Kubernetes

Initially developed by Google, Kubernetes is a powerful open-source system for running and coordinating containerized applications and services across a cluster of machines using methods that provide predictability, scalability, and high availability.

It’s main advantage is the freedom of defining how an application should run and interact with other application or the outside world.

Kubernetes Architecture

Kubernetes can be visualized as a system built in layers, with each higher layer abstracting the complexity found in the lower levels.It brings together individual physical or virtual machines into a cluster using a shared network to communicate between each server. This cluster is the physical platform where all Kubernetes components, capabilities, and workloads are configured.

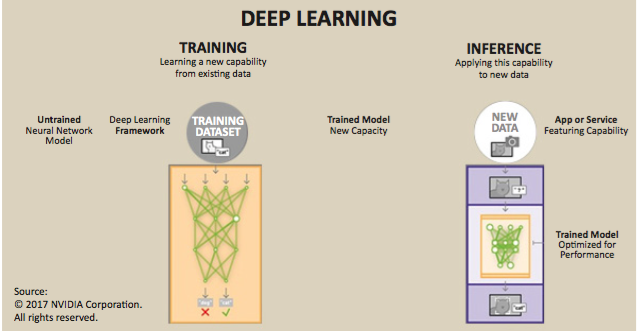
Each machine in the cluster are given a role within the system, one server functions as the master server. This server acts as a gateway and brain for the cluster by exposing an API for users and clients, checking other servers and deciding how best to split up and assign work (known as "scheduling"), and orchestrating communication between other components. The master server acts as the primary point of contact with the cluster and is responsible for most of the centralized logic Kubernetes provides.

The other machines in the cluster are designed as nodes:servers responsible for accepting and running workloads using local and external resources. The environments where the applications and services run in are called containers. Each node eceives work instructions from the master server and creates or destroys containers accordingly, adjusting networking rules to route and forward traffic appropriately.

Machine learning is well-suited for problems like making predictions (classification, regression) and knowledge discovery. Deep learning is an area of machine learning which benefits of some algorithms that allows the software to train itself in order to later be able to take decisions later with new data. It performs tasks like speech and image recognition,

Making use of the capabilities of multilayered neural networks to a wide range amount of data.

Deep Learning

Deep learning attempts to learn multiple levels of features of large data sets with multi-layer neural networks and make predictive decisions for the new data. This indicates two phases in deep learning: first, the neural network is “trained” with a large number of input data; second, the trained neural network is used for “inference” designed to make predictions with new data. Due to the large number of parameters and training set size, the training phase requires tremendous amounts of computation power.

However, training a neural require powerful CPU and takes time, and running on a cloud might not be ported to a mobile device due to their limitations on computing and battery power.

Fortunately, there is a difference between training a neural network, ‘offline’ and the trained model that can recognise new objects in real time, known as ‘inference’.

So when power and battery life are a concern, as they are for mobile devices, anyone creating a neural network application will want to train it in such a way as to optimise the inference stage as much as possible. This is done by reducing the computational complexity and bandwidth and as a result, save power.

In essence, optimising the network is done by shrinking down the trained neural network to make it easier to run by reducing redundancies. It’s similar in concept to how a digital image is commonly shrunk down from its uncompressed state to a space-efficient format such as a JPG. If the compressing algorithm is good, this can be done with very little appreciable difference in image quality. In neural network terms, image quality equates to inferencing accuracy. With suitable training, an optimised neural network can be greatly reduced in terms of size and complexity, all the while ensuring that the accuracy of the inferencing remains high.

The first step in obtaining the desired results is choosing the suitable model for the task.

Elimination & reduction

First, you want to reduce the number of operations inside your network, and then you want to further reduce the compute cost of the operations that remain.

In any given network there are two types of data flowing through the network: the weights( coefficients) and the activation data that’s being processed at any time. Reducing these two variables as much as possible cause those paths to be eliminated from the convulsion, therefore it will be able to run faster.( similar to removing the redundant information in the image)

Pruning the network

The aim of pruning the network is to increase the level of sparsity in the weights, removing connections and setting weight to zero. A function is used to decide wether or not a weight singnificantly improves accuracy, otherwise, it removes that weight from the network.

Distillation & regularisation

The process of transferring the knowledge from one network to another is called distillation.We can also add regularisation terms, a key tool in Deep Neural network (DNN) training, which adds a constraint that the trained network has a preference for zeros.

The last but not least method for reducing the cost of the computation is done by reducing the number of bits per weight.

We start off with weights at a full precision 32-bit float. We can set a target bit-depth and rerun the training, applying quantisation to a randomly selected subset of the weights. This means that over time we are pushing the weights to quantised values and allowing the network to adapt to this. Eventually, we can clamp the weights to their nearest quantised values.

Choosing to target bit-depths lower than 32-bits will provide a significant reduction in your energy per inference, especially if you’re able to go down to as low as 4-bit, should your chosen hardware platform support it.

Looking at the quantisation of the weights from 32-bit sweeping down to 5-bit, we can see you can drop to as low as 8-bit without much loss of performance without having to retrain, but to move below this, which is what you’ll want for an embedded device, retraining provides great benefits.

Optimising neural networks for efficient inferencing is not a trivial task, but can provide significant benefits, in terms of bandwidth reduction and therefore power consumption.

Convultional neural Networks approach in Tenserflow

The common classification problem is classifying an RGB 32X32 pixel image among 10 basic categories. Tenserflow proposes a model

Model Architecture

Model Inputs

Model Prediction

Model Training

Launching and Training the Model

Evaluating a model

Placing the result on devices

Art Classifier Flow Steps

* Google BigQuery
* selecting the attributes and creating labeled dataset
* downloading the data
* choosing a model for image classification
* Converting the data to TFRecord format
* Creating the Tenserflow container image
* Deploying and Running the training on Kubernetes (GPU??)
* Evaluating the accuracy of the trained model
* Saving the trained model and logs
* TensorBoard for statistics and visualisation of the training
* Loading the trained model in Kubernetes and running inference on a new input for classification