TensorFlow

-Ranjith MS

Why deep learning frameworks??...

- Easy to code
- Coding reduces hence easy to debug
- Functions are pre-built and are optimized

E.g: Simple 3x3 matrix multiplication in pytorch is 50,000 times faster than pure python code

Tensorflow

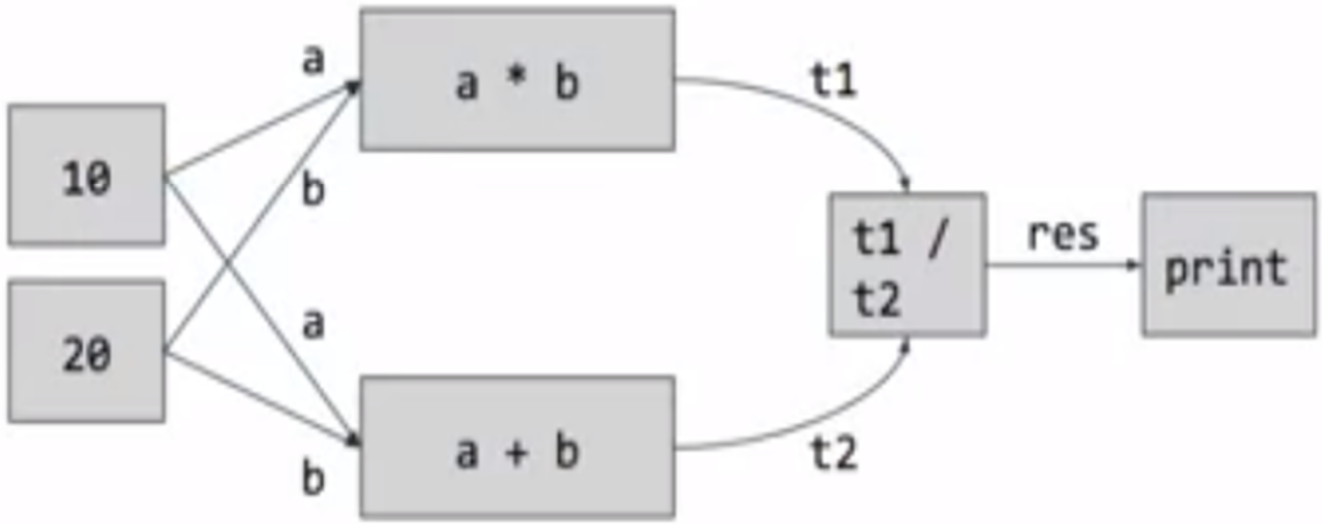
- It is being developed and maintained by Google
- TensorFlow is a free and open-source
- It's **written** in a combination of highly-optimized C++ and CUDA (Nvidia's **language** for programming GPUs)
- A library for defining computation graph and a runtime to execute such graph in different hardware

TensorFlow Computational Graphs

```
a=tf.constant(10)
b=tf.constant(20)

t1=tf.multiply(a,b)
t2=tf.add(a,b)
res=tf.divide(t1,t2)

tf.Print(res)
```



the data dependencies specify the order of execution,
operations that do not depend on each other can schedule in parallel

Steps involved in creating a TensorFlow model are

- Creating a placeholder
- Initializing the parameters
- Forward propagation
- Computing the cost
- Creating an optimizer

Model for digit recognition

- Data set : MNIST

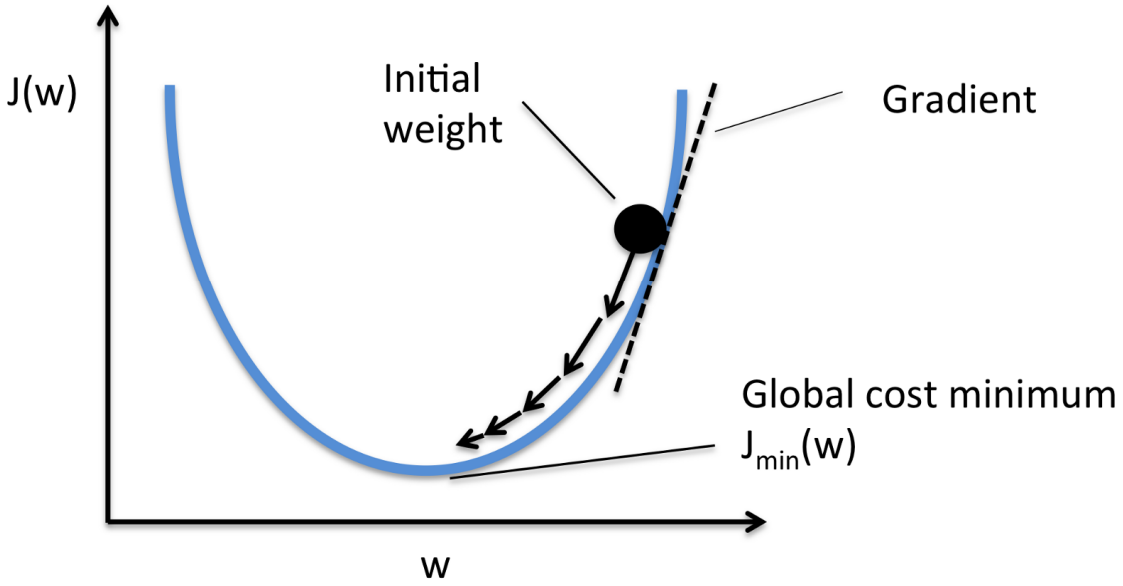
Gradient Descent

Batch Gradient descent

- It considers all the examples in training set to take one step

Mini batch gradient descent

- It considers batch of examples from training set to take each step



Repeat until convergence {

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

1. Creating a placeholder

- TensorFlow requires that you create placeholders for the input data that will be fed into the model when running the session
- we could use "None" as the batch size, it will give us the flexibility to choose it later

```
def create_placeholder(n,n_y):
  X=tf.placeholder(tf.float32,shape=(None,n),name='X')
  Y=tf.placeholder(tf.float32,shape=(None,n_y),name='Y')
  return X,Y
```

One Hot encoding

ranjithms523@gmail.com

2. Initialize the parameters

```
def initialize_parameters():

  W=tf.get_variable("W",[784,10],initializer=tf.contrib.layers.xavier_initializer())
  b=tf.get_variable("b",[1,10],initializer=tf.contrib.layers.xavier_initializer())
  parameters={'W':W,
              'b':b}
  return parameters
```

4. Forward Propagation

```
def forward_propagate(X,parameters):
    W=parameters['W']
    b=parameters['b']
    logits = tf.matmul(X, W) + b
    return logits
```

5. Loss Function

Softmax

```
def compute_cost(Z,Y):
  cost=tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=Z,labels=Y))
  return cost
```