statinfer

TensorFlow and Keras

statinfer.com

Contents

- Deep Learning frameworks
- What is TensorFlow
- Key terms in TensorFlow
- Working with TensorFlow
- Regression Model building
- Keras Introduction
- Keras Advantages
- Working with Keras
- MNIST on Keras

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.

TensorFlow Installation

```
!pip install tensorflow
```

Some key terms

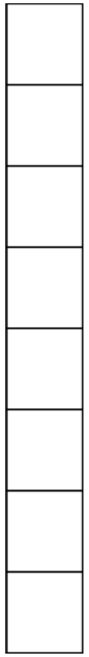
statinfer

- Tensors
- Dimensions
- Computational graphs
- Nodes and Edges

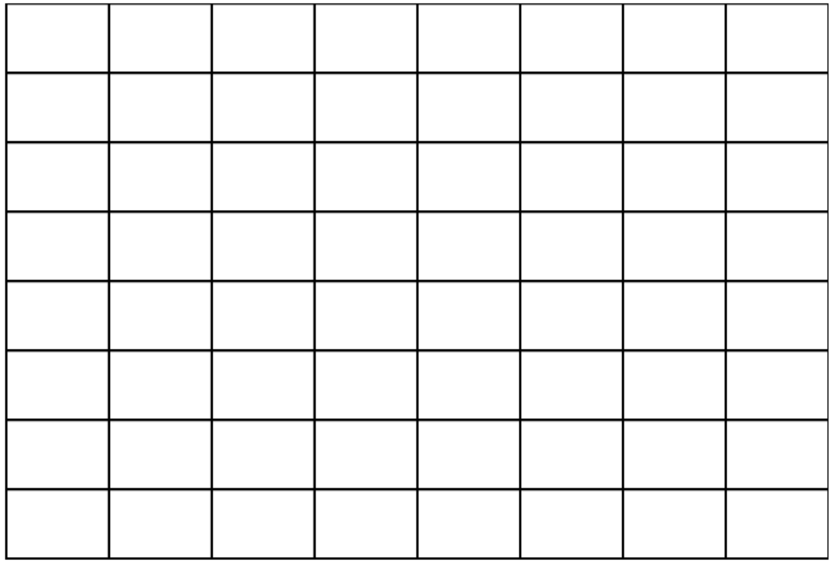
Tensor

Tensor means data

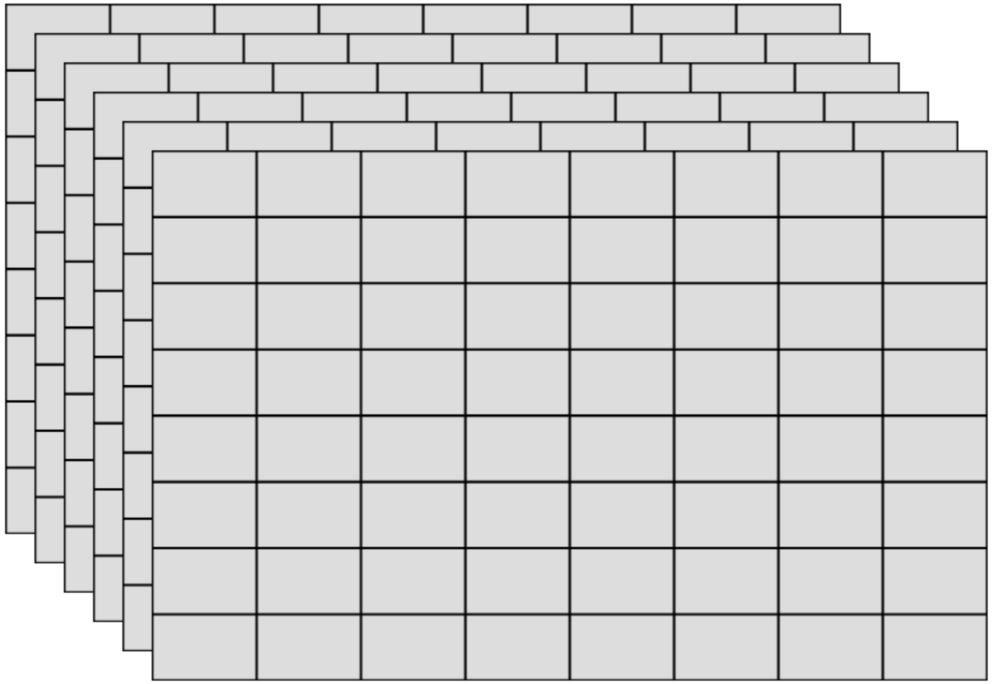
A vector/array is a collection of elements (scalars)



A matrix is a two dimensional vector



A Tensor is a multi-dimensional vector.



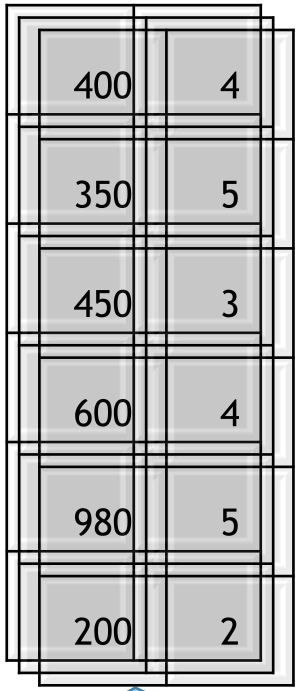
Tensor

400
350
450
600
980
200

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

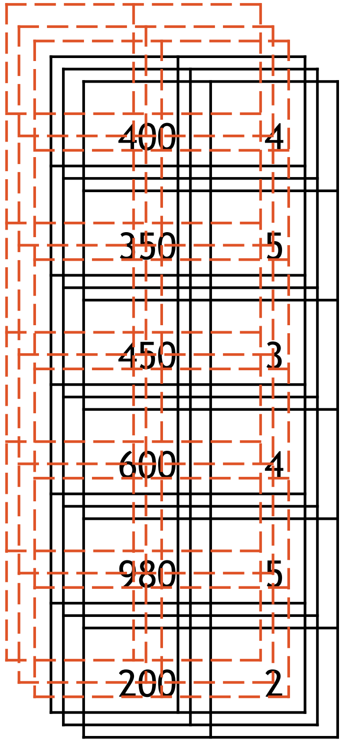
A matrix indicating the product price and rating
A tensor of dimension [6,2]

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]

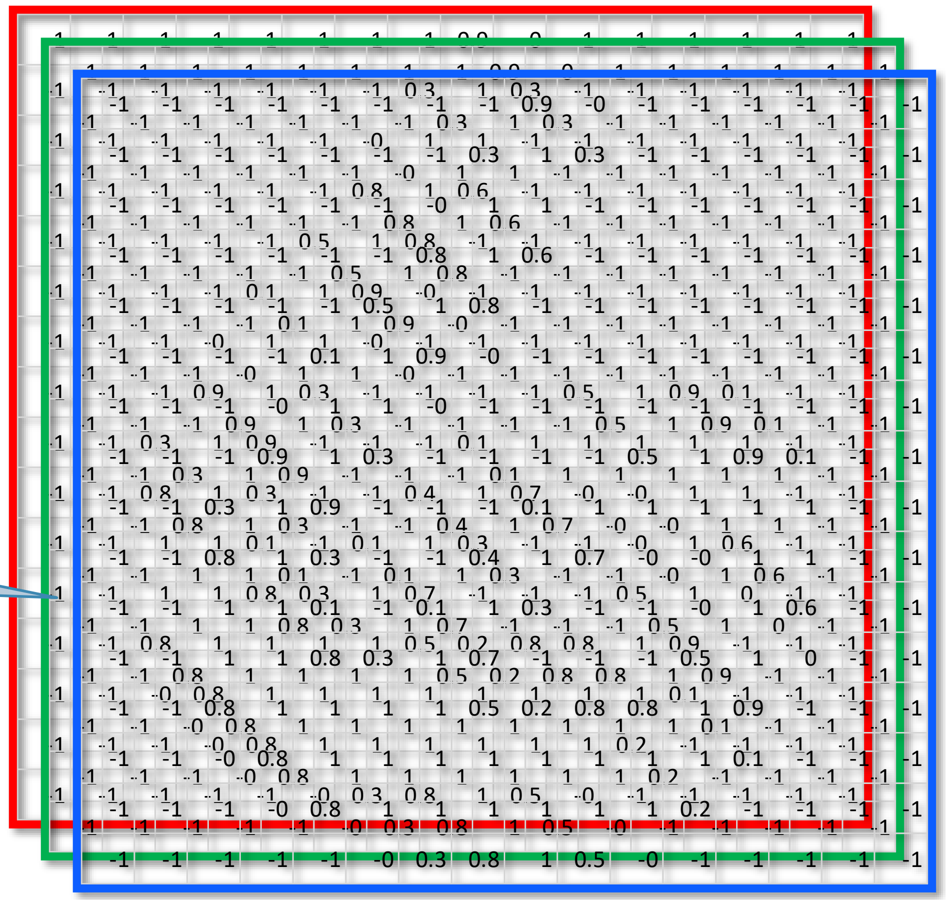
A tensor indicating the product price and rating for last three years in two countries
A tensor of dimension [6,2,3,2]



Tensor for images

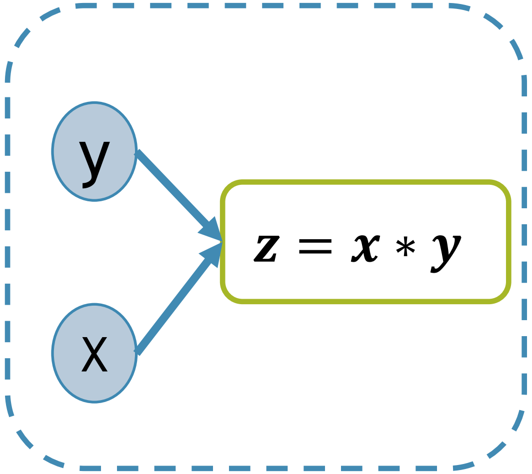
- A colour image is represented as a three dimensional tensor
- [Width, Height, Colour]
- The colour component depth 3, Red, Green and Blue

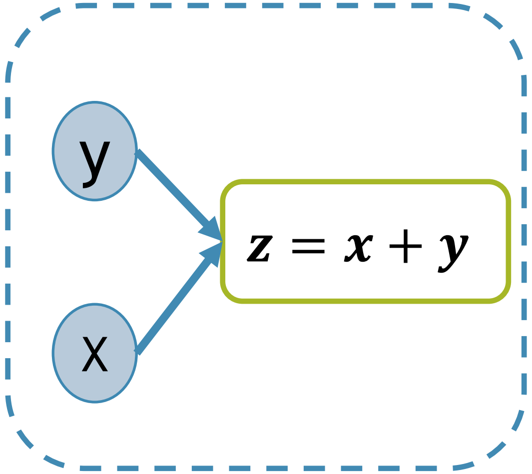
16X16 pixels colour image
A tensor with dimensions
[16,16,3]



Computational graphs

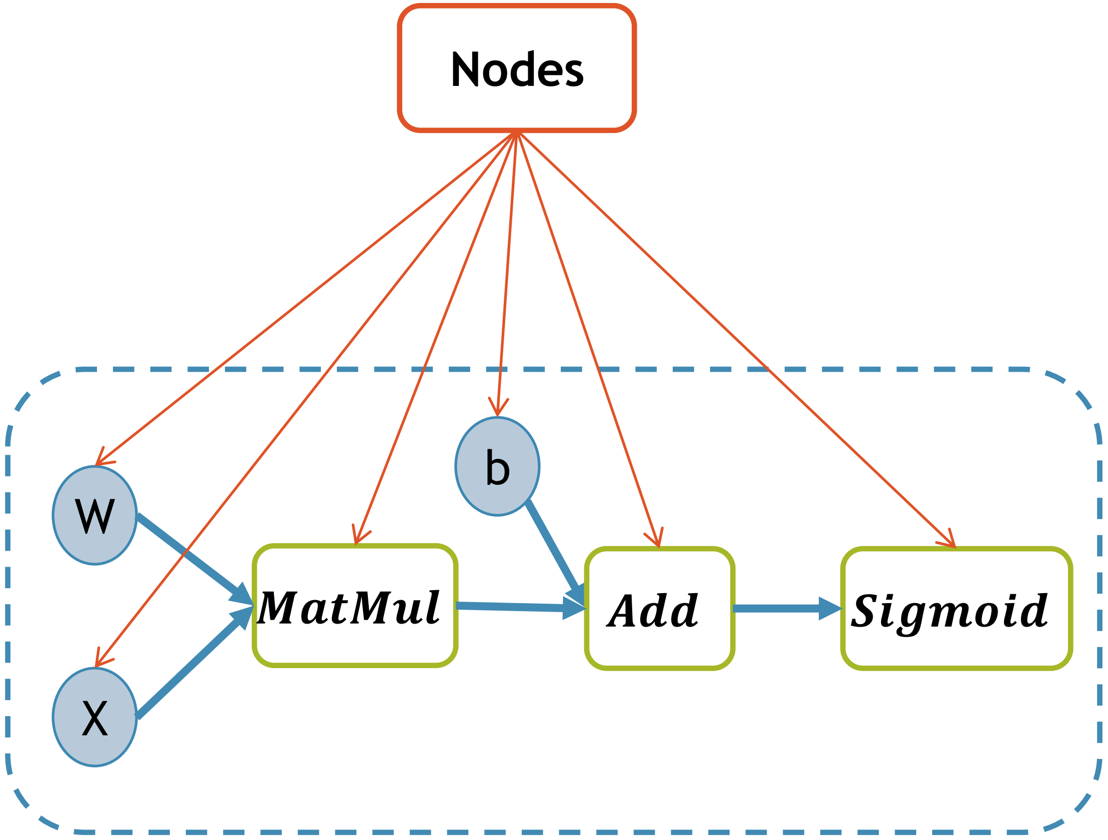
- Inside TensorFlow computations are represented using computational graphs
- You can call it as data flow graph. As sequence of operations on tensors(data)
- It has nodes and edges





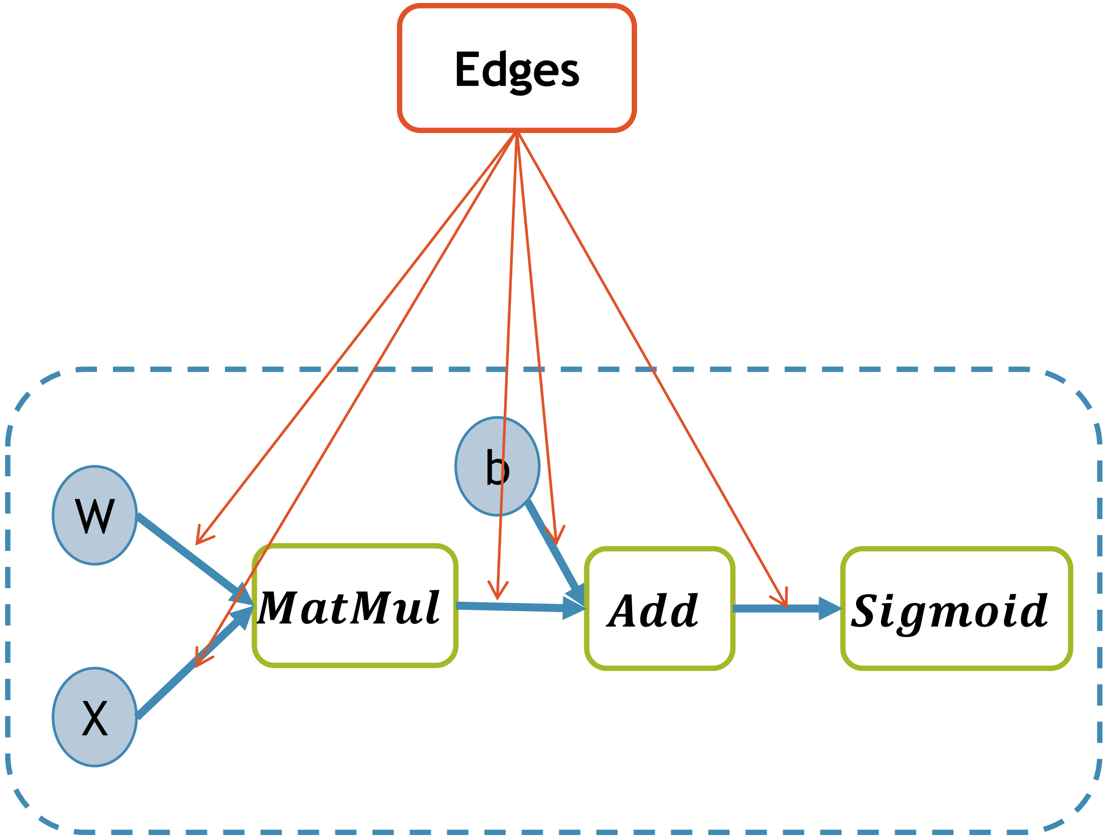
Computation Graphs-Nodes

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

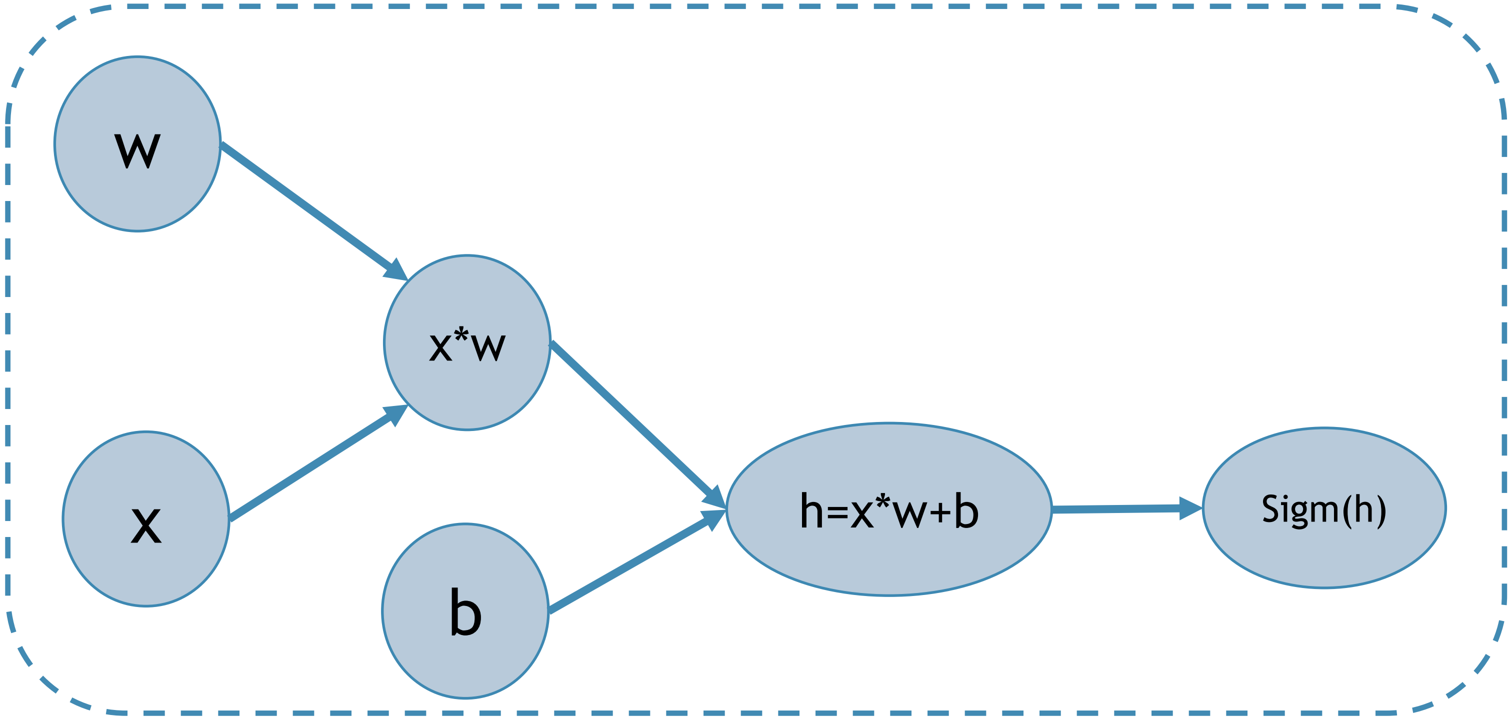


Computation Graphs- Edges

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

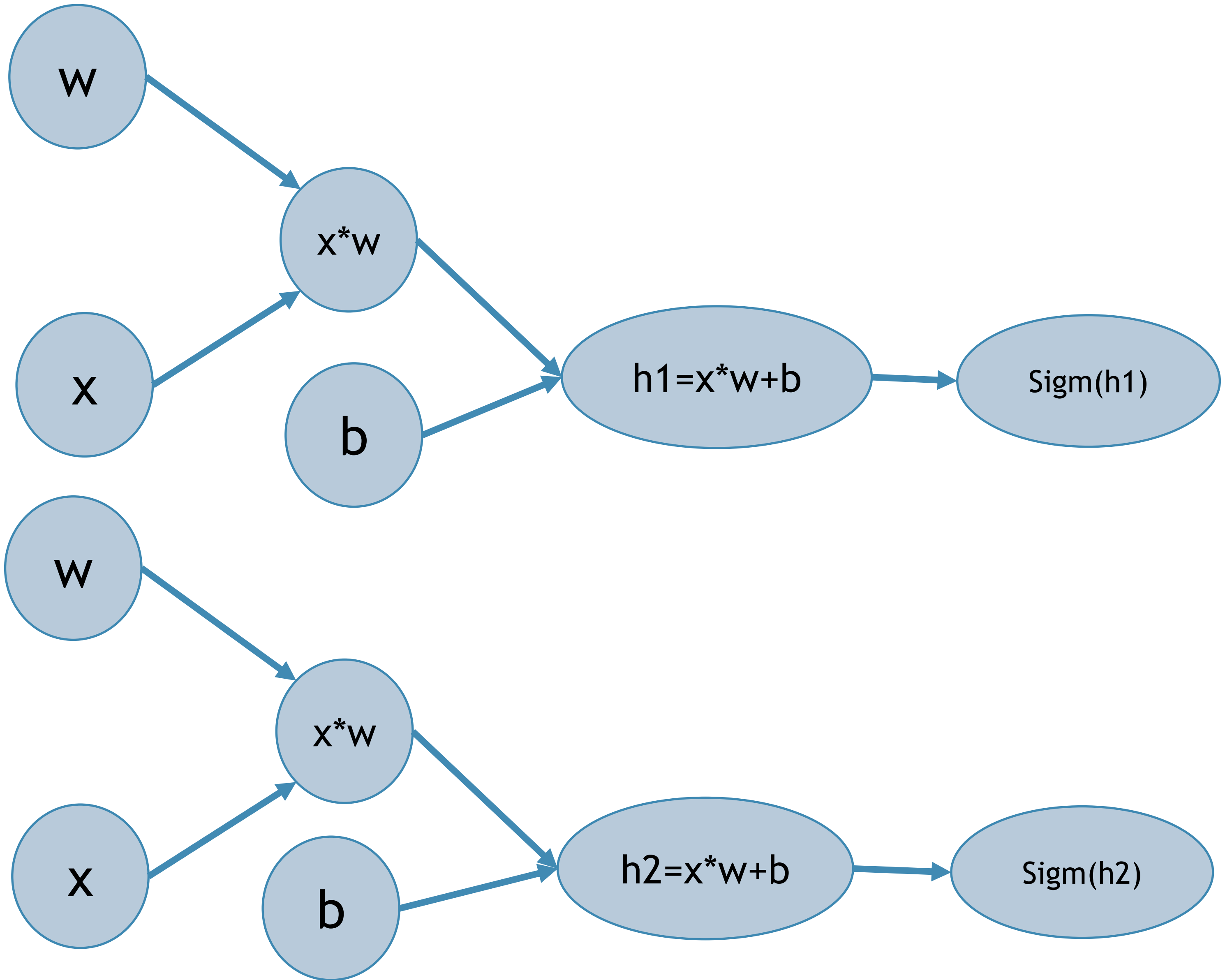


Computation Graphs



- 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!

Computation Graphs



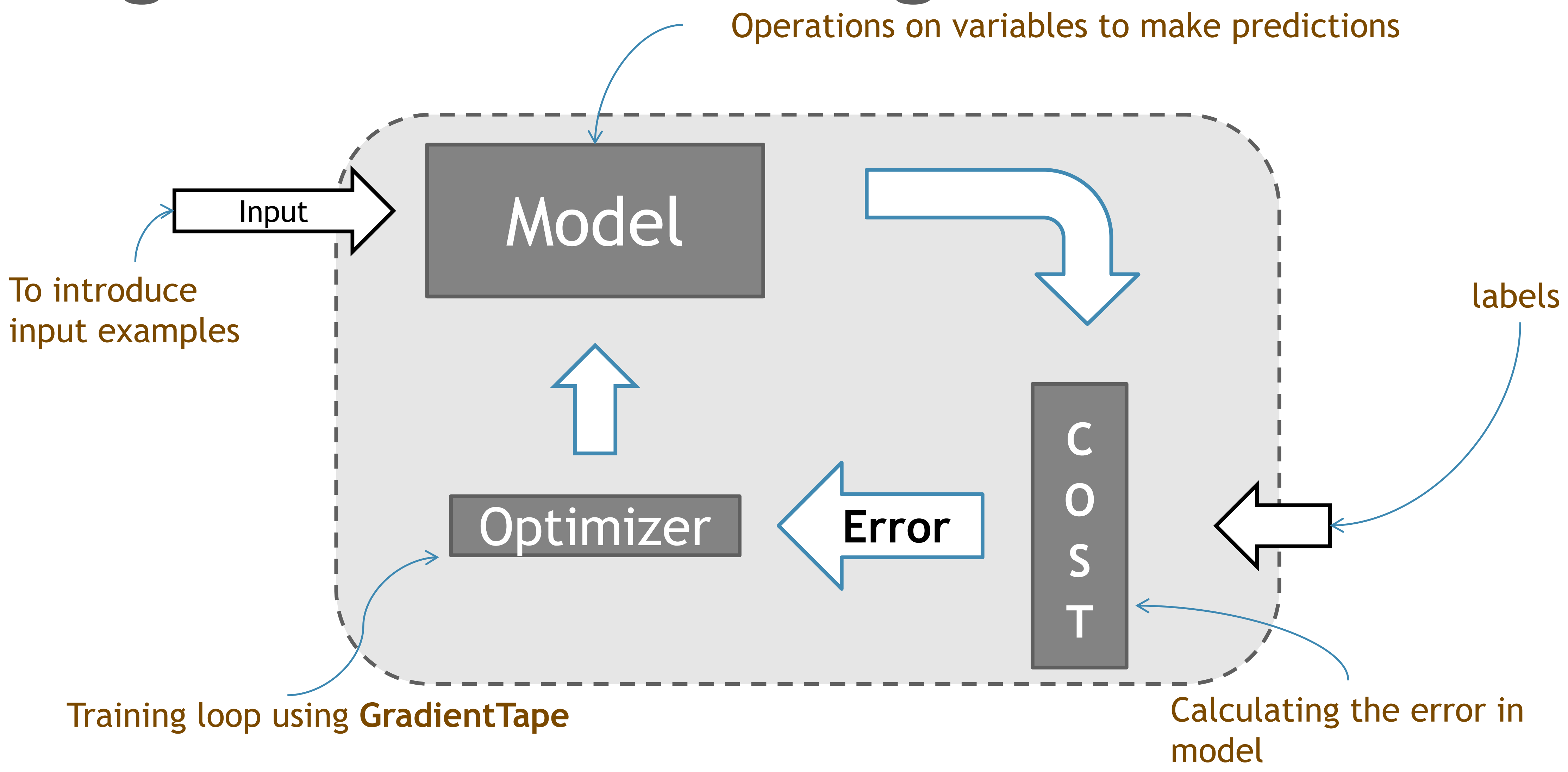
Why Computation Graphs?

- 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.

Model building on TensorFlow

- Write model equation Y=sig(X*W + b) or Y=X*W+b
- Initialize the parameters (weights)
- Define loss function(cost)
- Write training loop using gradient tape
  - Optimise the loss function to find the best parameter estimates using the gradients

Regression Model Training



LAB: Linear Regression on TensorFlow

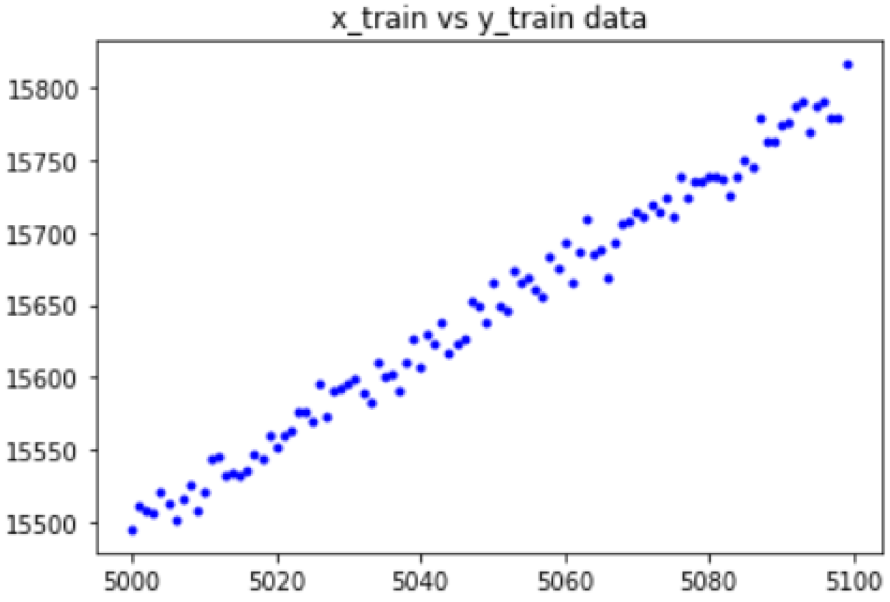
- Use code to generate the data for x and y.
- Build a regression model in TensorFlow to find W and b
- Verify the weight with the original data

Code: Linear Regression on TensorFlow

```
#This step is for data creation, x and y
import numpy as np
x_train= np.array(range(5000,5100)).reshape(-1,1)


y_train=[3*i+np.random.normal(500, 10) for i in x_train]

import matplotlib.pyplot as plt
plt.title("x_train vs y_train data")
plt.plot(x_train, y_train, 'b.')
plt.show()
```



Code: Linear Regression on TensorFlow

```
#Model y=X*W + b
#Model function
def output(x):
    return W*x + b

#Loss function Reduce mean square
def loss_function(y_pred, y_true):
    return tf.reduce_mean(tf.square(y_pred - y_true))

#Initialize Weights
W = tf.Variable(tf.random.uniform(shape=(1, 1)))
b = tf.Variable(tf.ones(shape=(1,)))

#Optimization
## Writing training/learing loop with GradienTape
learning_rate = 0.000000001
steps = 200 #epochs

for i in range(steps):
    with tf.GradientTape() as tape:
        predictions = output(x_train)
        loss = loss_function(predictions,y_train)
        dloss_dw, dloss_db = tape.gradient(loss, [W, b])
    W.assign_sub(learning_rate * dloss_dw)
    b.assign_sub(learning_rate * dloss_db)
    print(f"epoch : {i}, loss  {loss.numpy()},  W : {W.numpy()}, b  {b.numpy()}")
```

LAB: Simple ANN Model

- Data:
- Our model function:

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

- The cost function:

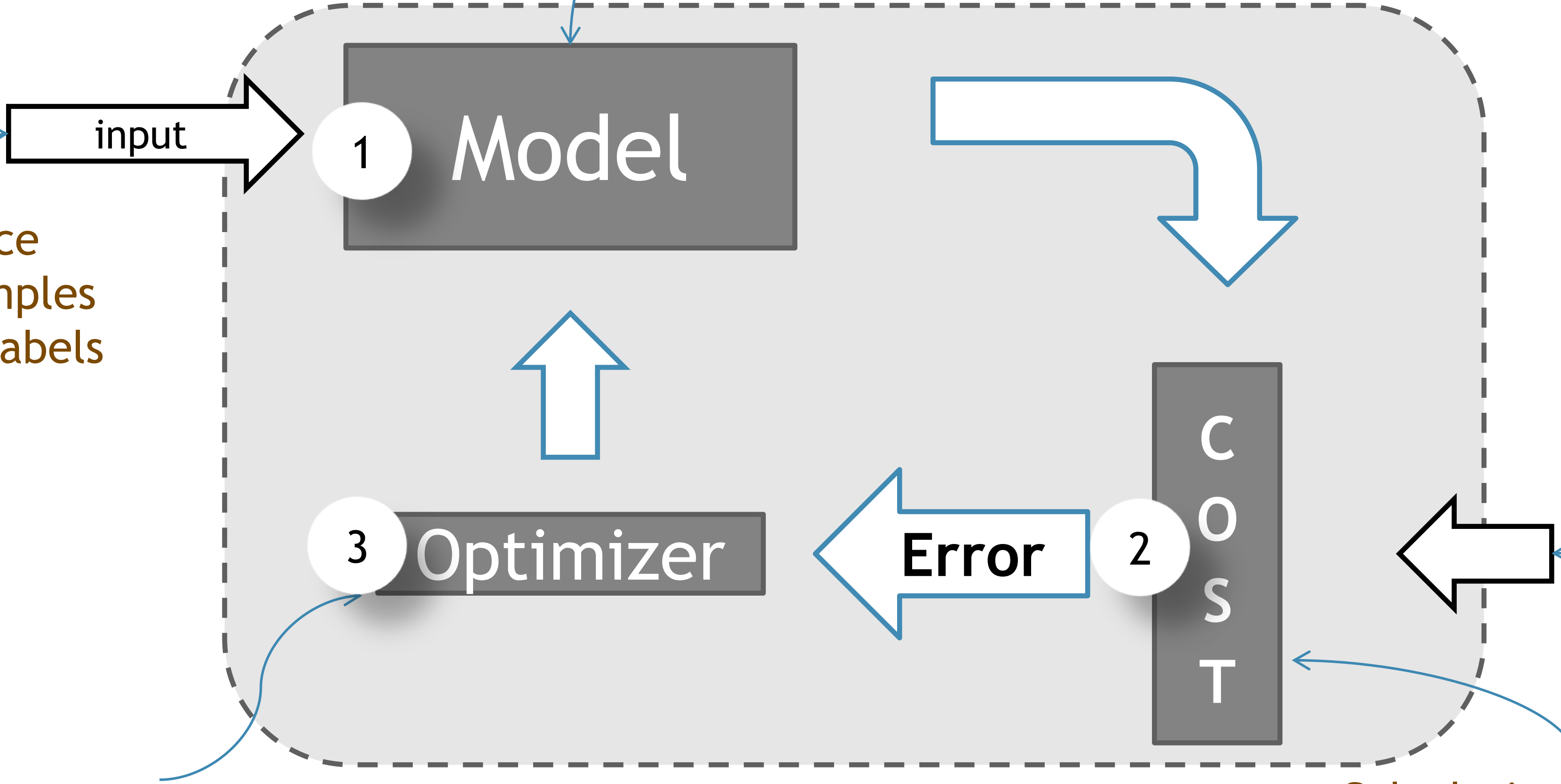
$$J(W, b) = \frac{1}{N}\sum_{i=1}^{N}(y_i - h)^2$$

- Optimizer for back propagation

Graph

Operations on variables to make predictions

To introduce input examples and their labels



labels

Training loop using GradientTape()

Calculating the error in model

Lab: ANN Model– Single Perceptron

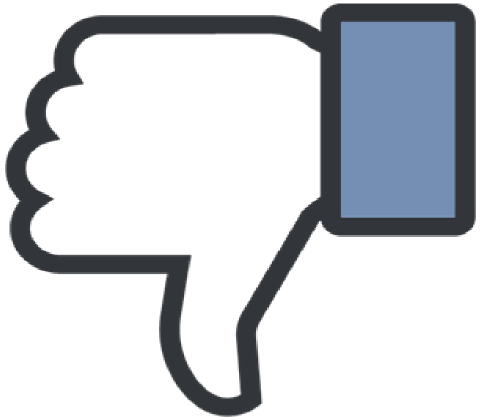
- Generate dummy x and y data
- Define our Model as Computation Graph
  - Place holders for X and y
  - W, b variables
  - Model output function
  - Cost function
  - Optimizer
- Run the graph as session, feeding data into placeholders

statinfer

TensorFlow Advantages

- 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: 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

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 tf.Keras

- Importing our Keras from tensorflow

```
from tensorflow import keras
from tensorflow.keras import layers
```

statinfer

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) = keras.datasets.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
```

```
# 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)
```

Scaling the pixel values between 0 and 1

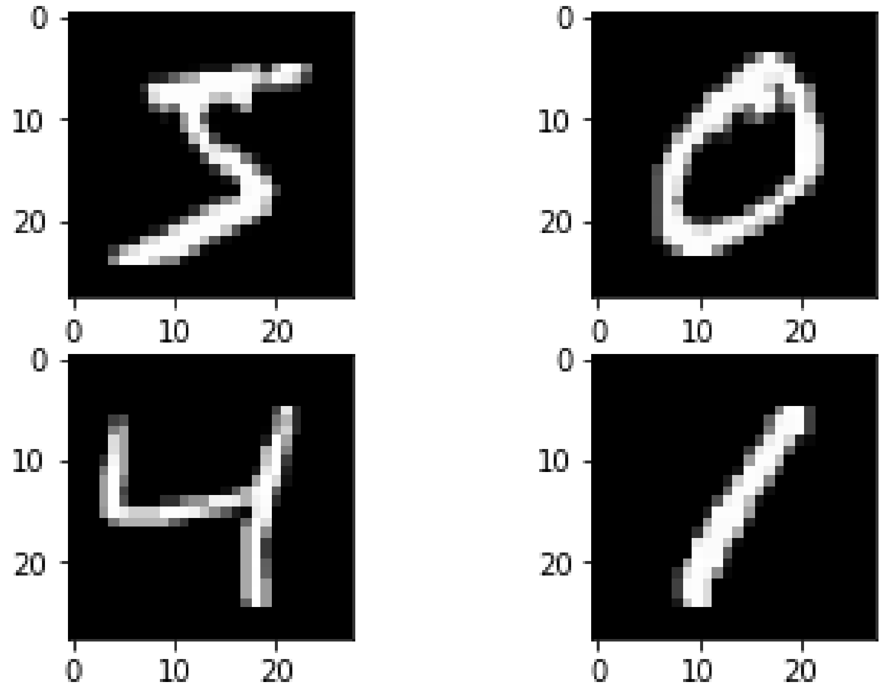
Class vector needs to be converted into binary class matrices

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

```
model = keras.Sequential()
model.add(layers.Dense(20, activation='sigmoid', input_shape=(784,)))
#Input Layer. The model needs to know what input shape it should expect. For this reason,
 the first layer in a Sequential model needs to receive information about its input shape
#Only the first need the snape information, because following layers can do automatic sha
pe inference
The dense layer is simply a layer where each unit or neuron is connected to each neuron i
n the next layer.
model.add(layers.Dense(20, activation='sigmoid'))

#In the final layer mention the output classes
model.add(layers.Dense(10, activation='softmax'))

#Model Summary
model.summary()
```

Code: MNIST on Keras

- Understanding the shape of our layers

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                15700
_________________________________________________________________
dense_1 (Dense)              (None, 20)                420
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210
=================================================================
Total params: 16,330
Trainable params: 16,330
Non-trainable params: 0
_________________________________________________________________
```

Code: MNIST on Keras

Compiling by giving: loss function, optimizer and validation matric

```
# Compiling model : we define loss function, optimizer and validation matric of our choice
model.compile(loss='mean_squared_error', metrics=['accuracy'])

# Fit method: actually running our model by supplying our input and validation data
model.fit(x_train, y_train,epochs=10)
```

Actual Training or running by feeding in the data

```
Epoch 1/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0416 - accuracy: 0.7382
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0151 - accuracy: 0.9060
Epoch 3/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0116 - accuracy: 0.9266
Epoch 4/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0101 - accuracy: 0.9359
Epoch 5/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0091 - accuracy: 0.9420
Epoch 6/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0084 - accuracy: 0.9472
Epoch 7/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0079 - accuracy: 0.9499
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0074 - accuracy: 0.9528
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0071 - accuracy: 0.9551
```

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.