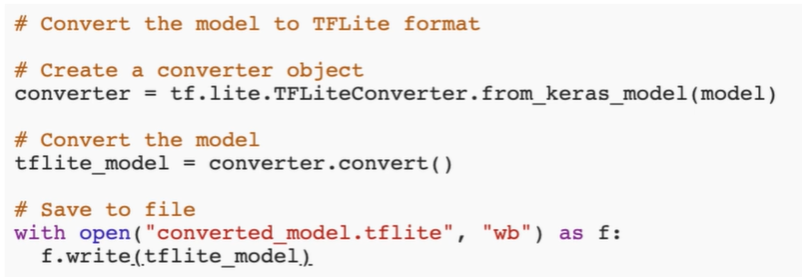
1. Web service (Tensorflow serving)

* E.g. Spam detection
* Application obvious
* If you provide an email service, you want to use spam detection
* How do we apply our spam network?
* Should we build our own version of Gmail or a desktop client?
* No! out of scope
* The average ML engineer probably will never have any clue how to build an email service
* Separation of responsibility
* Different teams – different responsibility
* If 2 different teams wrote 2 different applications, how can they communicate with each other?
* API
* Treat it like ‘a function I can call that will do some complicated work’
* Easy to use
* All I need to do is follow the instructions in the documentation
* E.g.: object recognition app
  + Create an API using Tensorflow serving
* Now you can use ML

1. Tensorflow serving (see notebook)
2. Tensorflow Lite (TFLite)

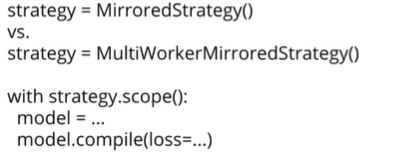
* A set of tools for those who want to use ML (in the form of Tensorflow) in mobile and embedded applications (ios, Android, Raspberry Pi, …)
* At a real company, mobile team will be a separate team
* High-level steps
* Train your models as we have already learned
  + Desktops / servers are more powerful than mobile devicees
  + So train first, then transfer the model (and weights) to the mobile device
  + Once on the device, model is used for prediction / inference
* Use Tensorflow Lite converter to convert the model to a .tflite file
  + Your model may have ben created using various APIs but let’s assume it’s Keras (the workflow remains the same)
  + .tflite file is just another ‘type of file’, like the .h5 file from model.save()
* On the mobile development side, the Tensorflow Lite library is used
  + Available for Java/C++ (Android), and Swift/Objective-C (iOS)
  + You use the Tensorflow Lite Interpreter to load in the .tflite file and make predictions with the loaded model
* Part of the mobile developer’s job will be to convert data from their own format into the right format for the Tensorflow Lite Interpreter
* E.g. they use the Android SDK to grab an image from the camera
* Tensorflow does not accept Image objects as input
* And there’s no numpy library for C++ and Java
* Mobile developer must convert image into an array of floats
* E.g. your model may be looking for pixel values in [0, 1] or [-1, +1]
* The 3 lines of code

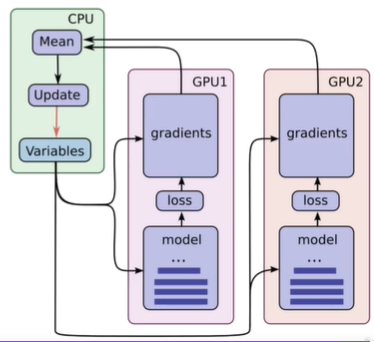


* Notebook: check out
* Big picture: TF Lite vs. TF Serving
* ML on device is an alternative to Tensorflow Serving (ML in the cloud)
* Using Serving, your device only makes API calls to the server
* Must be connected to the internet to get prediction
* TF Lit lets you store the model on the device
* Don’t need to use Internet
* When to use which -> up to you and your project’s constraints
  + E.g. work on the subway -> cloud is not a good choice
  + E.g. model is very large and requires heavy computation -> mobile is not a good choice
* Cut out the ML engineer
* For certain applications, a mobile developer may not have any need for a ML engineer
* Tensorflow has already published pretrained models
* E.g. object detection, pose estimation, smart reply

1. Why is Google the King of Distributed computing?

* Why distributed computing?
* Large datasets
* E.g. count how many lines of a file contain the word ‘lazy’
* But what if the file is 10GB? 100GB? 1000GB?
* Imagine a more serious application, such as calculating your company’s revenue each day
* Solution?
* Suppose you have 10 computers
* Split the data into 10 batches for each computer to process a batch
* Reduce computation time by a factor of 10
* Google’s solution
* Google came up with a solution a long time ago: MapReduce
* Distribute work across 100s or 1000s of workers, then combine results
* Since then, Google has invented many distributed technologies
  + BigTable – distributed NoSQL database
  + GFS – Google File System (sharding, replication)
  + Spanner – global NewSQL database
* Neural network training
* Distributed training today
* Thanks to the Tensorflow 2.0 API
* Access to multiple methods of distributed training – distribution strategies
* Available: MirroredStrategy
* Experimental:
  + MultiWorkerMirroredStrategy
  + CentralStorageStrategy
* The beauty of Tensorflow 2.0 is you don’t have to worry about how these work – just plug and play
* Even with just one machine with one GPU, you can still take advantage
* You can also have one machine + multiple GPUs
* MirroredStrategy
* Data parallelism
* E.g. you have 1000 training samples and 10 GPUs
* Send 100 samples to each GPU
* Similar to our low-tech approach
* Multiple workers
* Multiple workers each with multiple GPUs? Also possible





* You need to set the TF\_CONFIG env variable and actually setup each of the machines

1. Training with Distributed Strategies

* More than one GPU?
* Minimal code changes and minimal configuration
  + Don’t go for 1000 workers with 100 TPUs each just yet
* Try what is easy before trying what is hard
* Not: Google Colab is free >< other services are not
* Popular option: AWS (usually you want Px instance)
  + P3 instance: 96 Intel Xeon CPUs, 8 NVIDIA V100 Tensor Core GPUs with 32GB of RAM each
* Use AMIs (Amazon Machine Instance)
  + Choose “deep learning AMI”
* Everything comes pre-installed (except you may have to install TF2.0 yourself)

1. Using the TPU (check the notebook and run on Google Colab)