TensorFlow and Keras

statinfer.com

Contents

Limitations of Machine Learning tools

- Python, R and SAS work really well for solving the predictive modelling and machine learning problems
- The libraries like “sklearn” are sufficient for building regression models, trees, random forest and bosting models.
- But these tools have limited deep neural networks libraries
- What are the tools/frameworks for deep learning algorithms?

Deep Learning Frameworks

- TensorFlow(by Google)
- Torch(by Facebook)
- Caffe(by UC Berkeley)
- Theano(Old version of TensorFlow)
- MxNet(by Amazon)
- CNTK(by Microsoft)
- Paddle(by Baidu)

statinfer

TensorFlow

- TensorFlow was developed by the Google Brain team for internal use.
- It was released under the Apache 2.0 open source license on November 9, 2015

DEC 1, 2015 @ 01:34 PM 4,801

The Little Black Book of I

Reasons Why Google's Latest AI-TensorFlow is Open Sourced

CADE METZ BUSINESS 11.09.15 09:00 AM

GOOGLE JUST OPEN SOURCED TENSORFLOW, ITS ARTIFICIAL INTELLIGENCE ENGINE

TensorFlow

- Most popular among all Deep learning frameworks.
- TensorFlow works really well with matrix computations - All the deep learning algorithms are highly calculation intensive.
- Scalable to multi-CPUs and even GPUs
- Can handle almost all type of deep networks, be it ANN or CNN or RNNs

Working with TensorFlow

- Has Python API and python is very easy install and to work on.
- We can use numpy to build all the models from scratch. But TensorFlow does it better by providing function to do it easily.
- TensorFlow has one of the best documentation and great community support as of now.

Some Key Terms in TensorFlow

- TensorFlow represents data as tensors
- Represents the calculations as computational graphs
- Place holders are created and later filled with data
- Sessions will be used for executing graphs.

Some key terms

- Tensors
- Dimensions
- Computational graphs
- Nodes and Edges
- Placeholders
- feed_dict
- Session

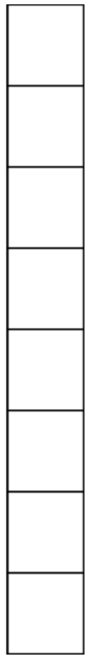
Tensor

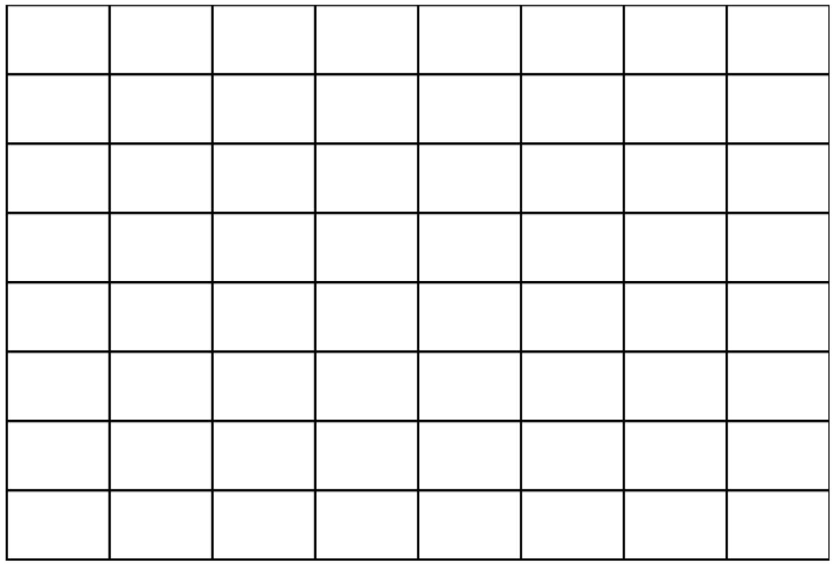
Tensor means data

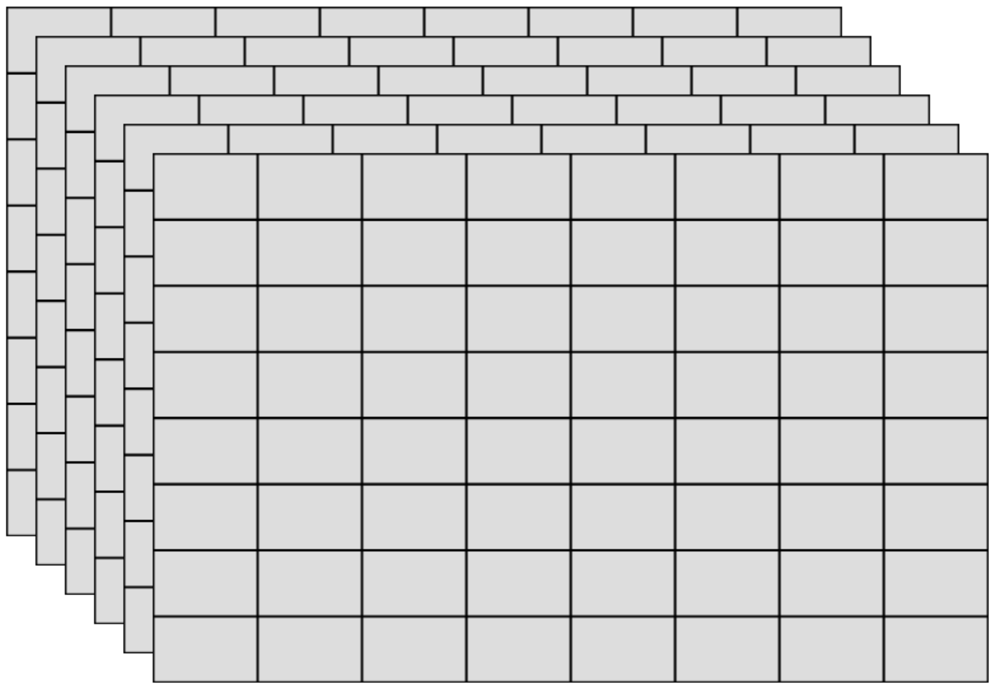
A vector/array is a collection of elements (scalars)

A matrix is a two dimensional vector

A Tensor is a multi-dimensional vector.







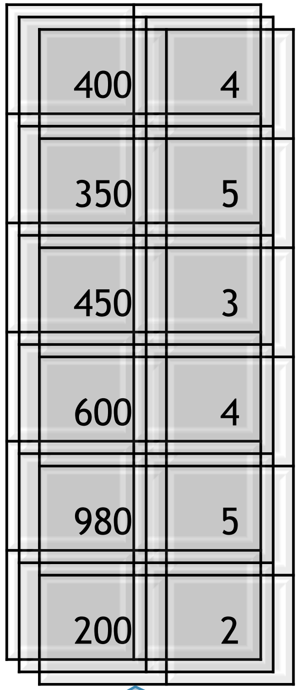
Tensor

400
350
450
600
980
200

A vector indicating the product price(say smart phones)
A tensor of dimension [6]

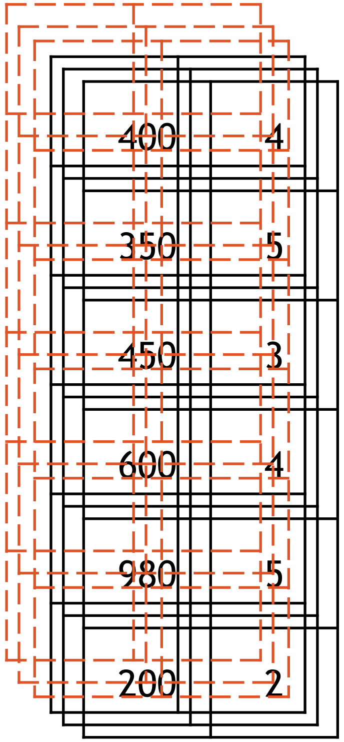
A matrix indicating the product price and rating
A tensor of dimension [6,2]

400	4
350	5
450	3
600	4
980	5
200	2



A tensor indicating the product price and rating for last three years
A tensor of dimension [6,2,3]

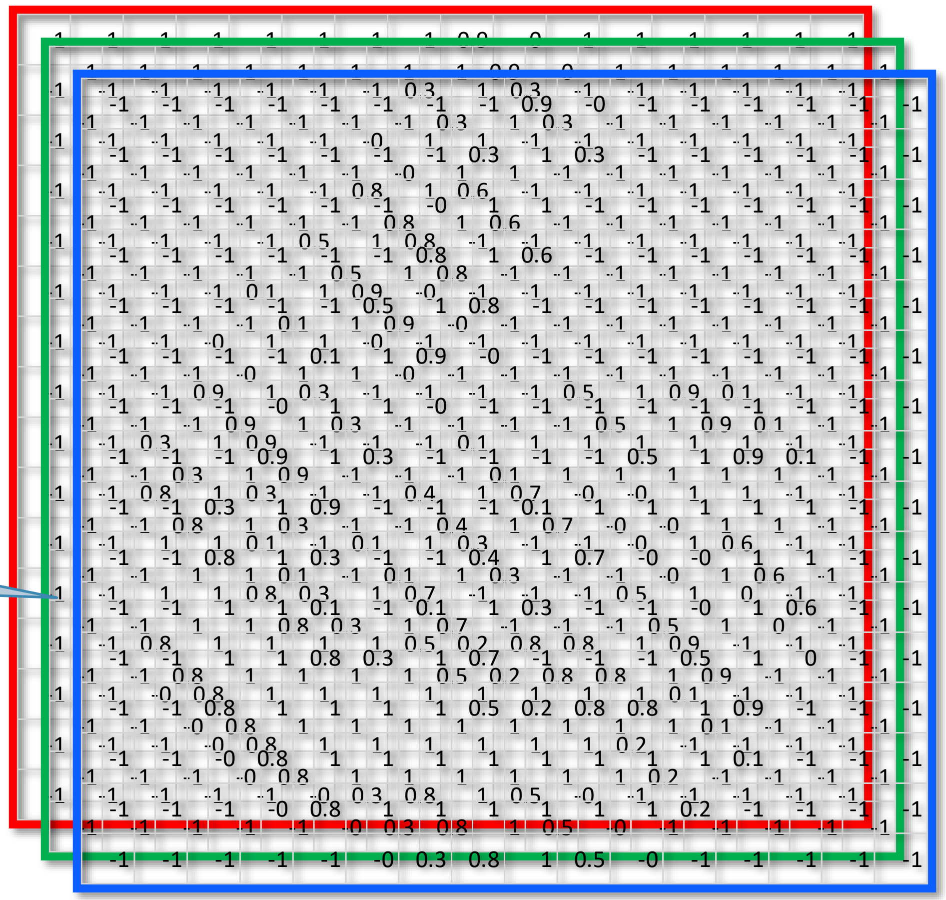
A tensor indicating the product price and rating for last three years in two countries
A tensor of dimension [6,2,3,2]



Tensor for images

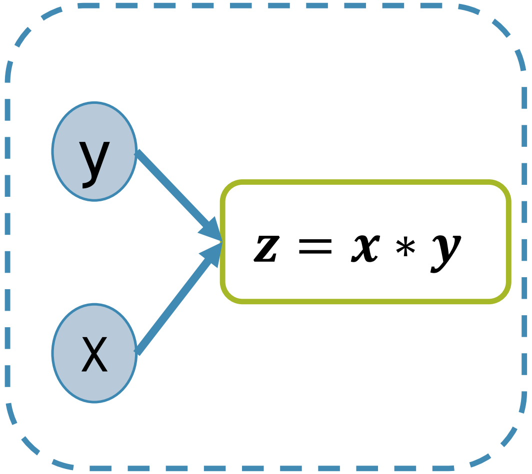
- A colour image is represented as a three dimensional tensor
- [Width, Height, Colour]
- The colour component depth 3, Red, Green and Blue

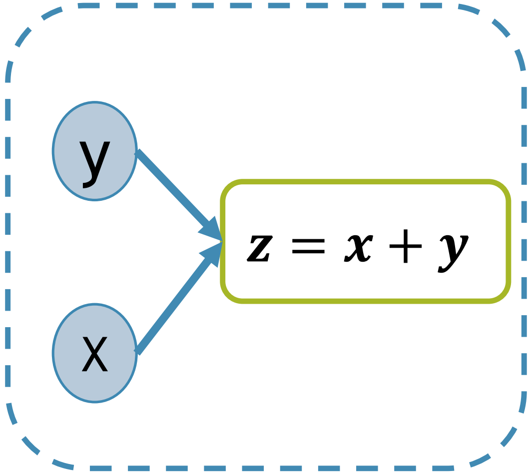
16X16 pixels colour image
A tensor with dimensions
[16,16,3]



Computational graphs

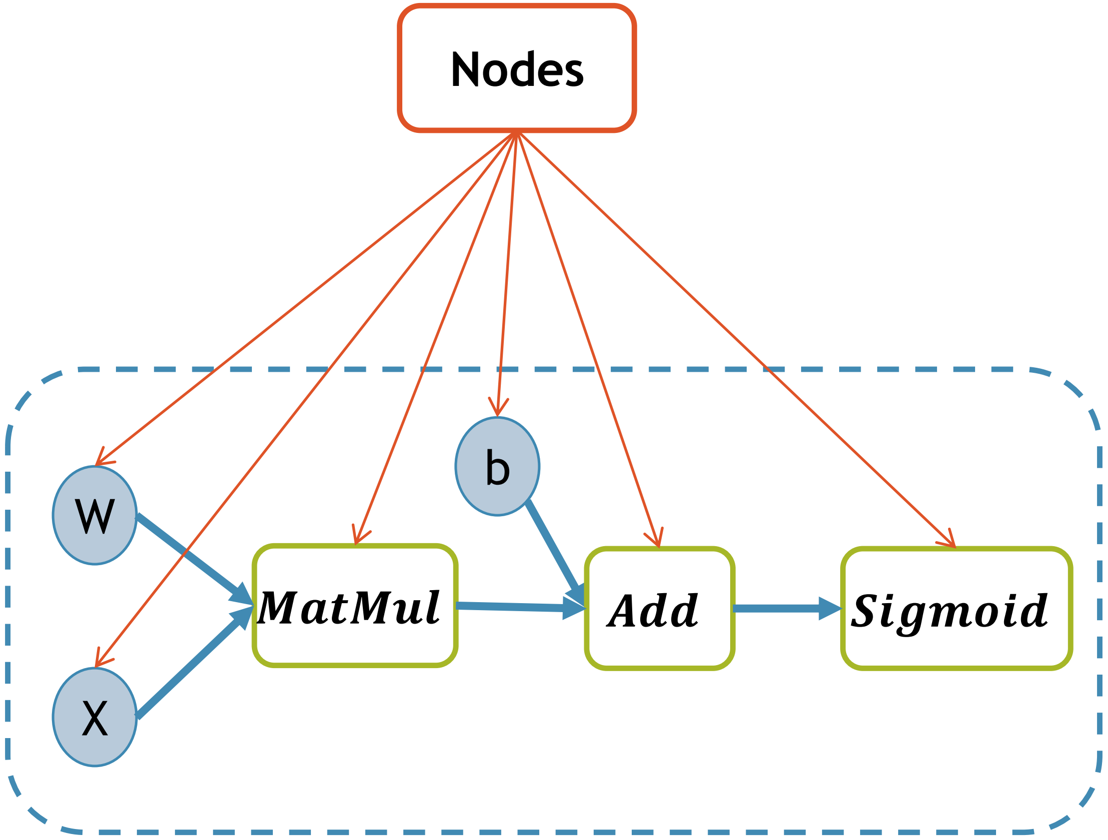
- Inside TensorFlow computations are represented using computational graphs
- You can call it as data flow graph. As sequence of operations on tensors(data)
- It has nodes and edges





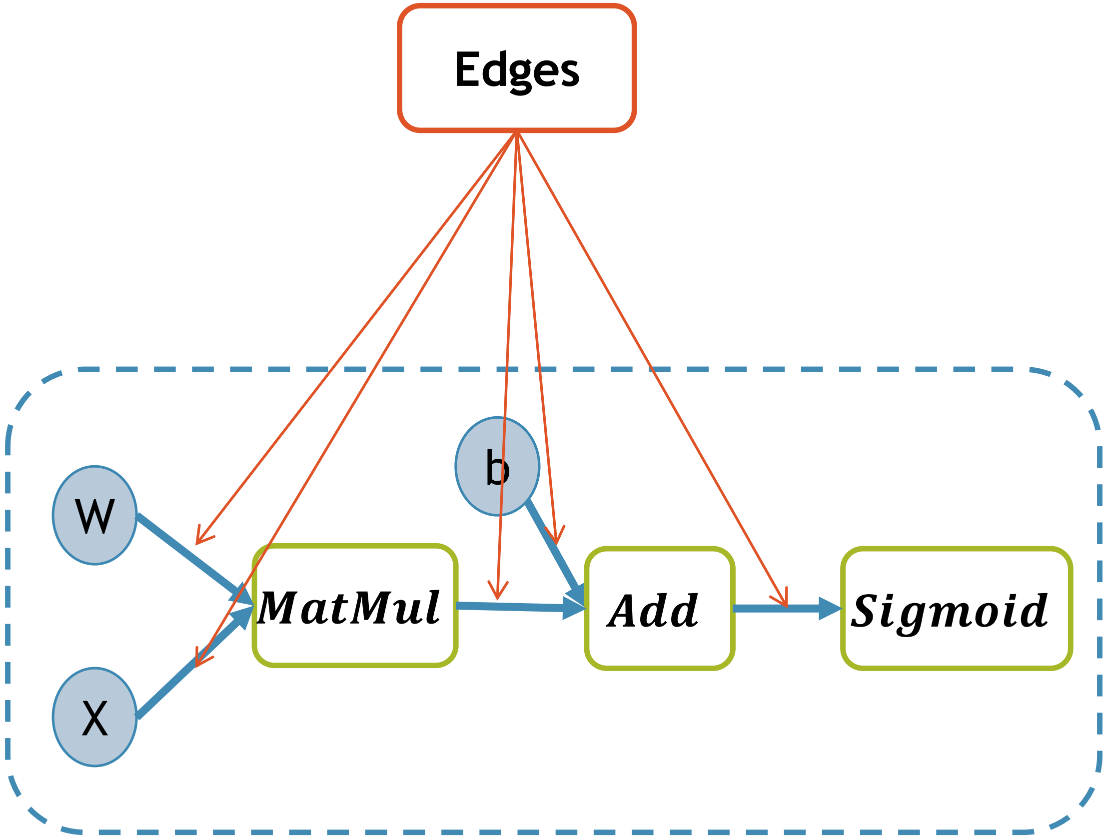
Computation Graphs-Nodes

- **Nodes**
  1. Data and Operations
  2. Operations which have any number of inputs and outputs.
  3. Variables/Tensors are also represented by nodes

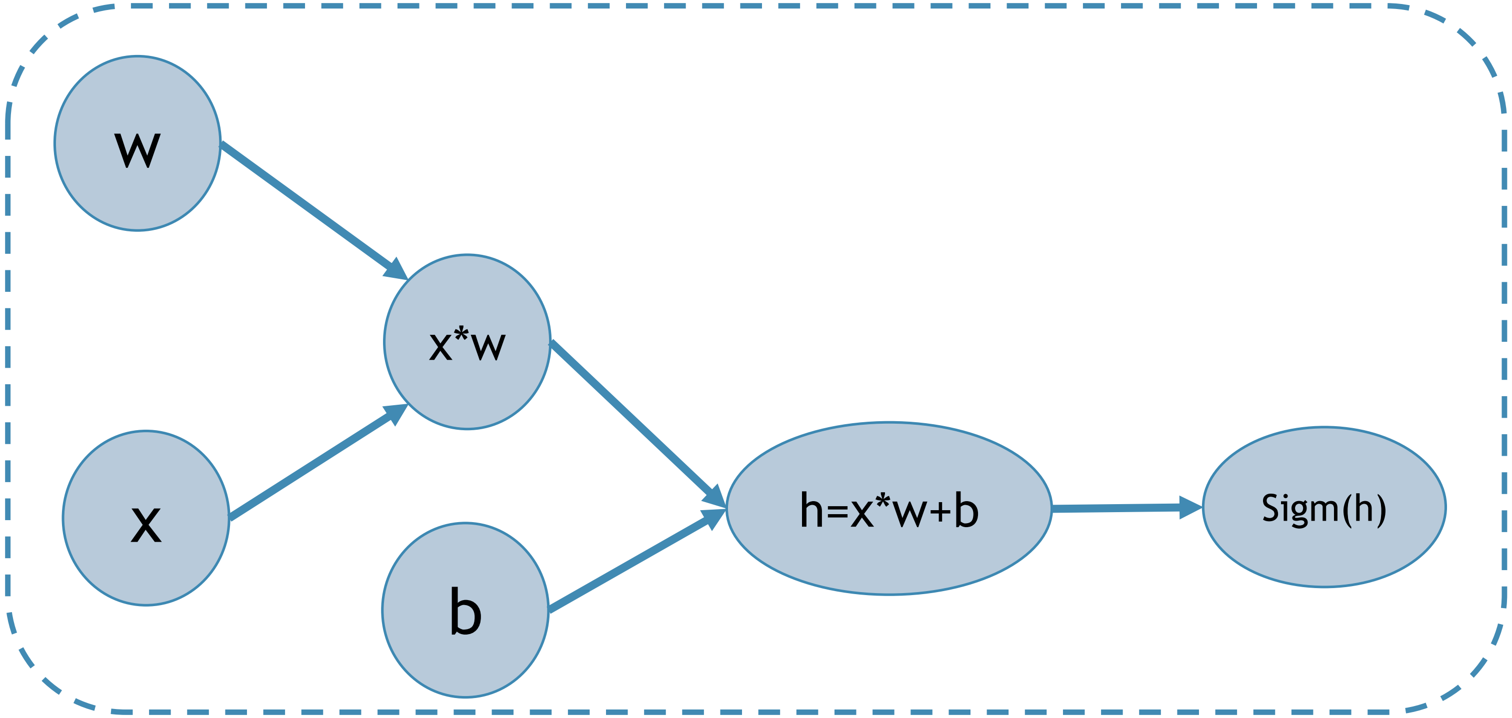


Computation Graphs- Edges

- **Edges :**
  - Data flow direction
  - Flow of tensors between nodes



Computation Graphs



- Computational graphs are particularly useful if the operations are complex
- TensorFlow computations define a computation graph that has no numerical value until it is evaluated!

Why Computation Graphs?

- Saves time by independently running the subgraphs that contribute to the final computation
- While training deep learning models, partial derivatives and chain rule applications are handled efficiently using these computational graphs.
- Break computation into small, differential pieces to facilitates auto-differentiation.
- Facilitate distributed computation, spread the work across multiple CPUs, GPUs, or devices.

Programming in TensorFlow

- Express the overall task as a computational graph
- A graph no numerical value until it is executed.
- Execute the graph along with the data in a session
- What is a session?

Building the Graph as an abstract

Running the graph in a Session

Session

- A session is used for launching the computational graph
- Session feeds Data into Graph
- A Session object summarizes the environment in which the data is fed into the graph
  - **Graph:** Abstract **without Data**
  - **Data:** Input and Labels.
  - **Session**: Launch the graph, feed the data to proceed with the calculations

Launch the graph in a session

Code: Programming in TensorFlow

statinfer

- TensorFlow does **NOT** execute any computation until the session is created and the run function is applied. - What does this mean?
- This below code will not return any computations.

```
In [1]: import tensorflow as tf

        a = tf.constant(20)
        b = tf.constant(5)

        c = a*b

        print(c)

        Tensor("mul:0", shape=(), dtype=int32)
```

No calculations are running, only an abstract of operation was created

- We need to run it as a whole using something called session.

statinfer

Code : Programming in TensorFlow

- We need to run our abstract graph by creating a session running it and then closing it.

```
In [2]: import tensorflow as tf

        a = tf.constant(20)
        b = tf.constant(5)

        c = a*b

        sess = tf.Session()
        print(sess.run([c]))
        sess.close()

        [100]
```

We are able to get the solution after launching graph(c) in a session

Code Programming in TensorFlow

```
a = tf.constant([[20,10],[1,64]])
b = tf.constant([[5,10],[1,64]])

c = a*b

sess=tf.Session()
print("a is " , a)
print("a in a session is \n ", sess.run([a]))
print("b is " , b)
print("b in a session is \n", sess.run([b]))
print("c is " , c)
print("c in a session is \n", sess.run([c]))
```

```
a is  Tensor("Const_26:0", shape=(2, 2), dtype=int32)
a in a session is
  [array([[20, 10],
       [ 1, 64]])]
b is  Tensor("Const_27:0", shape=(2, 2), dtype=int32)
b in a session is
 [array([[ 5, 10],
       [ 1, 64]])]
c is  Tensor("mul_13:0", shape=(2, 2), dtype=int32)
c in a session is
 [array([[ 100,  100],
       [   1, 4096]])]
```

Just printing gives abstract/size/type of object

Printing object through session gives calculated value of the object

Variables

- Constructs that allow us to change stored value and work as trainable parameters.
- To declare a variable, we create an instance of the class **tf.Variable()** with the **type and initial value**.

- We have to initialize variables before using them. Else we'll run into **FailedPreconditionError: Attempting to use uninitialized value tensor**

Variables : Defining, Initializing and eval

- Define using **tf.Variable()**
- Initialize using **.initializer**
  - Most practices way: initializing all variables in one go using:
  - **tf.global_variables_initializer()**
- Get the value using **.eval()** in a session.

Different Parameters available while declaring Variable

```
tf.Variable(initial_value=None, trainable=True, collections=None, validate_shape=True, caching_device=None, name=None, variabl
e_def=None, dtype=None, expected_shape=None, import_scope=None)

x=tf.Variable(...)
x.initializer # init
```

Constructing and Initializing a particular Variable

Code: Programming in TF: Variables

```
In [27]: import tensorflow as tf
         a=tf.Variable(10,name="scalar") #A variable with scalar value

         b=tf.Variable([11,32],name="vector") #A variable as a vector

         c=tf.Variable([[1,9],[8,5]],name="matrix") #A variable as a 2*2 matrix

         w=tf.Variable(tf.zeros([784,255])) #A variable 784*255 with zeros values

         init =tf.global_variables_initializer() #Global Initializer
         with tf.Session() as sess:
             sess.run(init) #Initializing all the variables
             print(a,"\n",a.eval())
             print(b,"\n",b.eval())
             print(c,"\n",c.eval())
             print(w,"\n",w.eval())
```

```
<tf.Variable 'scalar_16:0' shape=() dtype=int32_ref>
 10
<tf.Variable 'vector_9:0' shape=(2,) dtype=int32_ref>
 [11 32]
<tf.Variable 'matrix:0' shape=(2, 2) dtype=int32_ref>
 [[1 9]
 [8 5]]
<tf.Variable 'Variable:0' shape=(784, 255) dtype=float32_ref>
 [[ 0.   0.   0. ...,  0.   0.   0.]
 [ 0.   0.   0. ...,  0.   0.   0.]
 [ 0.   0.   0. ...,  0.   0.   0.]
 ...,
 [ 0.   0.   0. ...,  0.   0.   0.]
 [ 0.   0.   0. ...,  0.   0.   0.]
 [ 0.   0.   0. ...,  0.   0.   0.]]
```

Declaring a variable

Variable Global initialization

Declaring different variables and using .eval() functions to get the required output

Place holders: feed_dict

- How do we feed external data into computational graphs?
- Create a graph with “Place holders” and fill the data through these place holders
- Need to use a feed_dict that will map the place holders to the data.
- In simple terms feed_dict fills the data in the place holders
- feed_dict can feed data using lists, arrays, matrices etc.,

Code: Place holders

```
w1=tf.placeholder(tf.int32)
w2=tf.placeholder(tf.int32)
print(w1)
print(w2)
```

```
Tensor("Placeholder_34:0", dtype=int32)
Tensor("Placeholder_35:0", dtype=int32)
```

```
out = tf.multiply(w1, w2)
print(out)
```

```
Tensor("Mul:0", dtype=int32)
```

```
sess=tf.Session()
print(sess.run([out], feed_dict={w1:22,w2:5}))
sess.close()
```

```
[110]
```

Defining placeholders

Graph

Feeding the values into the placeholder

Place holders

```
#### Place holders for matrix multiplication

m1=tf.placeholder(tf.int32,shape=(1,2))
m2=tf.placeholder(tf.int32,shape=(2,2))
out = tf.matmul(m1, m2)
sess=tf.Session()
print(sess.run([out], feed_dict={m1:np.array([[3,2]]), m2:np.array([[4,8], [9,3]])}))
sess.close()
```

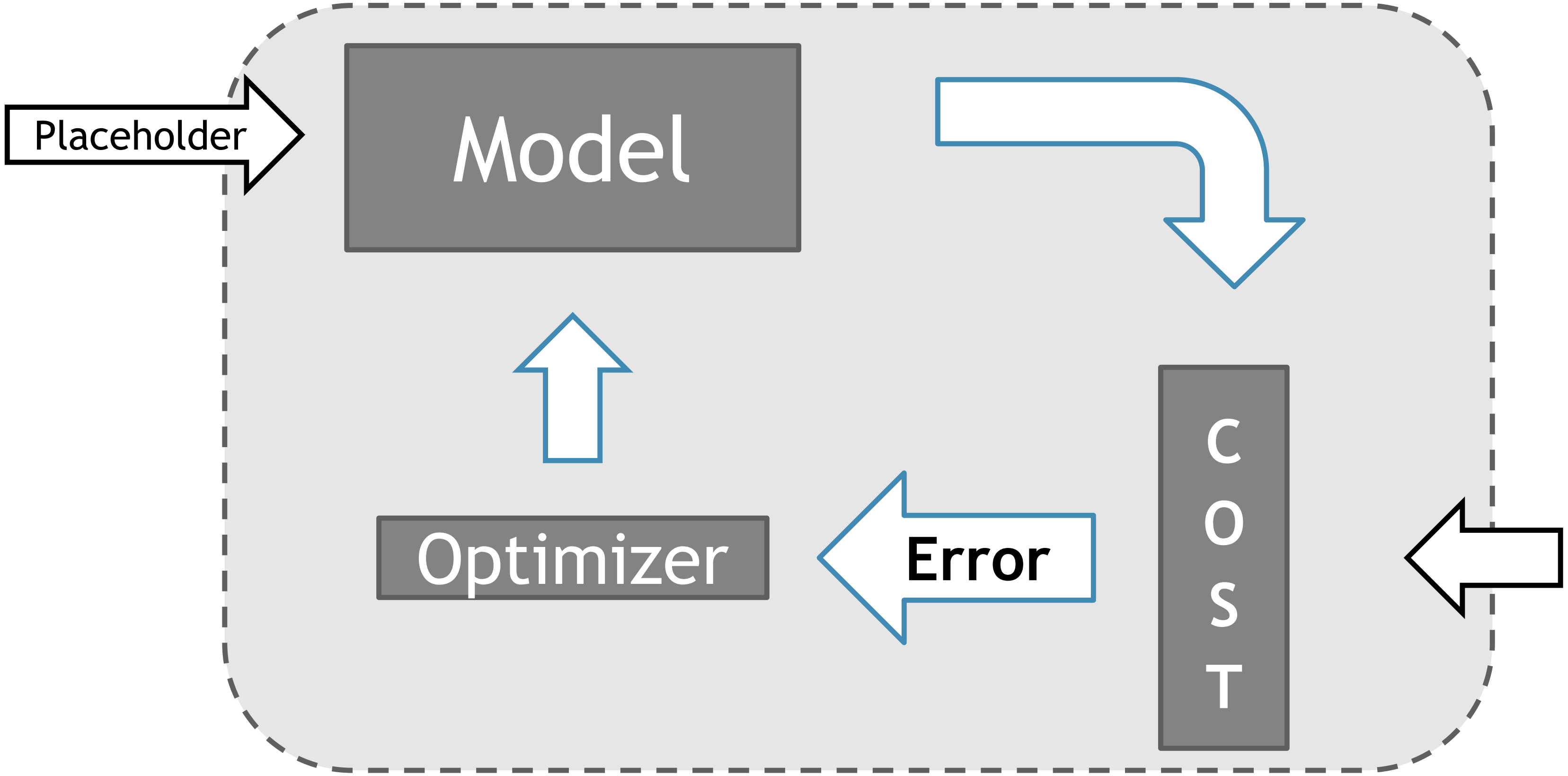
```
[array([[30, 30]])]
```

Defining a placeholder with dimensions

Model building on TensorFlow

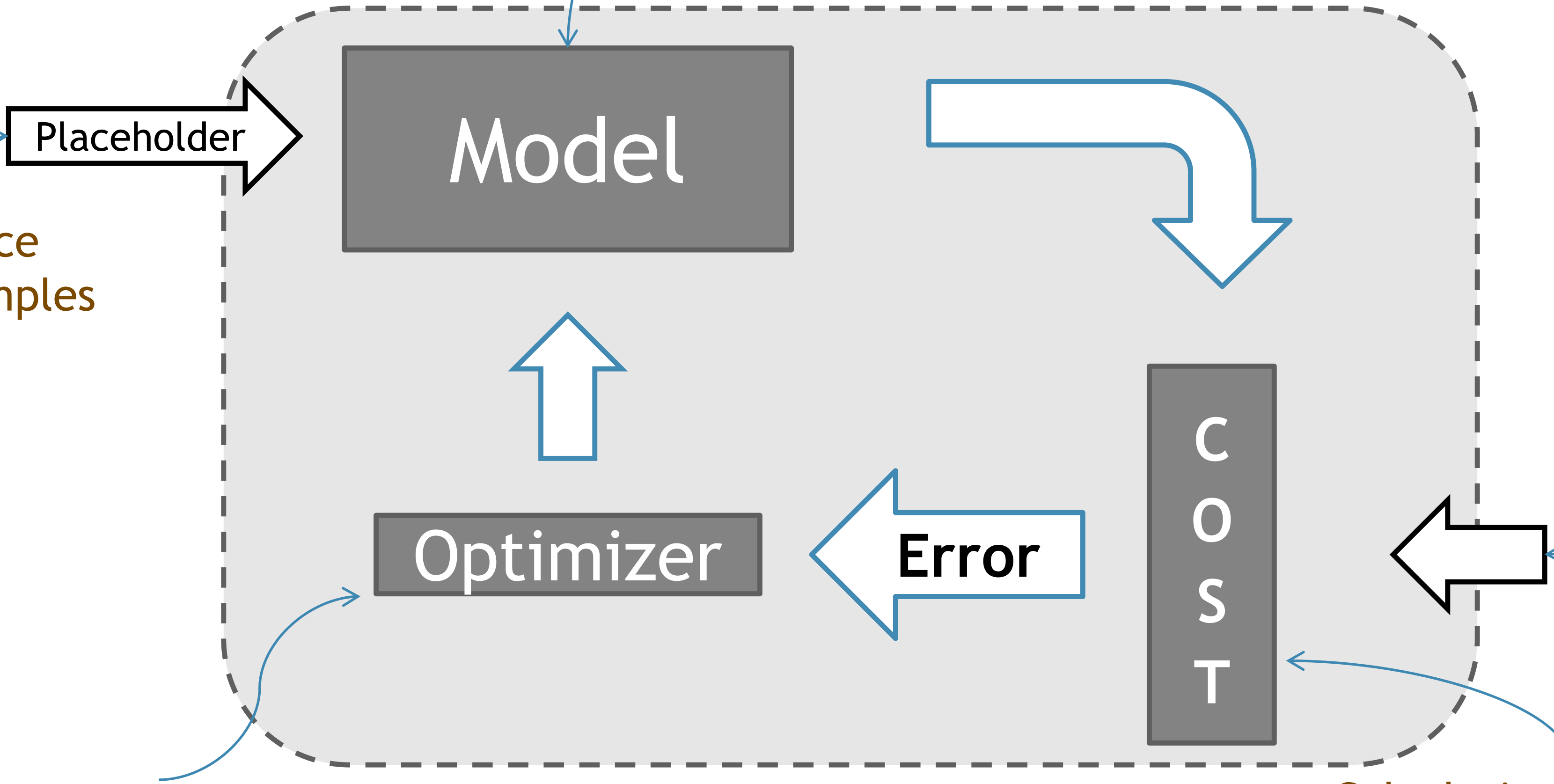
- Create place holders for X and Y tensors
- Take a note of the model. Write model equation Y=sig(X*W + b) or Y=X*W+b
- Initialize the model parameters W
- Define loss function(cost) - C
- Optimise the cost function to find the best parameter estimates

Graph



Graph

Operations on variables to make predictions



To introduce input examples

Place holder to introduce labels

Way to optimize variables to minimize the error

Calculating the error in model

LAB: Linear Regression on TensorFlow

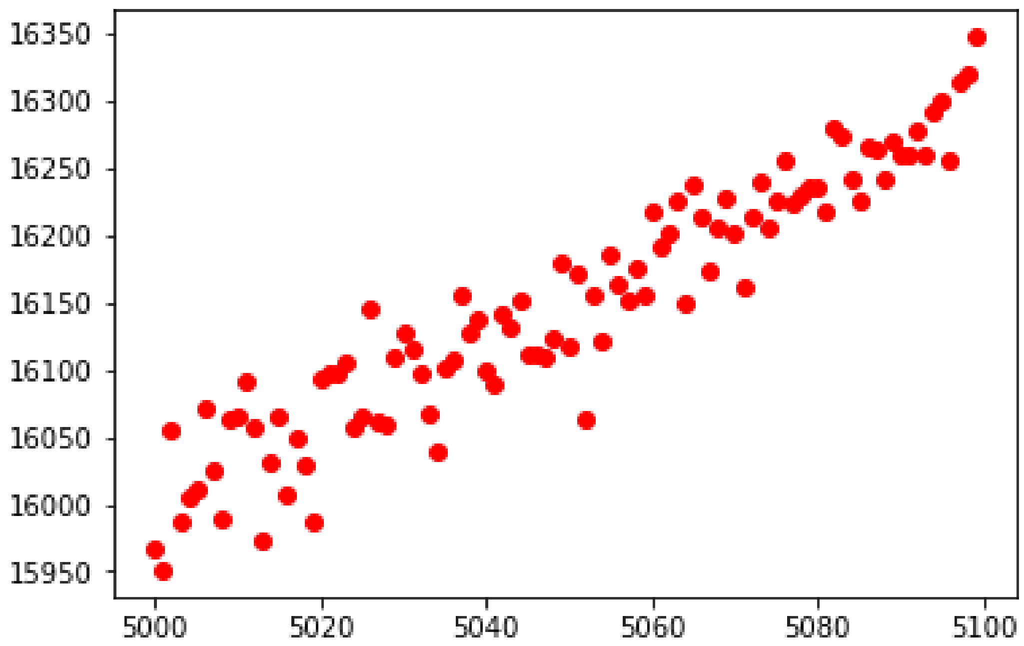
- Use code to generate the data for x and y.
- Build a regression model in TensorFlow to find W
- Verify the weight with the original data

Code: Linear Regression on TensorFlow

```
#This step is for data creation, x1 and y
x1= np.array(range(5000,5100)).reshape(-1,1)
y_actual=[3*i+np.random.normal(1000, 30) for i in x1]

import matplotlib.pyplot as plt
plt.plot(x1, y_actual, 'ro')
plt.show()
```

Randomly generating our X and y



Code: Linear Regression on TensorFlow

```
# PLACEHOLDERS
x = tf.placeholder(tf.float32, [None, 1])
y = tf.placeholder(tf.float32, [None, 1])

#Model y=X*W + b
W = tf.Variable(tf.truncated_normal([1,1], stddev=0.05))
b = tf.Variable(tf.random_normal([1]))
output = tf.add(tf.matmul(x, W), b)

#Cost function Σ[ yi - ( xi·W + b ) ]²
loss = tf.reduce_sum(tf.square(output - y))

#Optimization
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.0000000001)
optimizer = optimizer.minimize(loss)
```

Defining a placeholder with dimensions

Defining the model with initial weights, bias and output

Defining cost function

Choosing optimizer function with learning rate

Code: Linear Regression on TensorFlow

- Running our graph as session

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(0,501):
        sess.run(optimizer, feed_dict={
                x: x1,
                y: y_actual
            })
        if epoch%50 == 0:
            print("epoch is ", epoch," W is ", sess.run(W), "and b is ", sess.run(b))
```

```
epoch is  0  W is  [[0.22485927]] and b is  [0.3362292]
epoch is  50  W is  [[2.9797516]] and b is  [0.3367749]
epoch is  100  W is  [[3.1808896]] and b is  [0.3368148]
epoch is  150  W is  [[3.195575]] and b is  [0.33681774]
epoch is  200  W is  [[3.1966467]] and b is  [0.33681774]
epoch is  250  W is  [[3.1967251]] and b is  [0.33681774]
epoch is  300  W is  [[3.1967292]] and b is  [0.33681774]
epoch is  350  W is  [[3.1967292]] and b is  [0.33681774]
epoch is  400  W is  [[3.1967292]] and b is  [0.33681774]
epoch is  450  W is  [[3.1967292]] and b is  [0.33681774]
epoch is  500  W is  [[3.1967292]] and b is  [0.33681774]
```

Initializing all the variables in computation graph

Range of epochs

Feeding the data into X, y placeholders

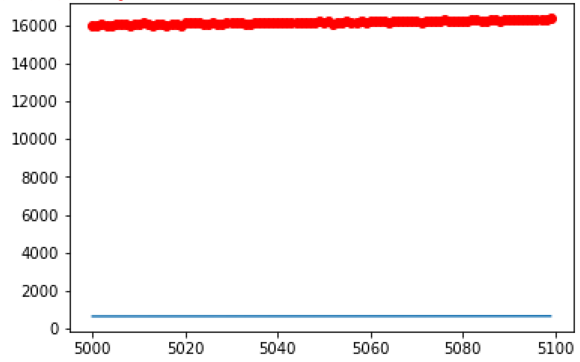
Code: Linear Regression on TensorFlow

```
    if epoch%100 == 0:
        print("epoch is ", epoch," W is ", sess.run(W), "and b is ", sess.run(b))
        plt.plot(x1, y_actual, 'ro')
        plt.plot(x1, sess.run(W) * x1 + sess.run(b))
        plt.show()
```

Plot the line fit every 100 epoch to visualize the training

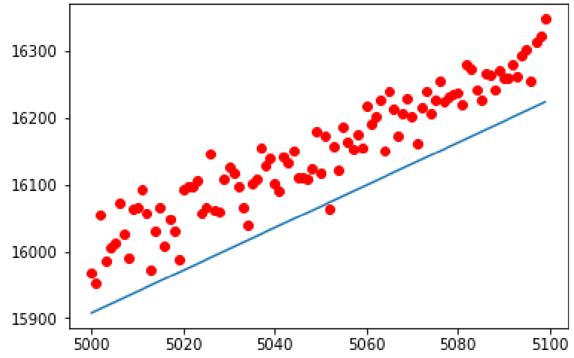
epoch is 0 W is [[0.12888031]] and b is [0.54140586]

Epoch :0



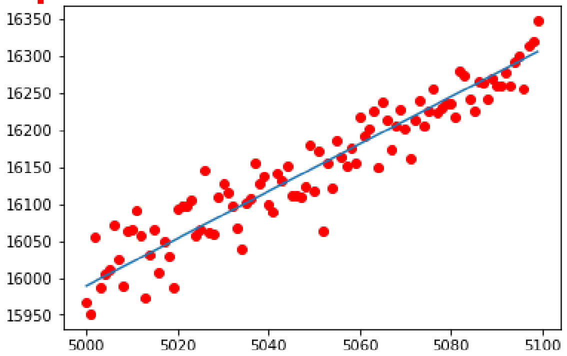
epoch is 100 W is [[3.18148255]] and b is [0.54201001]

Epoch :100



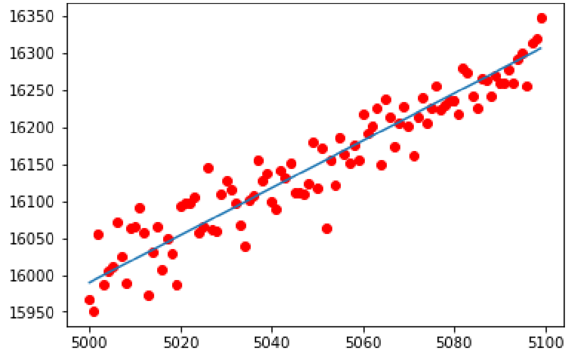
epoch is 200 W is [[3.19775558]] and b is [0.54201281]

Epoch :200



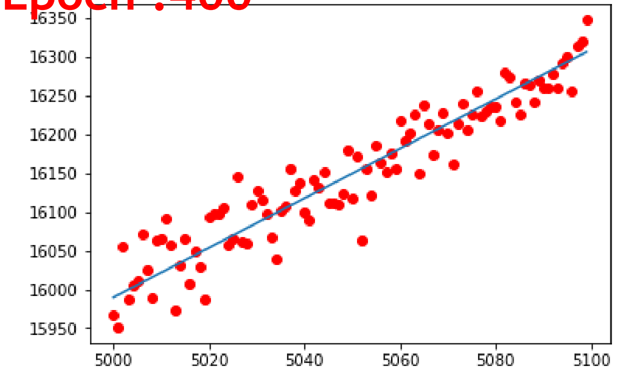
epoch is 300 W is [[3.19784021]] and b is [0.54201281]

Epoch :300



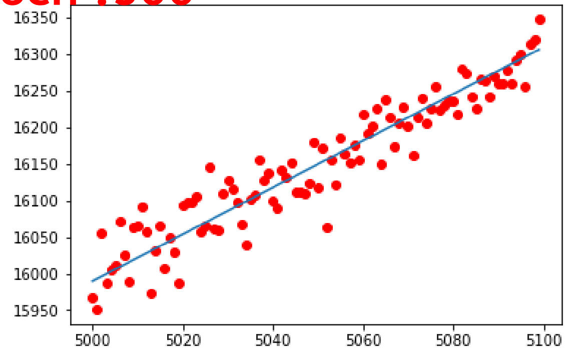
epoch is 400 W is [[3.19784021]] and b is [0.54201281]

Epoch :400



epoch is 500 W is [[3.19784021]] and b is [0.54201281]

Epoch :500



LAB: Simple ANN Model

- Data:
- Our model function:

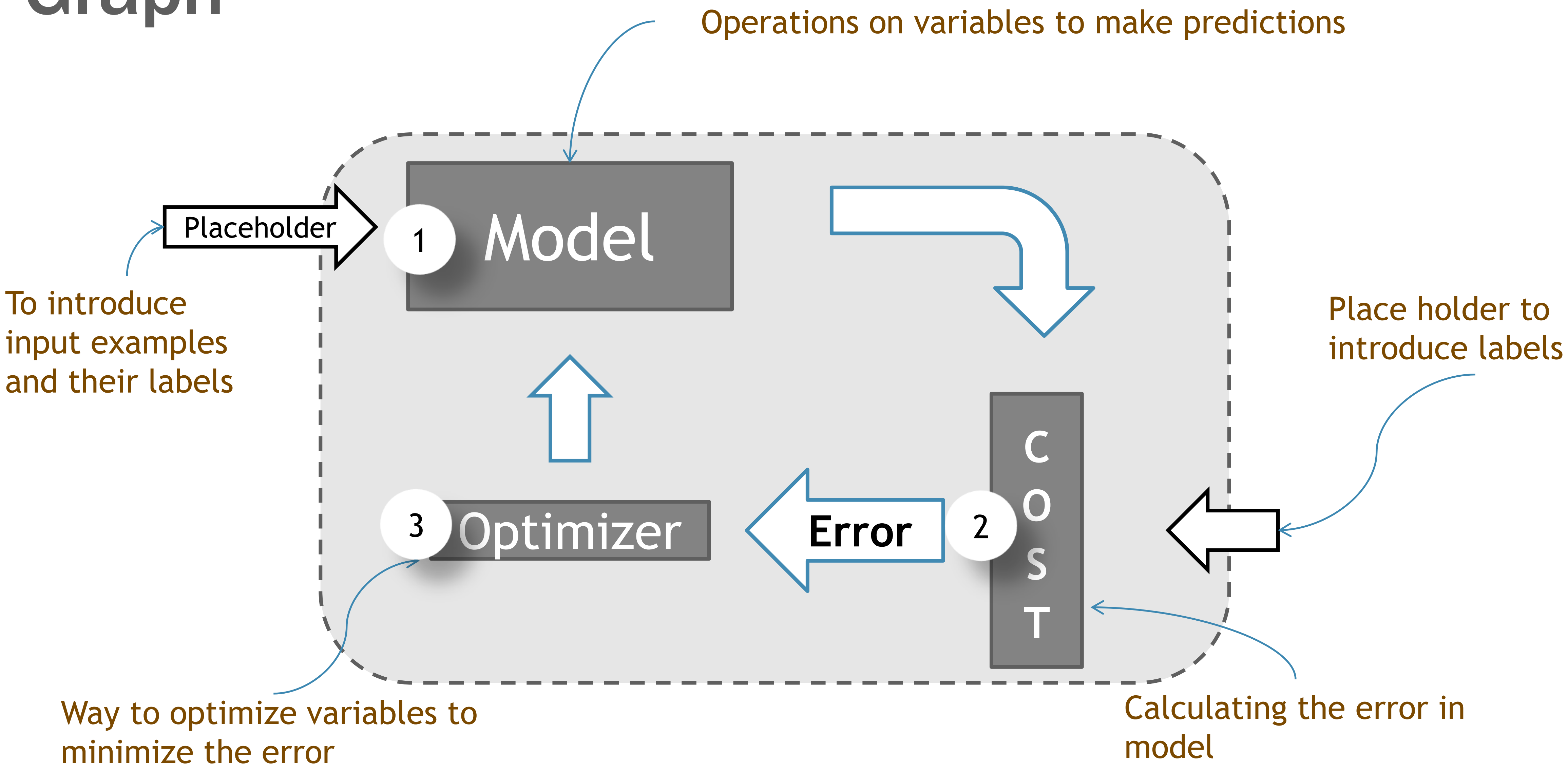
$$h = Sigmoid(Wx + b)$$

- The cost function:

$$J(W, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - h)^2$$

- Optimizer for back propagation

Graph


Operations on variables to make predictions
Placeholder
1
Model
To introduce input examples and their labels
Place holder to introduce labels
3
Optimizer
Error
2
COST
Way to optimize variables to minimize the error
Calculating the error in model

Lab: ANN Model– Single Perceptron

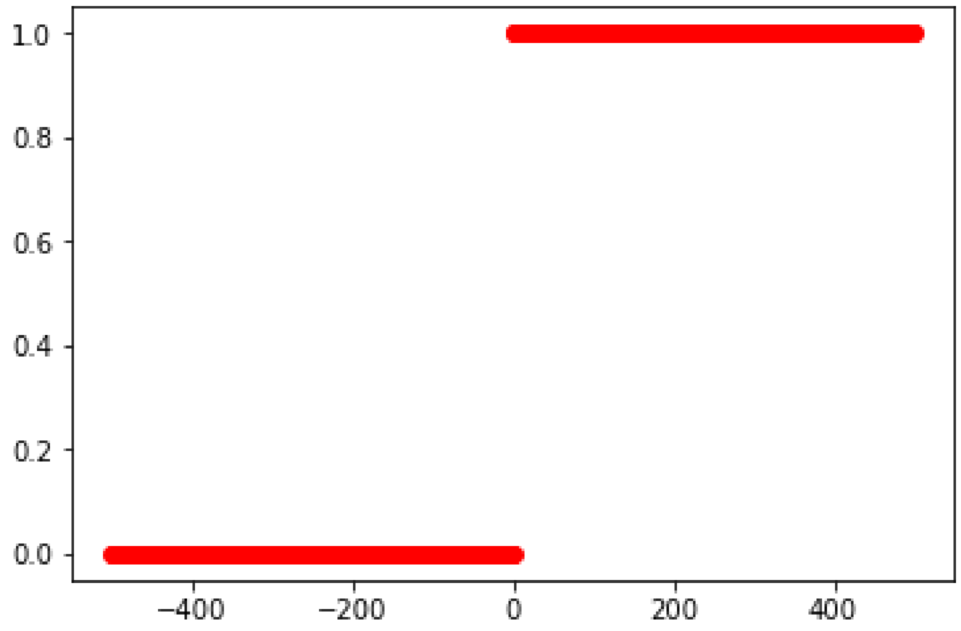
- Generate dummy x and y data
- Define our Model as Computation Graph
  - Place holders for X and y
  - W, b variables
  - Model output function
  - Cost function
  - Optimizer
- Run the graph as session, feeding data into placeholders

Code: ANN Model– Single Perceptron

```
# This step is for data creation, x1 and y
x1_data= np.array(range(-500,500)).reshape(-1,1)
y_actual=np.array([0 if i < 1 else 1 for i in x1_data]).reshape(-1,1)

import matplotlib.pyplot as plt
plt.plot(x1_data, y_actual, 'ro',)
plt.show()
```

Randomly generating our X and y



Code: ANN Model– Single Perceptron

```
#Simple ANN

# PLACEHOLDERS
x = tf.placeholder(tf.float32, [None, 1])
y = tf.placeholder(tf.float32, [None, 1])

#Model y=sigmoid(X*W + b)
W = tf.Variable(tf.random_normal([1,1], mean=0.6, stddev=0.005))
b = tf.Variable(tf.random_normal([1], mean=0, stddev=0.05))
output = tf.sigmoid(tf.add(tf.matmul(x, W) , b))

#Cost function Σ[ yi - ( xi·W + b ) ]²
loss = tf.reduce_sum(tf.square(output - y))
#cross_entropy = tf.reduce_mean(-tf.reduce_sum(* tf.log(y), reduction_indices=[1]))

#Optimization
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.0000001)
optimizer = optimizer.minimize(loss)
```

Defining a placeholder with dimentions

Defining initial weights and bias

Model function passed through Sigmoid(Non-Linearity)

Defining cost function

Choosing optimizer function with learning rate

Code: ANN Model– Single Perceptron

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for epoch in range(0,10001):
        sess.run(optimizer, feed_dict={
                x: x1_data,
                y: y_actual
            })
        if epoch%1000 == 0:
            print("epoch is ", epoch,  " W is ", sess.run(W), "and b is ", sess.run(b),)
```

```
epoch is  0  W is  [[ 0.59725714]] and b is  [ 0.00826101]
epoch is  1000  W is  [[ 0.59737635]] and b is  [ 0.00823493]
epoch is  2000  W is  [[ 0.59749556]] and b is  [ 0.00820885]
epoch is  3000  W is  [[ 0.59761477]] and b is  [ 0.00818278]
epoch is  4000  W is  [[ 0.59773397]] and b is  [ 0.0081567]
epoch is  5000  W is  [[ 0.59785318]] and b is  [ 0.00813062]
epoch is  6000  W is  [[ 0.59797239]] and b is  [ 0.00810455]
epoch is  7000  W is  [[ 0.5980916]] and b is  [ 0.00807847]
epoch is  8000  W is  [[ 0.59821081]] and b is  [ 0.00805239]
epoch is  9000  W is  [[ 0.59833002]] and b is  [ 0.00802631]
epoch is  10000  W is  [[ 0.59844923]] and b is  [ 0.00800024]
```

Initializing all the variables in computation graph

Range of epochs

Feeding the data into X, y placeholders

LAB: MNIST on TensorFlow

Example of ANN on MNIST data

LAB: MNIST on TensorFlow

- Importing the MNIST data, using existing TensorFlow APIs

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Successfully downloaded train-images-idx3-ubyte.gz 9912422 bytes.
Extracting MNIST_data/train-images-idx3-ubyte.gz
Successfully downloaded train-labels-idx1-ubyte.gz 28881 bytes.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Successfully downloaded t10k-images-idx3-ubyte.gz 1648877 bytes.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Successfully downloaded t10k-labels-idx1-ubyte.gz 4542 bytes.
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

LAB: MNIST on TensorFlow

- Basic observation on the imported MNIST data

```
print("Number of training samples:", mnist.train.num_examples)
print("Number of testing samples:",mnist.test.num_examples)
```

```
Number of training samples: 55000
Number of testing samples: 10000
```

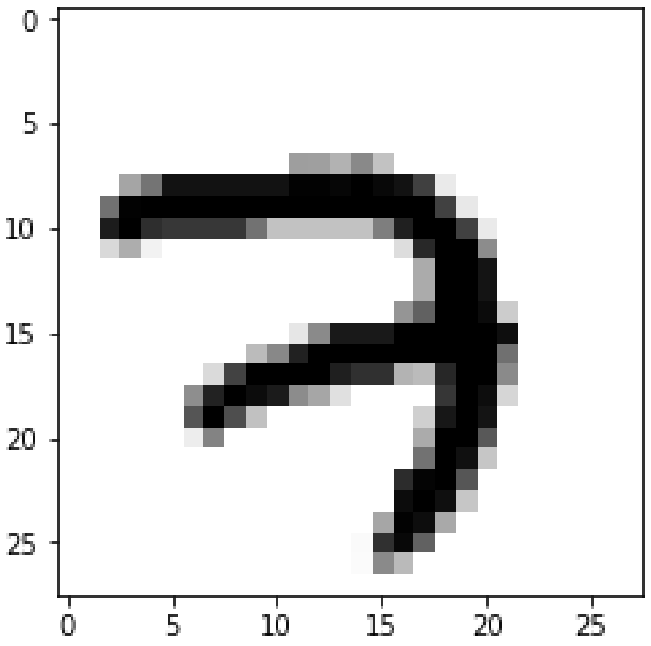
```
print("Hot encoded output label of first 10 samples: \n", mnist.train.labels[:10])
```

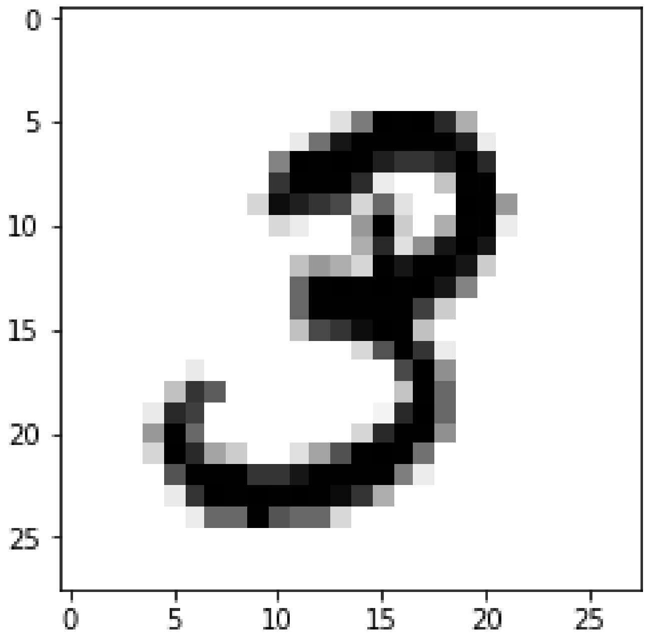
```
Hot encoded output label of first 10 samples:
 [[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]]
```

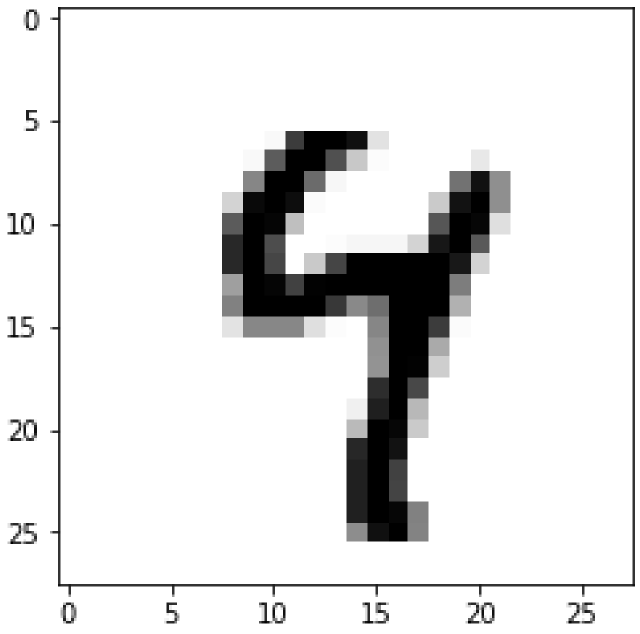
LAB: MNIST on TensorFlow

- Converting a few observations from numerical data into images

```
for i in range(3):
    image = mnist.train.images[i].reshape([28,28])
    plt.imshow(image, cmap=plt.get_cmap('gray_r'))
    plt.show()
```







LAB: MNIST on TensorFlow

- Writing our basic computation graph.

```
import tensorflow as tf
x = tf.placeholder("float", [None, 784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
```

```
matm=tf.matmul(x,W)
y = tf.nn.softmax(matm + b)
y_ = tf.placeholder("float", [None,10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

Code: MNIST on TensorFlow

- Running our computation graph into a session

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(500):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
    if i%100 == 0:
        print("accuracy in epoch ", i , " is ",
              sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
```

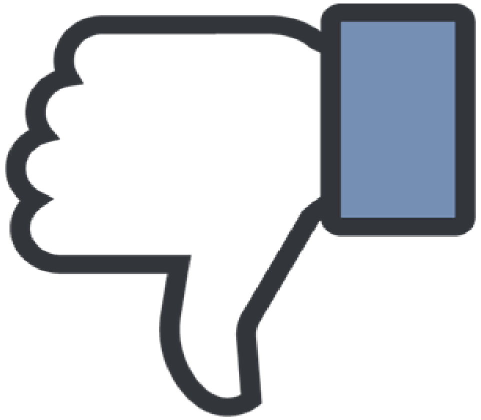
```
accuracy in epoch  0  is  0.3915
accuracy in epoch  100  is  0.8274
accuracy in epoch  200  is  0.9066
accuracy in epoch  300  is  0.9128
accuracy in epoch  400  is  0.9105
```

We were able to achieve 91% accuracy within a few seconds

TensorFlow Advantages

- Computational graphs make complex deep learning computations easy
- Very fast compared to other frameworks
- Visualizations - Tensor Board
- GPU for faster computations

- Lot of low level coding, may take some time to get familiarity.



Keras: TensorFlow made easy!!!

- Wrapper
  - Keras is a wrapper on top of TensorFlow.
  - High level API written in Python
- Easy
  - Less lines of code.
  - Easy to learn and implement deep learning models
- Best
  - Wide ranging options
  - Probably the best wrapper on top of TensorFlow

Keras

statinfer

Keras: TensorFlow made easy!!!

- Non-coders
  - Simple straight forward syntax
  - Provides detailed model summary statistics
  - Non-coders can start deep learning models with Keras
- Documentation
  - Good documentation on keras.io
  - Good support from community and userbase

K Keras

Keras

You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Read the documentation at Keras.io.

Major steps in model building on Keras

1. Prepare your data.
2. Model Configuration
   - Add input layer to your model object
   - Add the hidden layers
   - Add the output layer
3. Compile the model object
4. Finally train the model

What are Layers?

Sequence of Layers in the model

- Models building is done using sequence of layers
- The sequential model is a linear stack of layers.

- The first layer in the stack is “Input Layer” - Model receives the information on input shape
- The last layer is “Output Layer”. The model gets information on labels.
- We can add all the “model layers” in between. The model will prepare the weight parameters

- Lets see an example

LAB: MNIST on Keras

Example of ANN on MNIST data using Tensorflow-Keras

Code: MNIST on Keras

- Importing our Keras dependencies

```
#!pip install keras
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras. optimizers import SGD
```

```
Using TensorFlow backend.
```

Code: MNIST on Keras

- Keras's default MNIST data is in different format, make it a bit friendly.

```
# the data, shuffled and split between train and test sets
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
num_classes=10
x_train = X_train.reshape(60000, 784)
x_test = X_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

Scaling the pixel values between 0 and 1

```
# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(Y_train, num_classes)
y_test = keras.utils.to_categorical(Y_test, num_classes)
```

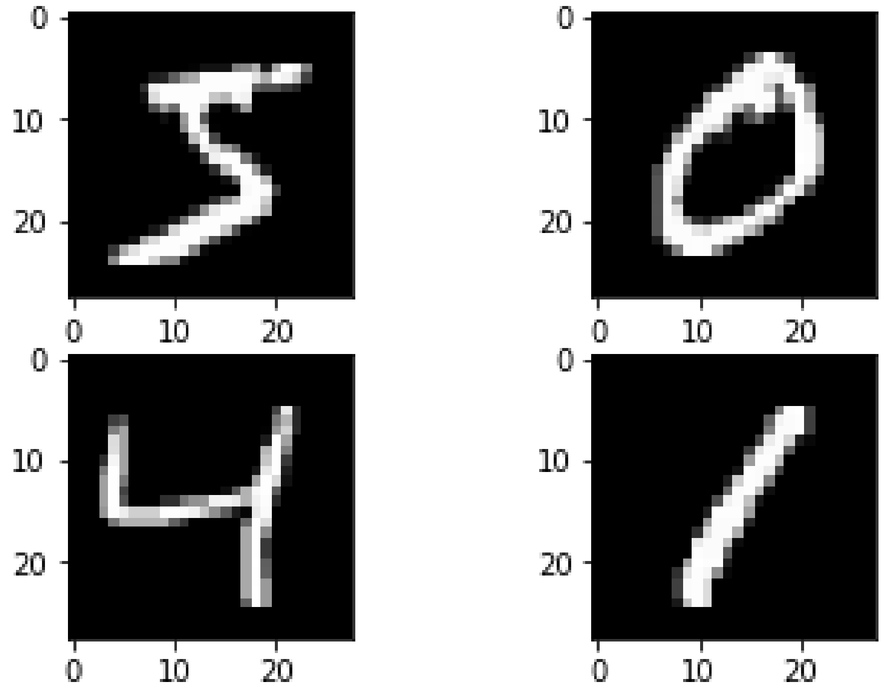
Class vector needs to be converted into binary class matrices

Code: MNIST on Keras

- Having a look at images using matplotlib

```
%matplotlib inline
import matplotlib.pyplot as plt
# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(X_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(X_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(X_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(X_train[3], cmap=plt.get_cmap('gray'))

# show the plot
plt.show()
```



Code: MNIST on Keras

- Defining our model and model parameters

```
batch_size = 128
num_classes = 10
epochs = 20

model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))

model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
```

Basic model parameters

Keras model blocktype

Adding our first dense layer

Adding activation Layer

Code: MNIST on Keras

- Understanding the shape of our layers

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 512)               401920
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

512(nodes) X 784(Input Shape) =401408

512(Input shape) X 10(Nodes)+10(Bias) =5130

Code: MNIST on Keras

- Compiling and running our model

```
model.compile(loss='categorical_crossentropy',optimizer=SGD(),metrics=['accuracy'])

model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,validation_data=(x_test, y_test))
```

Compiling by giving: loss function, optimizer and validation matric

```
Epoch 15/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.4136 - acc: 0.8882 - val_loss: 0.3909 - val_a
8947
Epoch 16/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.4049 - acc: 0.8899 - val_loss: 0.3822 - val_a
8971
Epoch 17/20
60000/60000 [==============================] - 5s 90us/step - loss: 0.3970 - acc: 0.8915 - val_loss: 0.3766 - val_acc: 0.
8969
Epoch 18/20
60000/60000 [==============================] - 5s 88us/step - loss: 0.3902 - acc: 0.8925 - val_loss: 0.3693 - val_acc: 0.
8982
Epoch 19/20
60000/60000 [==============================] - 6s 92us/step - loss: 0.3839 - acc: 0.8940 - val_loss: 0.3639 - val_acc: 0.
8992
Epoch 20/20
60000/60000 [==============================] - 5s 91us/step - loss: 0.3784 - acc: 0.8952 - val_loss: 0.3586 - val_acc: 0.
9011

<keras.callbacks.History at 0x144e0898>
```

Actual Training or running by feeding in the data

Code: MNIST on Keras

- Final Scores and weights of model

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("weights present in the model:",model.get_weights())
```

```
Test loss: 0.359743064487
Test accuracy: 0.9
weights present in the model: [array([[ 0.05601457,  0.04763146, -0.00425783, ..., -0.0032696 ,
        -0.01738391, -0.02678162],
       [ 0.05518264,  0.03735916, -0.01287513, ..., -0.0517627 ,
         0.02932869,  0.04873633],
       [-0.05539942, -0.01782691,  0.06680593, ...,  0.04894154,
         0.03940008,  0.05391582],
       ...,
       [ 0.00598568, -0.06686999, -0.05277239, ..., -0.06113473,
        -0.06114446,  0.05285732],
       [-0.06446217,  0.01289789,  0.00475509, ...,  0.01564217,
        -0.03017441,  0.00026505],
       [ 0.03756514, -0.01455592,  0.04311423, ..., -0.06567205,
         0.02208738,  0.01103662]], dtype=float32), array([ -8.54801969e-04,  -6.74002664e-03,   1.14037693e-02,
        -3.32280598e-03,   1.05455611e-02,  -6.75419625e-03,
         3.75575712e-03,   8.54193699e-03,  -1.57072477e-03,
        -3.88102932e-03,   1.24682065e-05,   3.53176845e-03,
        -8.18445405e-05,  -1.22737465e-03,  -1.22051099e-02,
```

Code Comparison –Keras vs TensorFlow

MNIST on TensorFlow

```
import tensorflow as tf
x = tf.placeholder("float", [None, 784])
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
```

```
matm=tf.matmul(x,W)
y = tf.nn.softmax(tf.matmul(x,W) + b)
y_ = tf.placeholder("float", [None,10])
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
for i in range(500):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
    correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

- Keras is not just about less lines of code.
- Keras syntax is also simple

MNIST on Keras

```
model = Sequential()
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer=SGD(),metrics=['accuracy'])
model.fit(x_train, y_train,batch_size=batch_size,epochs=epochs,validation_data=(x_test, y_test))
```

Other Advantages of Keras

- Biggest advantage is: Easy and fast
  - Friendliness
  - Modularity
  - Extensibility
- Keras provides easy pipelining of our model.
- Very less and tidy code.
- Pre-existing APIs make our work quite easy.

Conclusion

- The Deep Learning algorithms are really calculation intensive
- There are many deep learning frameworks
- TensorFlow is one such framework and Keras is a high level API on top of it
- Torch is the next best option.