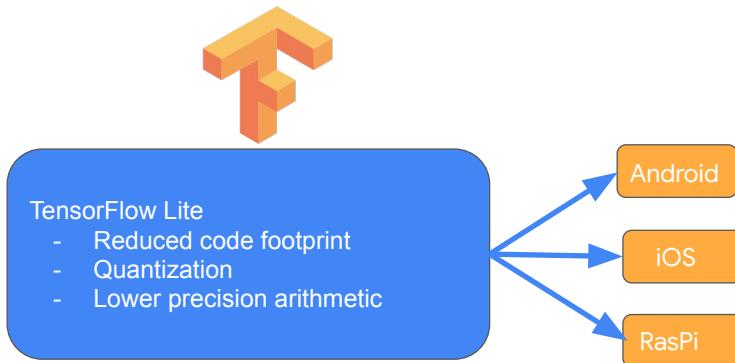
It's important to note that you often can't use the microservices approach for mobile devices, as they can add unwanted latency. Since you can't delegate to a microservice, like you can when running in the cloud, you'll now want a library, not a process.

Train data in the cloud → Do predictive modeling on a device



In these types of situations, it's best to train models in the cloud and do predictive modeling on a device. This means embedding the model within the device itself.

TensorFlow supports multiple mobile platforms



TF supports multiple mobile platforms, including Android, iOS and RasPi. In this talk, we focus on mobile devices.

Mobile TensorFlow makes sense when there is a poor or missing network connection, or where sending continuous data to a server would be too expensive. The purpose is to help developers make lean mobile apps using TensorFlow, both by continuing to reduce the code footprint, and by supporting [quantization](#) and [lower precision arithmetic](#) that reduce model size.

Build with Bazel by starting with a git clone

Install:

TensorFlow

Bazel

Android Studio (optional)

Android SDK

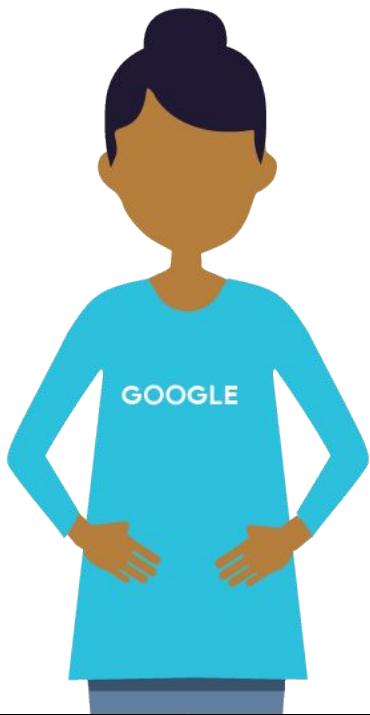
Android NDK

Config:

Edit tensorflow/WORKSPACE

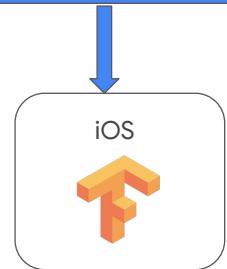
```
    android_sdk_repository(  
        name = "androidsdk",  
        api_level = 23,  
        build_tools_version = "25.0.2",  
        path =  
            "<path-to-android-sdk>",  
    )  
  
    android_ndk_repository(  
        Name = "androidndk",  
        Path = "<path-to-android-ndk>",  
        api_level=14  
    )
```

You can build a TensorFlow-shared object on Android from Android Studio using a continuous integration tool called Bazel.



Cocoapods support for iOS

```
CocoaPod  
Podfile  
target 'MyApp'  
pod  
'TensorFlow-experimental'
```



And for iOS, we added CocoaPod integration as well. It's quite simple.

Understand how to Code with the API

```
c.inferenceInterface =  
    new TensorFlowInferenceInterface(assetManager, modelFilename);  
  
// Copy the input data into TensorFlow.  
inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);  
  
// Run the inference call.  
inferenceInterface.run(outputNames, logStats);  
  
// Copy the output Tensor back into the output array.  
inferenceInterface.fetch(outputName, outputs);
```

Let's take a look how you can use the Tensorflow API.

The Android inference library integrates with Tensorflow for Java applications.

The library is a thin wrapper from Java to the Native implementation and the performance impact is minimal.

At first, create `TensorFlowInferenceInterface`, opening the model file from the asset in the APK.

Then, set up an input feed using `feed` API.

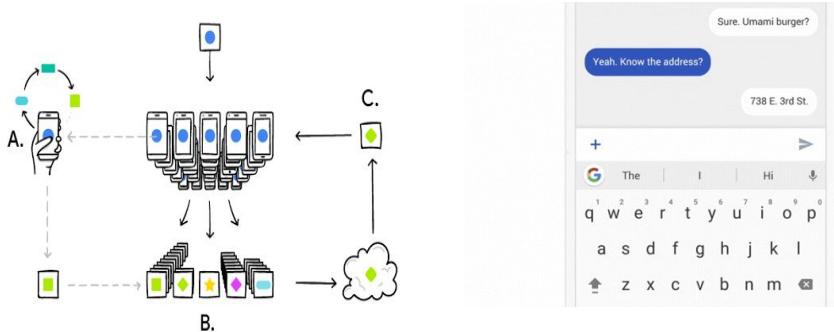
On mobile, the input data tends to be retrieved from various sensors, cameras etc.

Then run the inference,
and finally you can fetch the result using `fetch` method.

You would notice that those are all blocking calls.
So you would want to run them in a worker thread than the main

thread since an API call would take long time.

Even though we have talked primarily about prediction on mobile,
a new frontier is federated learning



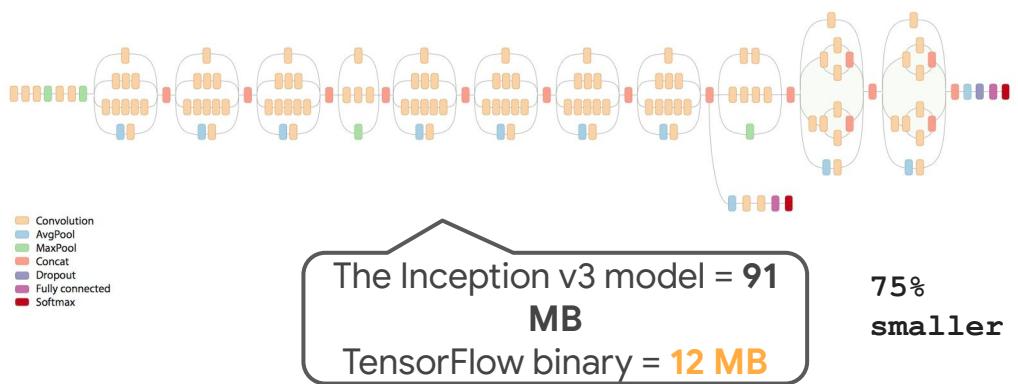
Federated learning in Google Keyboard

<https://research.googleblog.com/2017/04/federated-learning-collaborative.html>

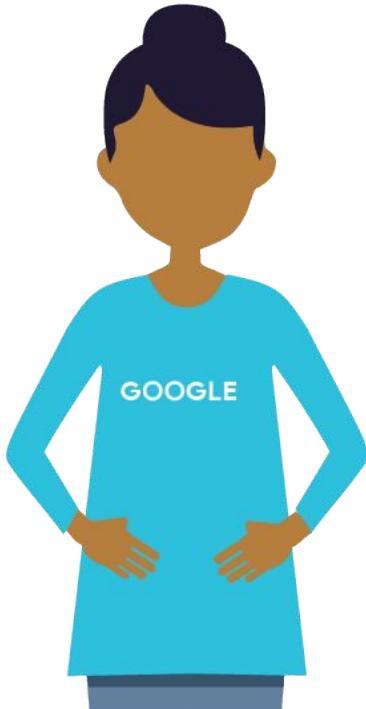
Even though we have talked primarily about prediction on mobile, a new frontier is federated learning.

The idea is to continuously train the model on device, and then combine model updates from a federation of users' devices to update the overall all. The goal is for each user to get a customized experience, and still retain their privacy.

Large neural networks can be compressed



Let's take a look of Inception v3 model. The model takes around 91MB in storage with a 25MM parameters. That would fit into a server or desktop machine but it is a bit huge for mobile.



There are several methods to reduce model size

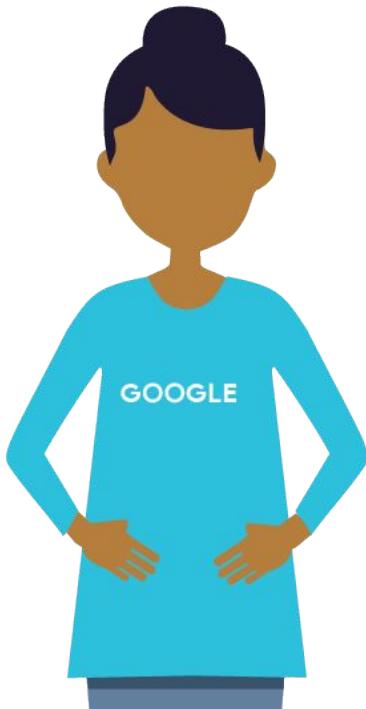


- Freeze graph
- Transform the graph
- Quantize weights and calculations

There are multiple techniques to optimize a graph, such as freeze graph, using graph transform tool, quantization, mem mapping etc.

Image (compress wallet) cc0:

<https://pixabay.com/en/credit-squeeze-taxation-purse-tax-522549/>



Freezing a graph can do load time optimization



Converts
variables to
constants and
removes
checkpoints

Freezing a graph is a load time optimization, which converts Variable nodes into Constant nodes.

In Tensorflow, Variable nodes are stored in different files whereas constants nodes are embedded in the graph itself (in the same file).

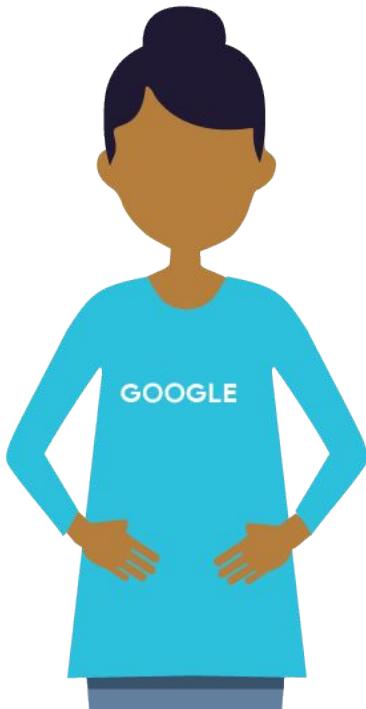
So, converting variable nodes into constant nodes would be a slight performance win in mobile and easier to handle, too.

You can use a python script doing it.

If you do this, though, you can't do federated learning, since there are no longer variables to train, just constants

Image cc0:

<https://pixabay.com/en/winter-frost-snow-snowdrifts-trees-2644107/>



Transform your graph to remove nodes you don't use
in prediction

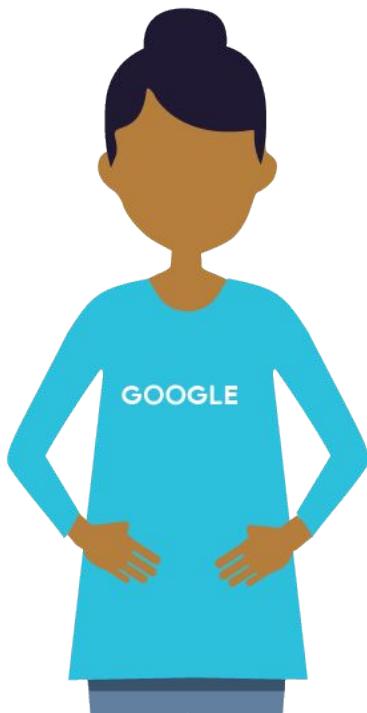


strip_unused_nodes:

Remove training-only
operations

The Graph Transform Tool that is part of the tensorflow distribution is your friend.

The tool supports various optimization tasks such as stripping nodes that are not used during inference, but were used in the learning phase. Nodes like gradient computation or batch norm can be removed during inference. The tool supports the removal of such training-only operations.



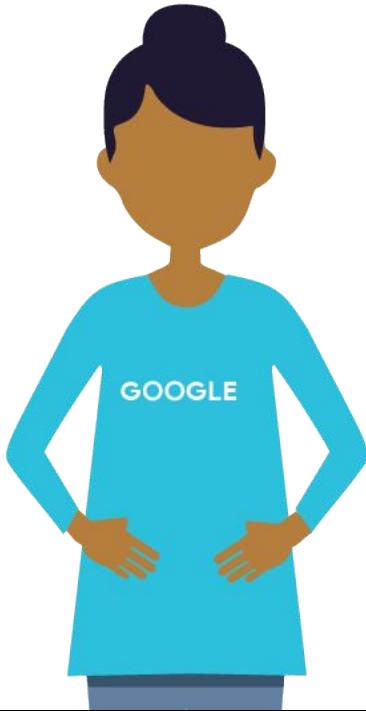
Transform your graph to remove nodes you don't use
in prediction



remove_nodes:

Remove debug
nodes

Obviously, debug nodes can also be removed.



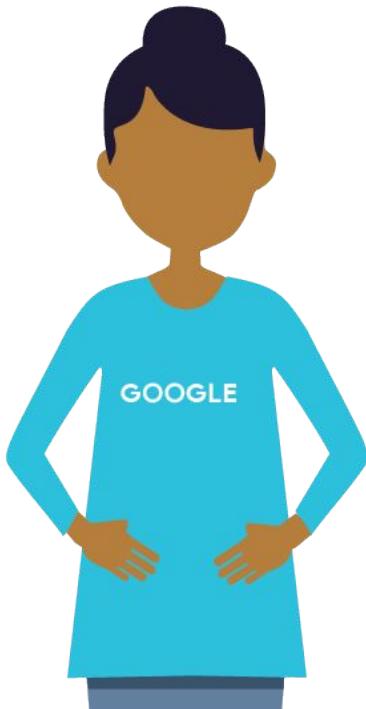
Transform your graph to remove nodes you don't use
in prediction



fold_batch_norms:

Remove Muls for
batch norm

What Fold_batch_norms does is that it converts Conv2D or MatMul ops followed by column-wise Muls into equivalent ops with the **Mul baked into the convolution weights**, to save computation during inference.



Transform your graph to remove nodes you don't use
in prediction



quantize_weights
quantize_nodes

Add quantization

Finally, if you want, the weights can be quantized to make the model more compressible.

You are trading off accuracy, however, when you do that. The question is how much accuracy are you trading off? You will have to measure, since this varies from model to model.

Image cc0

<https://pixabay.com/en/box-hedge-topiary-shears-gardener-869073/>

Image cc0

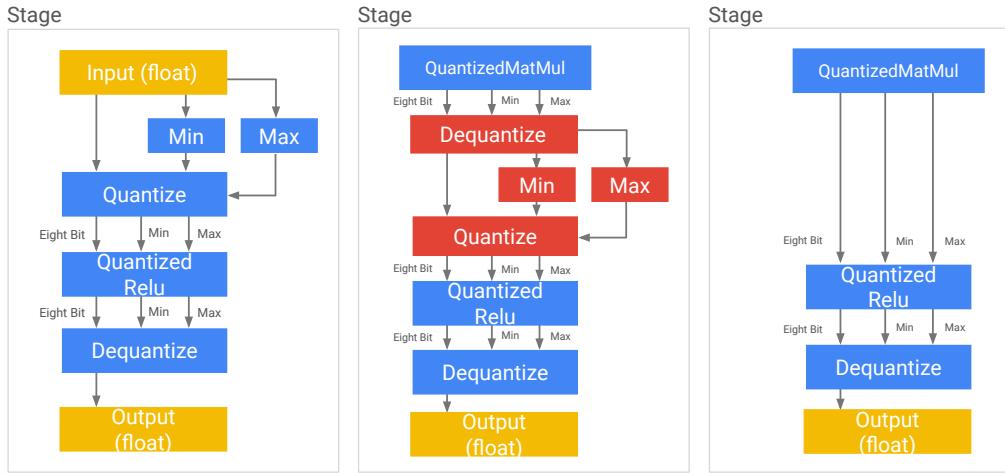
<https://pixabay.com/en/strip-bug-bug-macro-insect-red-812767/>

Image cc0 <https://pixabay.com/en/cream-puffs-delicious-427181/>

Image cc0

<https://pixabay.com/en/stress-tension-pressure-stressing-624220/>

Quantizing weights and calculations boosts performance



When modern neural networks were first developed, accuracy and speed were the prime concerns. As a result neural networks focused on 32 bit floating point arithmetic.

Now, researchers are deploying a lot of different models, especially in commercial applications. When we consider, the number of cycles needed for inference grows in proportion to the number of users, it is easy to see why the focus of neural networks has shifted to the efficiencies of inference.

To combat these inefficiencies, different techniques for storing numbers and performing calculations have been developed. These techniques are known as quantization.

Quantization compresses each float value to an 8 bit integer, which reduces the size of the files and the computational resources required to had the data.

This slide demonstrates quantization.

Stage 1:

The graph on the left shows a typical relu (rectified linear unit) operation with the internal conversion from float to 8 bit values. The min and max values are from the Input float tensor. Once the relu function is performed, the values are dequantized and output as floats.

Stage 2:

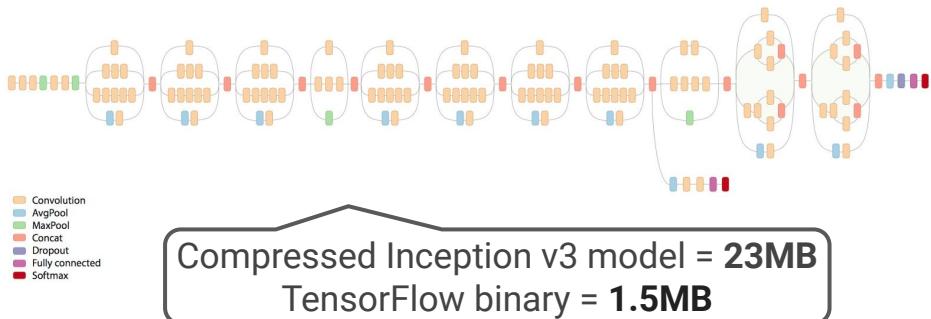
The middle graph shows the next stage in quantization, removing the unnecessary conversions to and from float. This stage identifies any patterns in the conversions performed in stage 1
And removes any redundancies.

Stage 3:

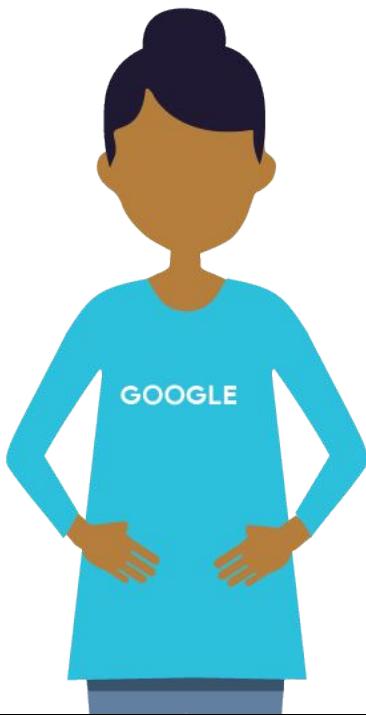
The final stage shows a graph where all the tensor calculations done in eight bit, there are no conversions to float necessary.

Reference: <https://www.tensorflow.org/performance/quantization>

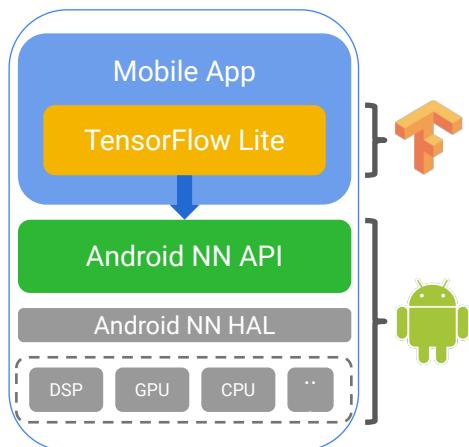
After these optimizations, the neural network is 75% smaller



With these optimizations, optimized graph of Inception V3 now becomes 23MB, which is 75% smaller now!!



TensorFlow Lite is optimized for mobile apps



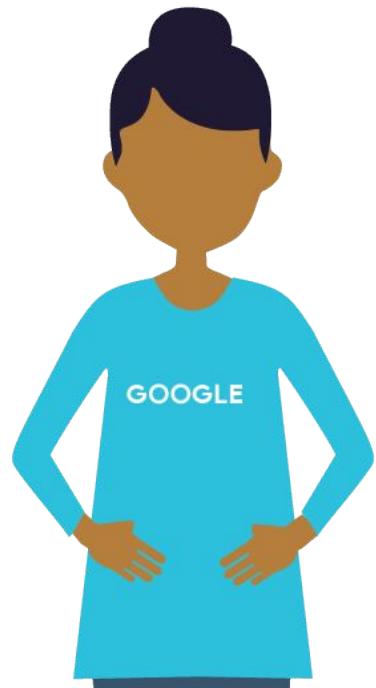
While you can do the freezing, quantization, etc. yourself, we recommend that you check out TensorFlow Lite which is a new TensorFlow runtime. TensorFlow Lite allows you to run TensorFlow models right on the device and leverages the Android NN API.

It is optimized for mobile apps.

Summary

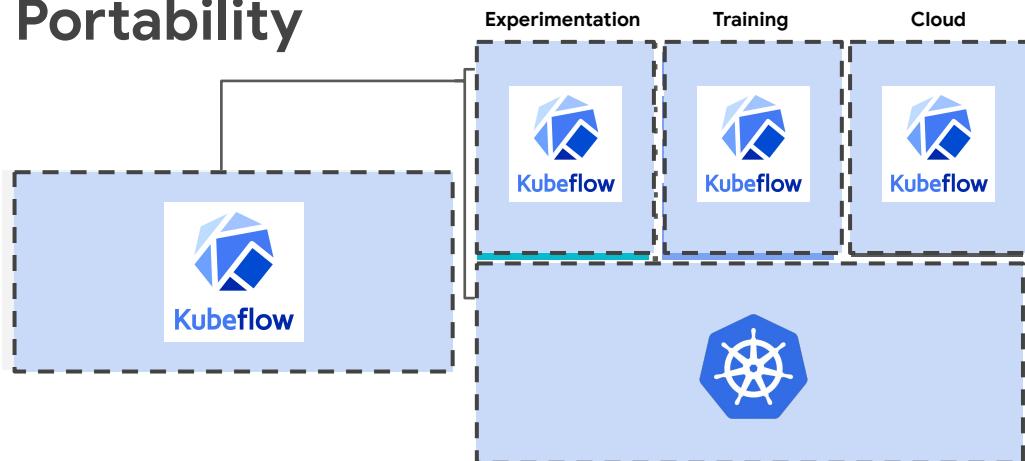
Build hybrid cloud machine learning models

Optimize TensorFlow graphs for mobile



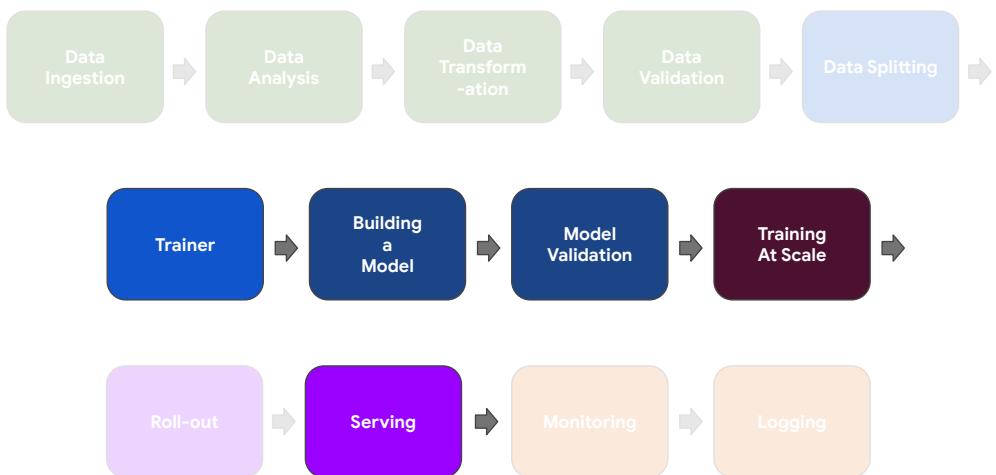
In this module, we showed you two technologies -- Kubeflow and TensorFlow Lite -- that are important in hybrid machine learning systems.

Portability



Kubeflow gives you composability, portability and scalability while preserving the ability to run everywhere.

What's in the box?

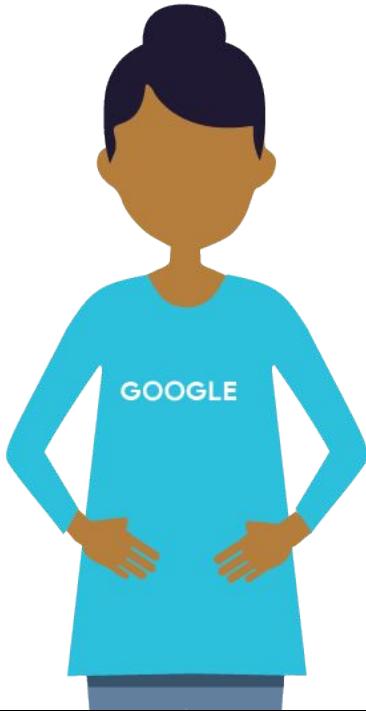


Specifically, Kubeflow offers portability and composability between your on-prem environment and Cloud ML Engine.

The tradeoff is that Kubeflow is not serverless. You will have to do cluster management.

Still, retaining the ability to move to cloud and serverless at some point in the future, or for some fraction of your workloads, provides flexibility.

The presence of Kubeflow also limits lockin. You can always take your models off Google Cloud, and you have a way to continue training and serving those models.



Freezing a graph can do load time optimization



Converts
variables to
constants and
removes
checkpoints

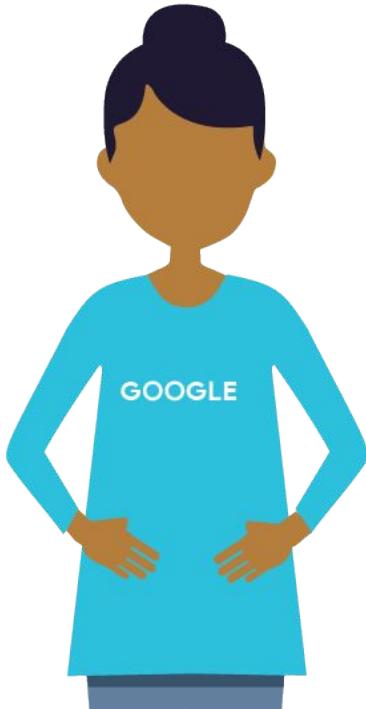
TensorFlow Lite makes specific compromises to enable ML inference on low-power devices.

For example, you can convert variable nodes into constant nodes, which streamlines your model because constant nodes are embedded in the graph itself.

However, you sacrifice maintainability and portability since you can not resume training from that model graph.

Image cc0:

<https://pixabay.com/en/winter-frost-snow-snowdrifts-trees-2644107/>



Transform your graph to remove nodes you don't use
in prediction



quantize_weights
quantize_nodes

Add quantization

Another compromise you might make is to use a less accurate model on device.
Perhaps you quantize the nodes
Or use a smaller model.

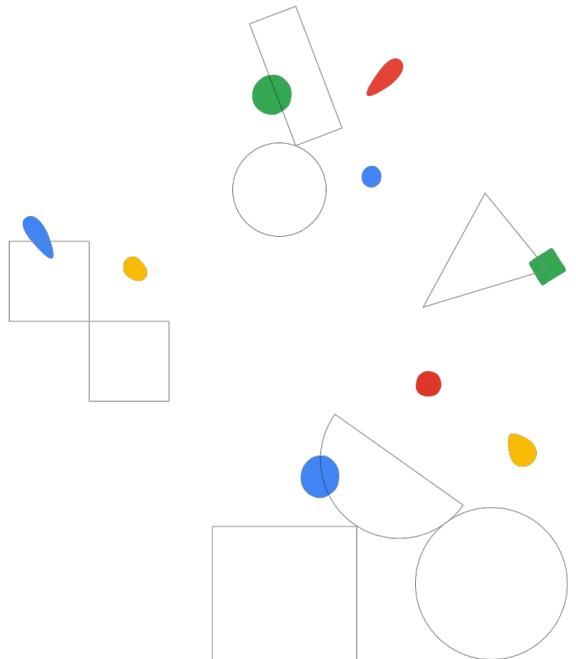
Of course, we hope that you choose to train and serve ML models on Google Cloud so that you don't have to make these compromises.
But if business and real-world considerations require you to be able to train or serve machine learning models outside a cloud environment, it is good to know that you do have options.

Kubeflow and TensorFlow Lite are good to know about. To have in your backpocket when such situations arise.



Course summary

Production ML systems



Let's do a quick recap





- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |





- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |





- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |





- | | |
|-----------|---------------------------------------|
| 01 | Architecting Production ML Systems |
| 02 | Designing Adaptable ML Systems |
| 03 | Designing High-Performance ML Systems |
| 04 | Building Hybrid ML Systems |



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud



Specialization

Advanced Machine Learning on Google Cloud

Production Machine Learning Systems

Image Processing and
Generation with Google Cloud

Sequence Models for Time Series
and Natural Language Processing

Recommendation Systems
with TensorFlow on Google Cloud

