Artificial Intelligence

LAB PROJECT SUBMISSION

Submitted To: Dr. Varun

Submitted By: 10211601, 102116017, 102116030, 102116110

Problem Statement:

- Credit Card Fraud Detection
- Stock Market Prediction

Description Of the problem:

- Credit Card Fraud Detection:

> Credit card fraud is an inclusive term for the unauthorized use of a payment tool, such as a credit card or debit card, to fraudulently obtain money or property.

- Stock Market Prediction:

> Stock market prediction and analysis are some of the most difficult jobs to complete. Predicting the stock trends can be benefitial in the business purposes.

Packages Used:

- Pandas

```
import pandas as pd
```

> `Pandas` is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.

```
# Syntax of read_csv()
pd.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None, header='infer', names=NoDefault.no_d
```

> `pd.read_csv()`: It is a function that convert csv(comma-separated-values) file to a dataFrame. (table data-structure with columns of different data-types).

```
# Read CSV file into DataFrame
df = pd.read_csv('courses.csv')
print(df)

#Yields below output
#  Courses   Fee     Duration  Discount
------------------------------------
#0   Spark   25000   50 Days      2000
#1  Pandas   20000   35 Days      1000
#2    Java   15000       NaN       800
#3  Python   15000   30 Days       500
#4     PHP   18000   30 Days       800
```

- Numpy

```
import numpy as np
```

> `NumPy` is an open source project that enables numerical computing with Python.

> `np.float64()`: It alters the input to 64-bit floating point number.

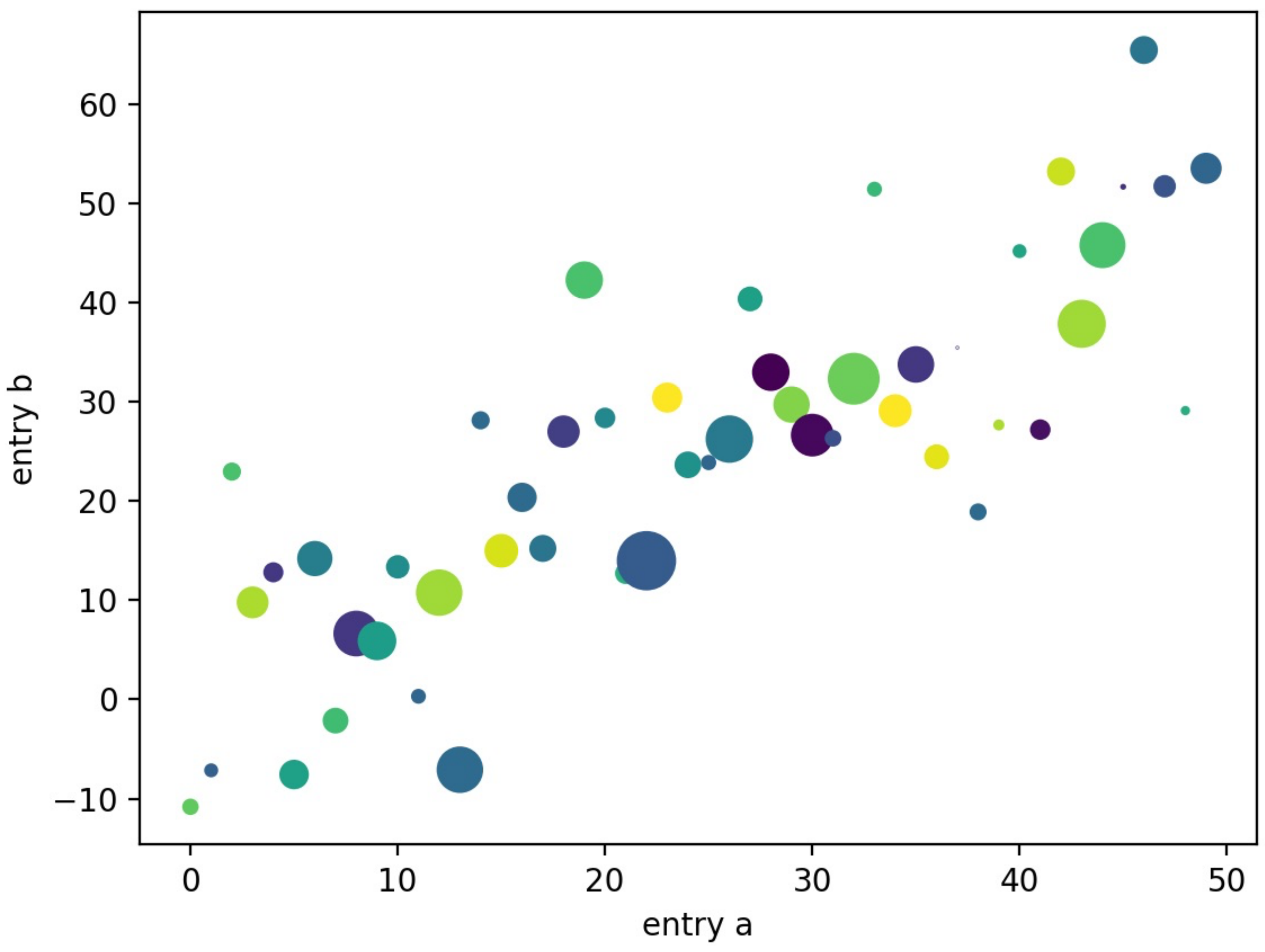
```
a = np.float64(12.23)

#dtype() gives datatype as output.
np.dtype(a)

#Output:
dtype('float64')
```

- Matplotlib

```
import matplotlib.pyplot as plt
```

> `MatPlotlib` is a comprehensive library for creating static, animated, and interactive visualizations in Python.

> `matplotlib.pyplot`: is a collection of functions that make matplotlib work like MATLAB.



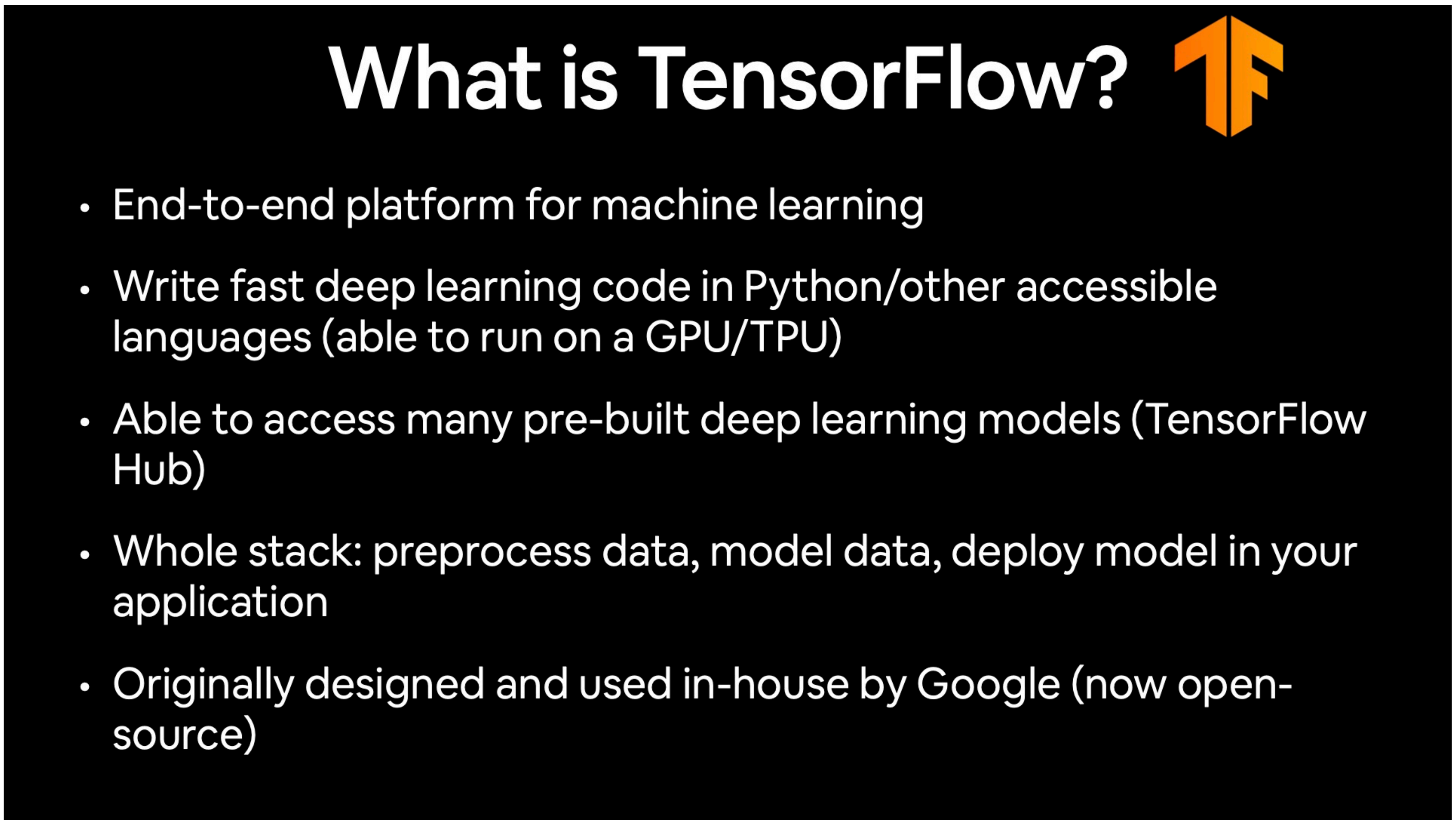
> `plt.figure()`: creates a new figure, or activate an existing figure.
>
> `plt.plot()`: plots y versus x as lines and/or markers.
>
> `plt.title()`: is used to specify title of the visualization depicted and displays the title using various attributes.
>
> `plt.xlabel()`: sets the label for the x-axis.
>
> `plt.ylabel()`: sets the label for the y-axis.
>
> `plt.show()`: displays all open figures.

- TensorFlow

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
tf.compat.v1.disable_eager_execution()
```

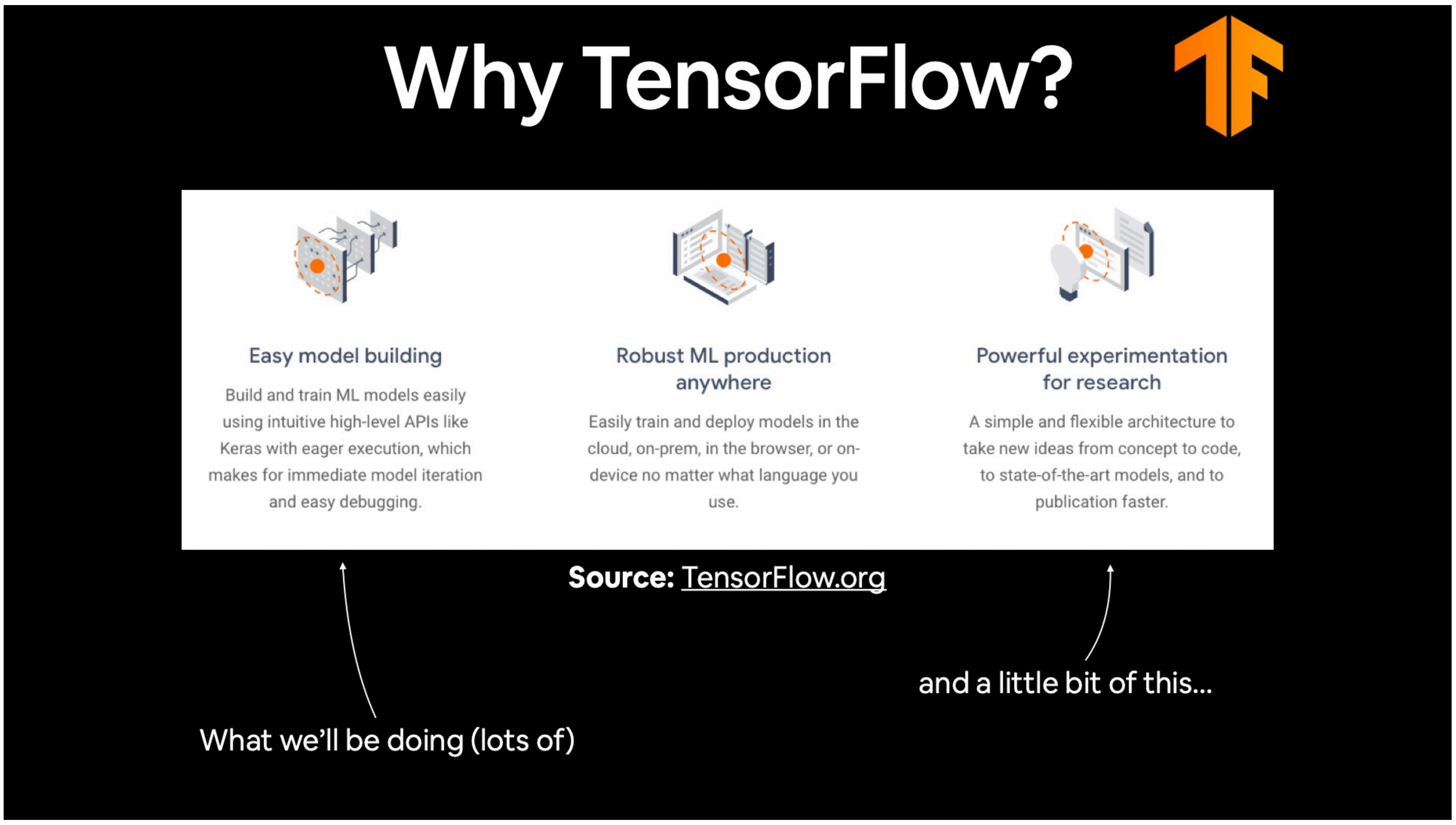
> `tf.disable_v2_behavior()`: can be called at the beginning of the program (before Tensors, Graphs or other structures have been created, and before devices have been initialized. It switches all global behaviors that are different between TensorFlow 1.x and 2.x to behave as intended for 1.x.
>
> `tf.compat.v1.disable_eager_execution()`: can only be called before any Graphs, Ops, or Tensors have been created. It can be used at the beginning of the program for complex migration projects from TensorFlow 1.x to 2.x.

What is TensorFlow & Why TensorFlow? What are Tensors, Neural Networks?



What is TensorFlow?

- End-to-end platform for machine learning
- Write fast deep learning code in Python/other accessible languages (able to run on a GPU/TPU)
- Able to access many pre-built deep learning models (TensorFlow Hub)
- Whole stack: preprocess data, model data, deploy model in your application
- Originally designed and used in-house by Google (now open-source)



Why TensorFlow?

Easy model building

Build and train ML models easily using intuitive high-level APIs like Keras with eager execution, which makes for immediate model iteration and easy debugging.

Robust ML production anywhere

Easily train and deploy models in the cloud, on-prem, in the browser, or on-device no matter what language you use.

Powerful experimentation for research

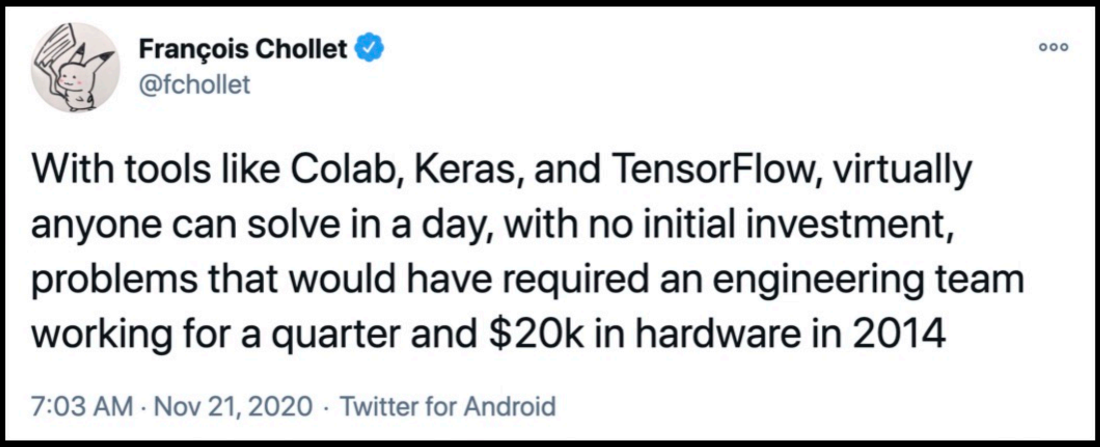
A simple and flexible architecture to take new ideas from concept to code, to state-of-the-art models, and to publication faster.

Source: TensorFlow.org

and a little bit of this...

What we'll be doing (lots of)

Why TensorFlow?

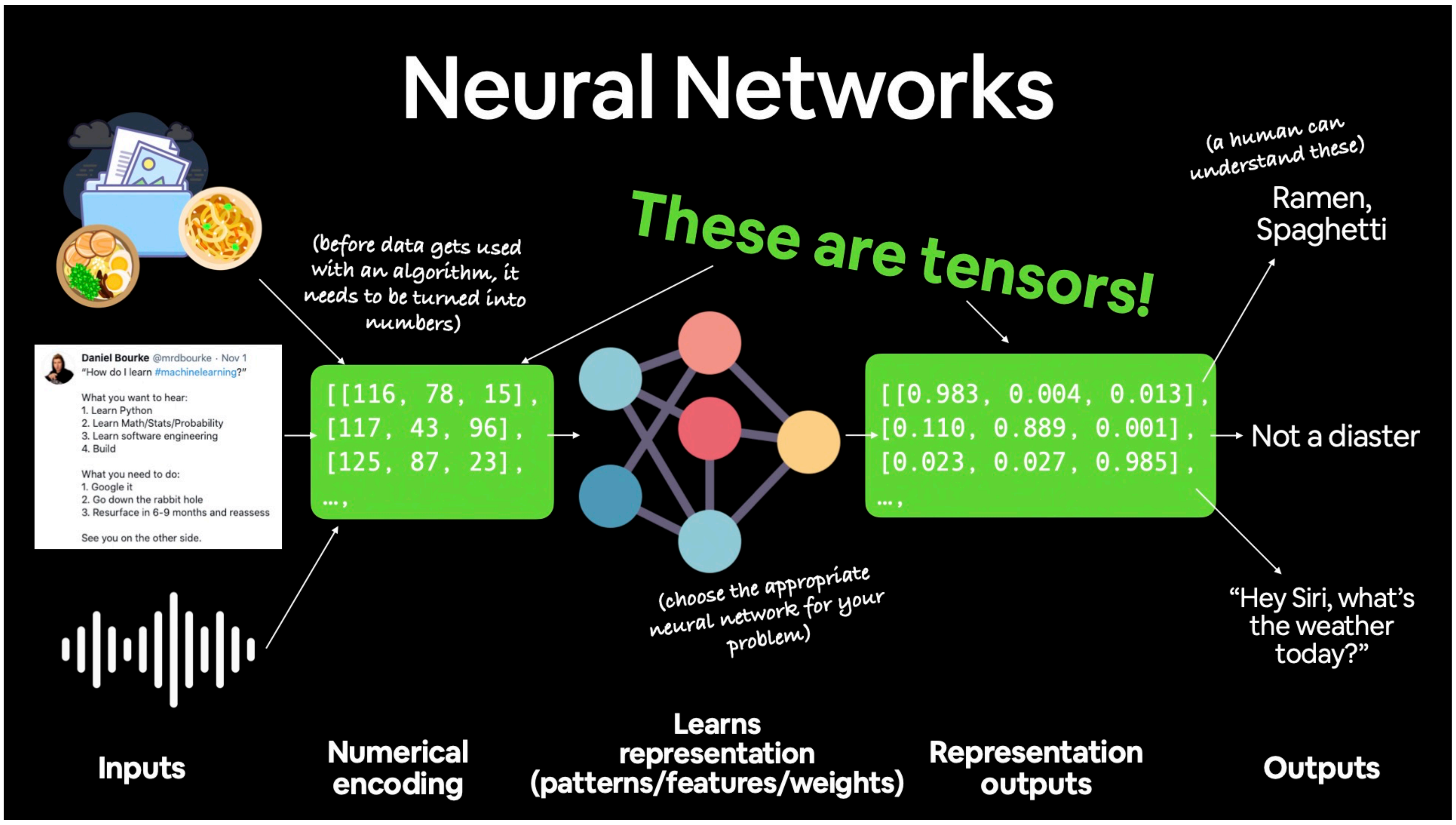

François Chollet
@fchollet

With tools like Colab, Keras, and TensorFlow, virtually anyone can solve in a day, with no initial investment, problems that would have required an engineering team working for a quarter and $20k in hardware in 2014

7:03 AM · Nov 21, 2020 · Twitter for Android

Source: @fchollet Twitter

“What is a tensor?”


Neural Networks
(a human can understand these)
Ramen, Spaghetti
These are tensors!
(before data gets used with an algorithm, it needs to be turned into numbers)
[[116, 78, 15],
[117, 43, 96],
[125, 87, 23],
...,
[[0.983, 0.004, 0.013],
[0.110, 0.889, 0.001],
[0.023, 0.027, 0.985],
...,
Not a diaster
"Hey Siri, what's the weather today?"
(choose the appropriate neural network for your problem)
Inputs
Numerical encoding
Learns representation (patterns/features/weights)
Representation outputs
Outputs

Daniel Bourke @mrdbourke · Nov 1
"How do I learn #machinelearning?"

What you want to hear:
1. Learn Python
2. Learn Math/Stats/Probability
3. Learn software engineering
4. Build

What you need to do:
1. Google it
2. Go down the rabbit hole
3. Resurface in 6-9 months and reassess

See you on the other side.

Code Explaination(Stock Prediction):

- CSV Files

```
# List of all csv filenames
GAS_TRAIN_DATA = 'CSV_Files/Gas Data Last Year.csv'
GAS_TEST_DATA = 'CSV_Files/Gas Data Last Month.csv'
GOLD_TRAIN_DATA = 'CSV_Files/Gold Data Last Year.csv'
GOLD_TEST_DATA = 'CSV_Files/Gold Data Last Month.csv'
OIL_TRAIN_DATA = 'CSV_Files/Oil Data Last Year.csv'
OIL_TEST_DATA = 'CSV_Files/Oil Data Last Month.csv'
SILVER_TRAIN_DATA = 'CSV_Files/Silver Data Last Year.csv'
SILVER_TEST_DATA = 'CSV_Files/Silver Data Last Month.csv'

# Data sets for stock we are currently assessing
current_train_data = GOLD_TRAIN_DATA
current_test_data = GOLD_TEST_DATA

# Number of data points to retrieve from csv files(varies with each
stock we assess)
NUM_TRAIN_DATA_POINTS = 266
NUM_TEST_DATA_POINTS = 22
```

> CSV Files are retrieved and initialized accordingly. For instance, we are using `GOLD_TRAIN_DATA` & `GOLD_TEST_DATA`.Number of train & test data-points are fixed.

- Functions Used:
    - `load_stock_data()`:

> *Funtion-Explaination*: Here, a CSV file with number of data points will be recieved in the function. `Price` is the final price at the end of the day. `Open` is the opening price at the start of the day. `Vol.` is the volume of stock sold at that particular day. The volume of stock sold will effect the opening prices of the stock next day, /,

```
def(load_stock_data(stock_name, num_data_points):
data = pd.read_csv(stock_name,
               skiprows=0,
               nrows=num_data_points,
               usecols=['Price', 'Open', 'Vol.'])
```

`stock_name`: Name of CSV file
`skiprows`: This parameter is use to skip passed rows in new data frame.
`nrows`: Number of entities is the number of data points extracted.
`usecols`: This parameter is Only uses the passed col[string list] to make data frame

```
# Prices of stock at the end of each day
final_prices = data['Price'].astype(str).str.replace(',','').astype(np.float64)
# Prices of stock at the beginning of each day
opening_prices = data['Open'].astype(str).str.replace(',', '').astype(np.float64)
# Volume of stock exchanged throughout the day
volumes = data['Vol.'].str.strip('MK').astype(np.float64)
return final_prices, opening_prices, volumes
```

> As the CSV data contain `,` in the prices e.g. 1,234. So, we are formatting it by first changing it to float and deleting `,` and finally changing it to `float64`. Same for other attributes, all the extra non-float symbols are removed.

- `calculate_price_differences()`:

```
def calculate_price_differences(final_prices, opening_prices):
price_differences = []
   for d_i in range(len(final_prices) - 1):
     price_difference = opening_prices[d_i + 1] - final_prices[d_i]
     price_differences.append(price_difference)
return price_differences
```

> This function stores the difference(opening_prices - final_prices) to the `price_difference = []`

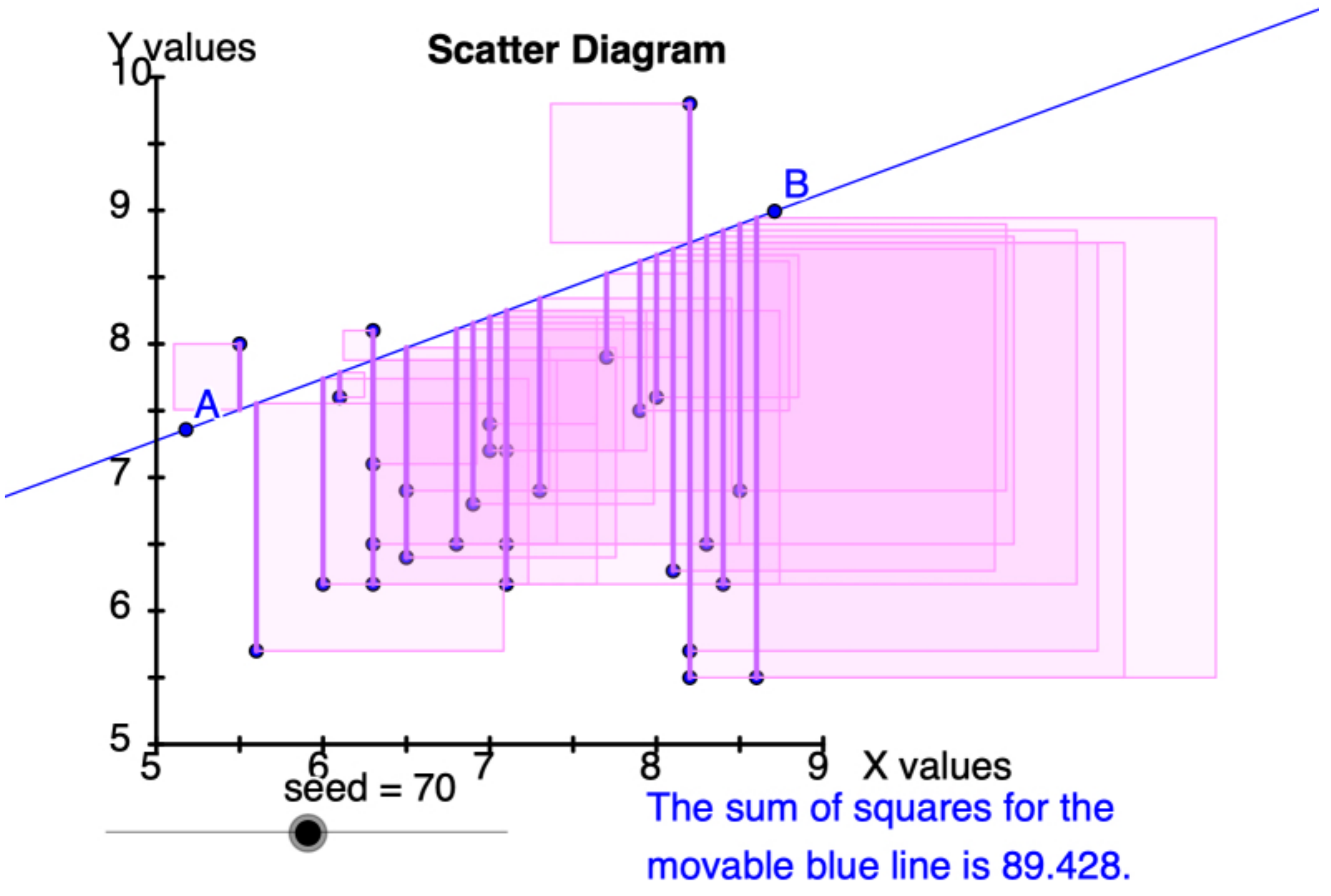
- `calculate_accuracy()`:

```
def calculate_accuracy(expected_values, actual_values):
num_correct = 0
for a_i in range(len(actual_values)):
   if actual_values[a_i] < 0 < expected_values[a_i]:
       num_correct += 1
   elif actual_values[a_i] > 0 > expected_values[a_i]:
       num_correct += 1
return (num_correct / len(actual_values)) * 100
```

> In the above function, `expected_values` takes `price_difference = opening_prices[] - final_prices[]` as the input and `actual_values` takes model generated input.

- Linear Regression:

> *Linear Regression* is one of the simplest machine learning models.(`Y = MX + C`)
>
> In other words, a line is generated considering that all the data points satisfy the line with least `Sum of Sqaures`. This is done by adjusting the `M` & `C`.



- Training a Linear Model `Y = MX + C`:

> Take `X` values and expected `Y` values

> Start with guess for `M` and `B` and measure loss

> Program is executed in order to adjust the `M` and `B` and minimize the loss based on the input

Finally, the model will fit a good line through data and will be able to predict correct `Y` values given `X` input.

- Calculating Loss:

```
# Loss function based on the difference between actual and expected outputs
loss = tf.reduce_sum(tf.square(y - y_predicted))
```

> `tf.square()`: It calculated the square of the input.

> `tf.reduce_sum()`: It calculates the sum of all elements along the default given axis of the tensor.

```
# For performing operations with Tensors td.Session is executed
sess = tf.Session()
x_train = [1, 2, 3, 4]
sess.run(tf.reduce_sum(x_train))
```

```
Output: 10
```

- Minimizing Loss:

```
# Optimizer aimed at minimizing loss by changing W and b
optimizer = tf.train.AdamOptimizer(LEARNING_RATE).minimize(loss)
for _ in range(NUM_EPOCHS):
    # Run the optimizer which will allow it to change the values of W and b to minimize loss
    session.run(optimizer, feed_dict={x: train_volumes, y_predicted: train_price_differences})
```

> `Learning_Rate`: It is the change made to slope `M` and intercept `C`. For example `Learning_Rate = 0.01` means `M` and `C` will be altered by 0.01 each time.

> `Epochs`: Tecnically, it is the time taken by training dataset around the algorithm. In our case, more the number of epochs will effect the alterations made to `M` and `C`.

> `tf.train.AdamOptimizer(LEARNING_RATE).minimizer(loss)` It is a kind of optimizer that uses Adam algorithm. Also, the `minimizer(loss)` takes `loss` as the input which means `M` and `C` will be altered to minimize the `loss` variable.

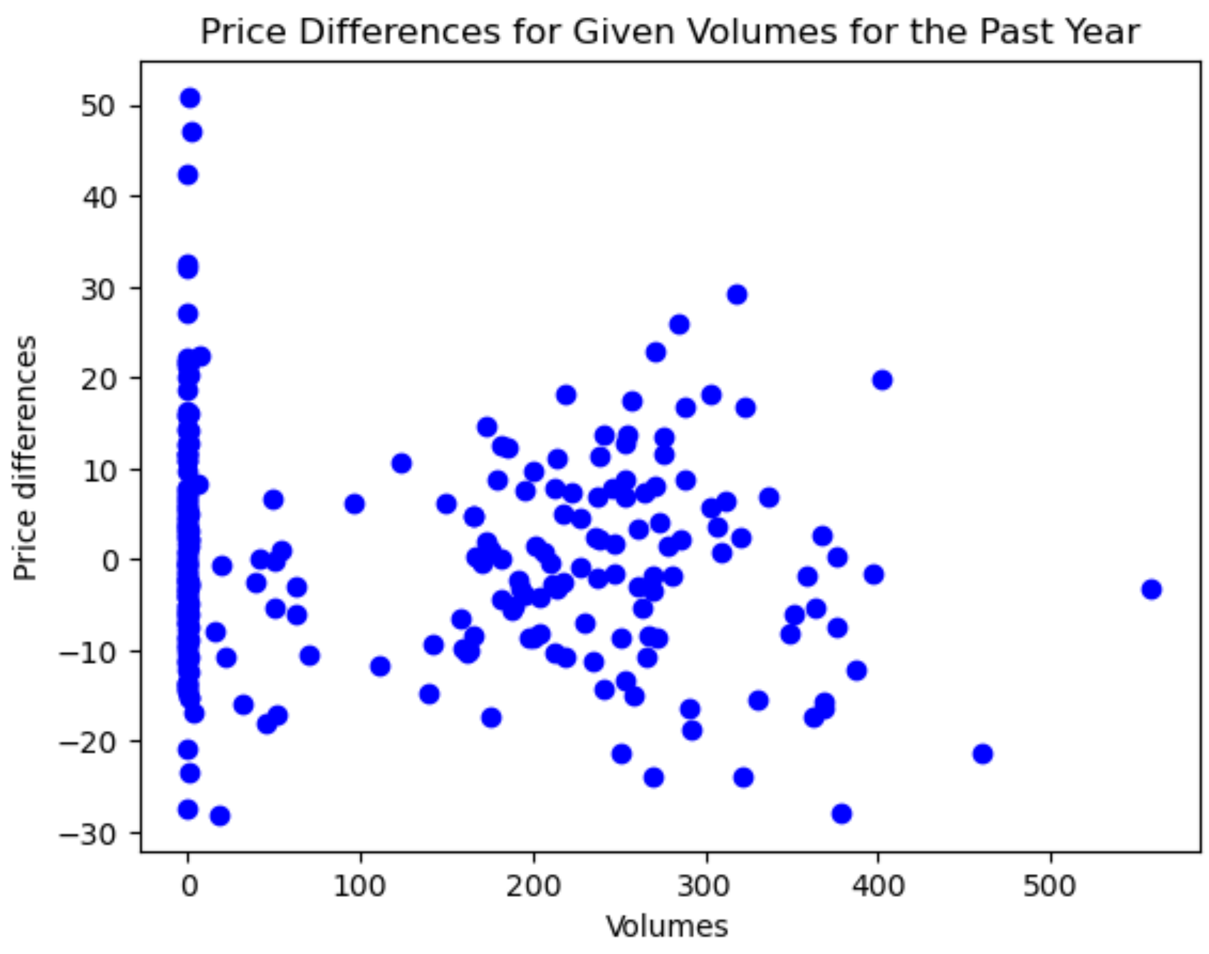
- Calculating Accuracy:

> Using the test data-points, model is judged by calculating the accuracy. Each time it yields a deviation from the test data-points we increment a counter and finally calulate the *Accuracy*.

```
Accuracy of model: 61.90%
```

- Plotting the Graph:

```
plt.figure(1)
plt.plot(train_volumes, train_price_differences, 'bo')
plt.title('Price Differences for Given Volumes for the Past Year')
plt.xlabel('Volumes')
plt.ylabel('Price differences')
plt.show()
```



Code Explaination(Credit Card Fraud Detection):

- Splitting data into 4 sets:
  1. Shuffle/randomize data
  2. One-hot encoding
  3. Normalize
  4. Splitting up X/Y values
  5. Convert data_frames to numpy arrays (float32)
  6. Splitting the final data into X/Y train/test

> - Shuffle/randomize data:
>
> ```
> shuffled_data = credit_card_data.sample(frac=1)
> ```
>
> Here, the sample is the whole data(`frac = 1`) is shuffled.This is because if let's assume the starting 10000 data-points have same trend and the last 2000 data-points have different but same trend then the output will be effected. This is because our training model will get steep bend towards a single trend.

> - One-hot Encoding:
>
> ```
> Fruit  Number Of Items  Price
> Apple.       1            5
> Mango        2            10
> Banana       1            15
> Orange       3            20
> <!-- The above code is encoded as: -->
> apple    mango    orange    price
> 1          0         0        5
> 0          1         0        10
> ```

```
1          0         0        15
0          0         1        20
```

- Normalizing the data: When we the data to be converted to such a value that it ranges between certain numbers i.e 1 and 0. This is because we want the data or piece of information to be rational rather than deviating.

```
<!-- We use the following formula for conversion: -->
V' = (V - min(A)) / (max(A) - min(A))
V' = New Normalized Value
V = Old value
A = Attribute which is under consideration
Set = [2, 4, 5, 8, 1, 13] #Un-Normalized Data
max(Set) = 13
min(Set) = 1
Normalized_data = [0.0833, 0.25, 0.3333, 0.5888, 0, 1]
```

- Splitting up X/Y values:

```
   <!-- Values excluding Classes -->
   df_X = normalized_data.drop(['Class_0', 'Class_1'], axis=1)
   <!-- Only Classes -->
   df_y = normalized_data[['Class_0', 'Class_1']]
```

- Convert data_frames to numpy arrays (float32):

```
  ar_X, ar_y = np.asarray(df_X.values, dtype='float32'), np.asarray(df_y.values, dtype='floa
```

- Splitting the final data into X/Y train/test:

```
 <!-- Allocate first 80% of data into training data and remaining 20% into testing data -->
train_size = int(0.8 * len(ar_X))
(raw_X_train, raw_y_train) = (ar_X[:train_size], ar_y[:train_size])
(raw_X_test, raw_y_test) = (ar_X[train_size:], ar_y[train_size:])
```

- Calculating Percentage Of Fradulent Transactions:

In the above given expaination, there were two classes 0 and 1

`0 = Legitimate Transaction` and `1 = Fraudulent Transaction`

```
 <!-- Gets a percent of fraud vs legit transactions (0.0017% of transactions are fraudulent) !-->
count_legit, count_fraud = np.unique(credit_card_data['Class'], return_counts=True)[1]
fraud_ratio = float(count_fraud / (count_legit + count_fraud))
print('Percent of fraudulent transactions: ', fraud_ratio)
```

`np.unique` calculates unique values from the input set.

```
 <!-- Applies a logit weighting of 578 (1/0.0017) to fraudulent transactions to cause model to pa
weighting = 1 / fraud_ratio
raw_y_train[:, 1] = raw_y_train[:, 1] * weighting
```

> `weighting`: is a large number which help model differentiating between `Legitimate` and `Fradulent` transaction.

Building Layers in a Neural Network:

- ***Neural Network*** is a network consisting of a large number of `Artificial Neurons`. Components of a Artificial Neuron are:

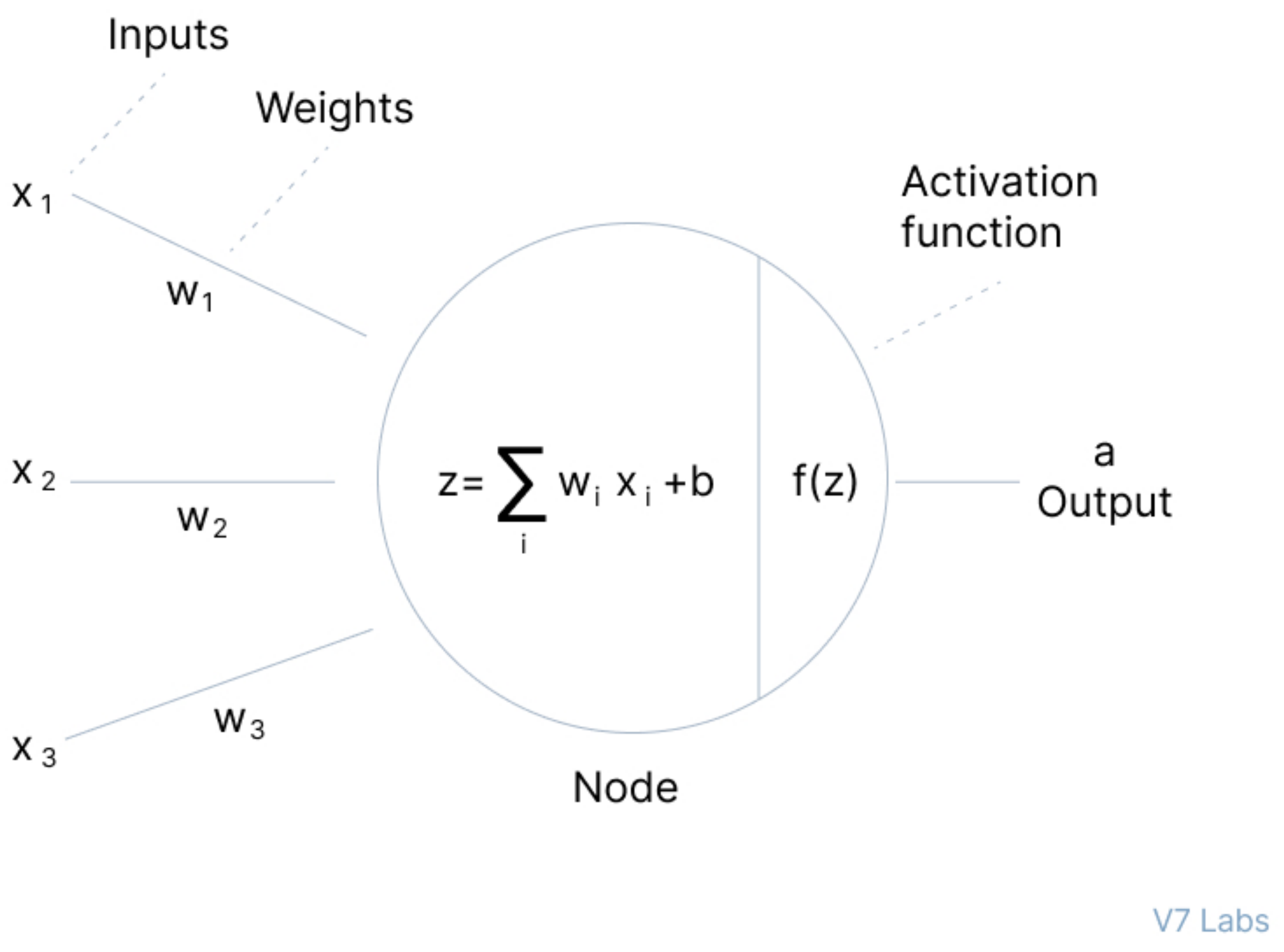
  1. ***Inputs***: are the set of values for which we need to predict a output value. They can be viewed as features or attributes in a dataset.

  2. ***Weights***: are the real values that are attached with each input/feature and they convey the importance of that corresponding feature in predicting the final output.

  3. ***Bias***: is used for shifting the activation function towards left or right, you can compare this to y-intercept in the line equation.

  4. ***Summation Function***: The work of the summation function is to bind the weights and inputs together and calculate their sum.

  5. ***Activation Function***: decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations.

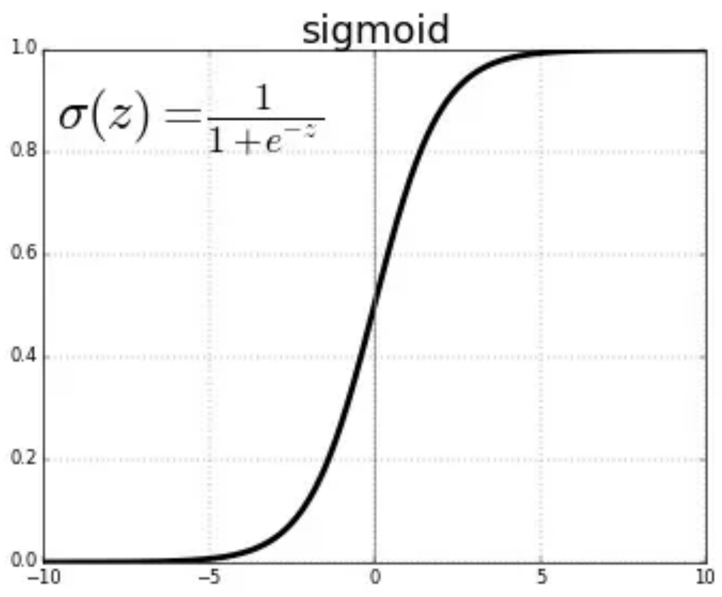
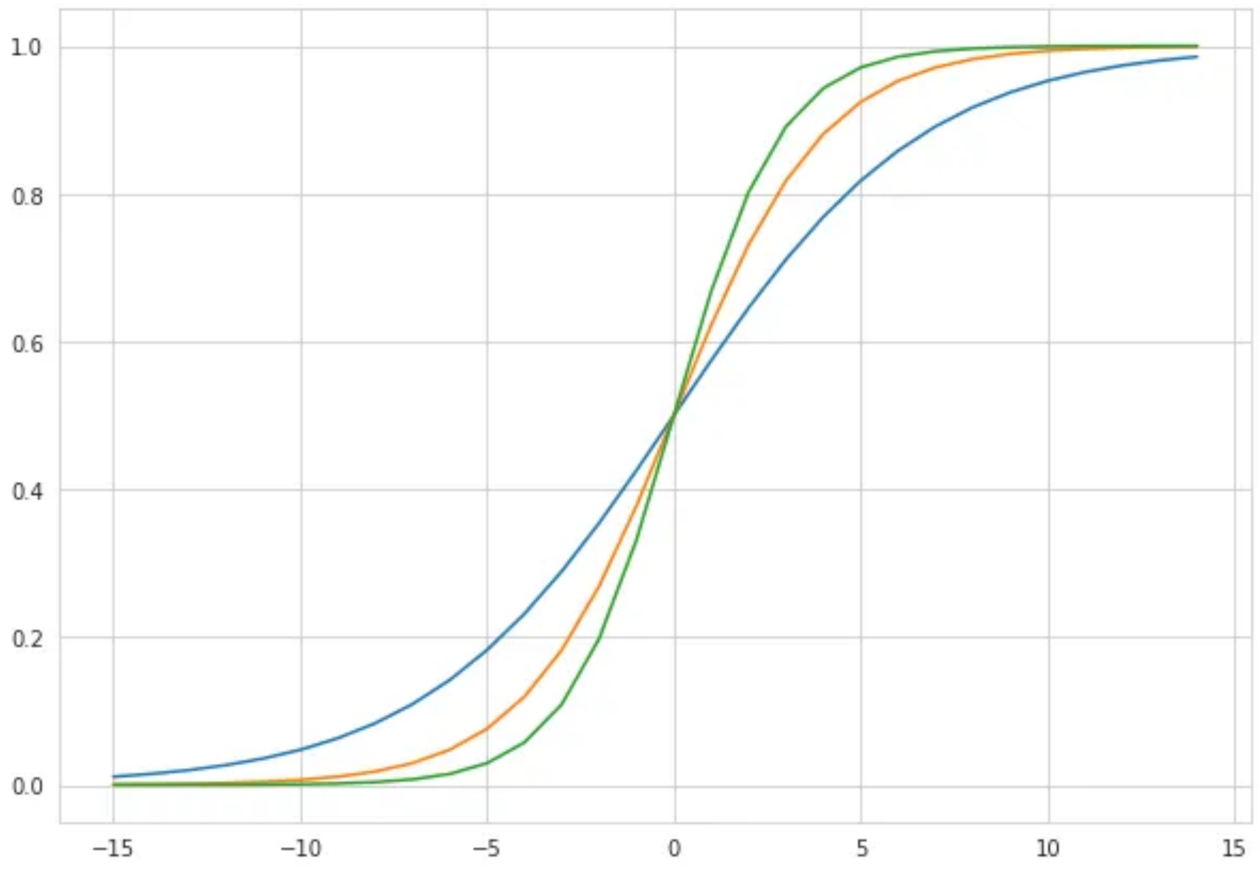


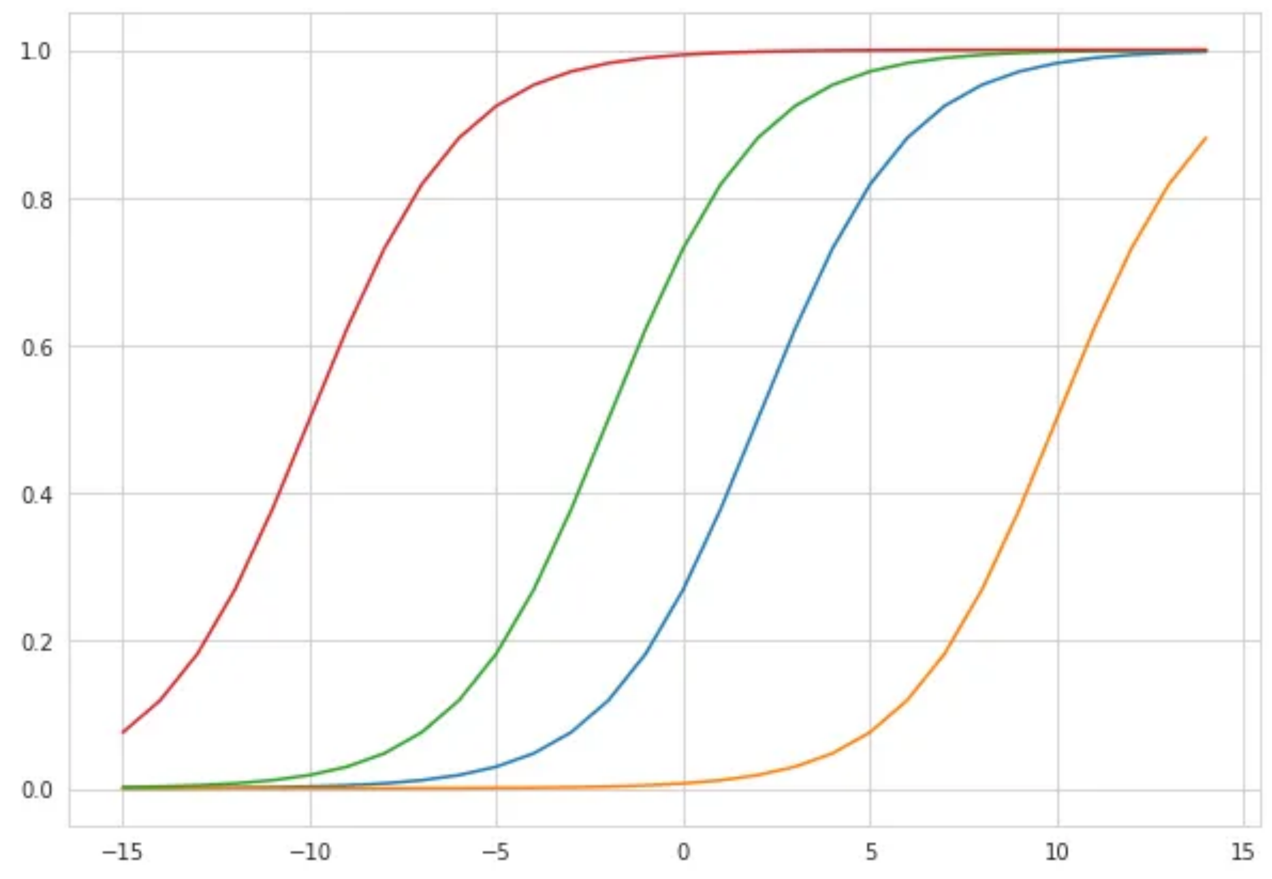
Why we need `Weight`, `Bias` and `Activation-Function`?

- Weight

> We know that each input has different effect on the output. Some attributes will have more effect some have less. So, to differentiate within attributes they are characterised by their Weights. More is the weight, more it will effect the output.

- Bias

> Bias is used for shifting the activation function towards the left of right.
>
> *let us consider a sigmoid activation function to demonstrate the use of bias, we can represent the sigmoid activation function with the mathematical expression as*
>
> 
>
>
> In our case, our output ranges between 0 and 1 so Sigmoid-Function suits it the most. As the summation function is: `WX + B` where B is the bias. It helps the activation function(Sigmoid-Function) to shift towards right or left
>
> - Sigmoid Function with fixing `B` and changing `X`
>
> 
>
> - Sigmoid Function with changing `B` and changing `X`



- Activation-Function

> Think of what will happen if there is no activation function?
>
> As, going forward from ***Input Layer*** to ***Hidden Ist Layer*** we are calculating ***Summation Function***, i.e. `WX + B` which is always a linear graph. It is not always the case that we are always fitting our data-points linearly so to give our model a different fit / regression, we introduce an `Activate Function` at the end of node.

- Applying the components & functions:
  - Adding Layers:

```
# 30 cells for the input
input_dimensions = ar_X.shape[1] # 2 cells for the output
output_dimensions = ar_y.shape[1] # 100 cells for the 1st layer
num_layer_1_cells = 100
# 150 cells for the second layer
num_layer_2_cells = 150

# We will use these as inputs to the model when it comes time to
train it (assign values at run time)
X_train_node = tf.placeholder(tf.float32, [None, input_dimensions], name='X_train') y_train_node = tf.pla

# We will use these as inputs to the model once it comes time to test it
X_test_node = tf.constant(raw_X_test, name='X_test') y_test_node = tf.constant(raw_y_test, name='y_test')

# First layer takes in input and passes output to 2nd layer
weight_1_node = tf.Variable(tf.zeros([input_dimensions, num_layer_1_cells]), name='weight_1') biases_1_no

# Second layer takes in input from 1st layer and passes output to 3rd layer
weight_2_node = tf.Variable(tf.zeros([num_layer_1_cells, num_layer_2_cells]), name='weight_2') biases_2_n

# Third layer takes in input from 2nd layer and outputs [1 0] or [0 1] depending on fraud vs legit
weight_3_node = tf.Variable(tf.zeros([num_layer_2_cells, output_dimensions]), name='weight_3')
```

```
biases_3_node = tf.Variable(tf.zeros([output_dimensions]), name='biases_3')

# Function to run an input tensor through the 3 layers and output a tensor that will give us a fraud/leg
# Each layer uses a different function to fit lines through the data and predict whether a given input
# Result in a fraudulent or legitimate transaction

def network(input_tensor):
# Sigmoid fits modified data well
layer1 = tf.nn.sigmoid(tf.matmul(input_tensor, weight_1_node) + biases_1_node)
# Dropout prevents model from becoming lazy and over confident
layer2 = tf.nn.dropout(tf.nn.sigmoid(tf.matmul(layer1, weight_2_node) + biases_2_node), 0.85)
# Softmax works very well with one hot encoding which is how results are outputted
layer3 = tf.nn.softmax(tf.matmul(layer2, weight_3_node) + biases_3_node)
return layer3

# Used to predict what results will be given training or testing input data
# Remember, X_train_node is just a placeholder for now. We will enter values at run time
y_train_prediction = network(X_train_node)
y_test_prediction = network(X_test_node)
```

- Drop-Out

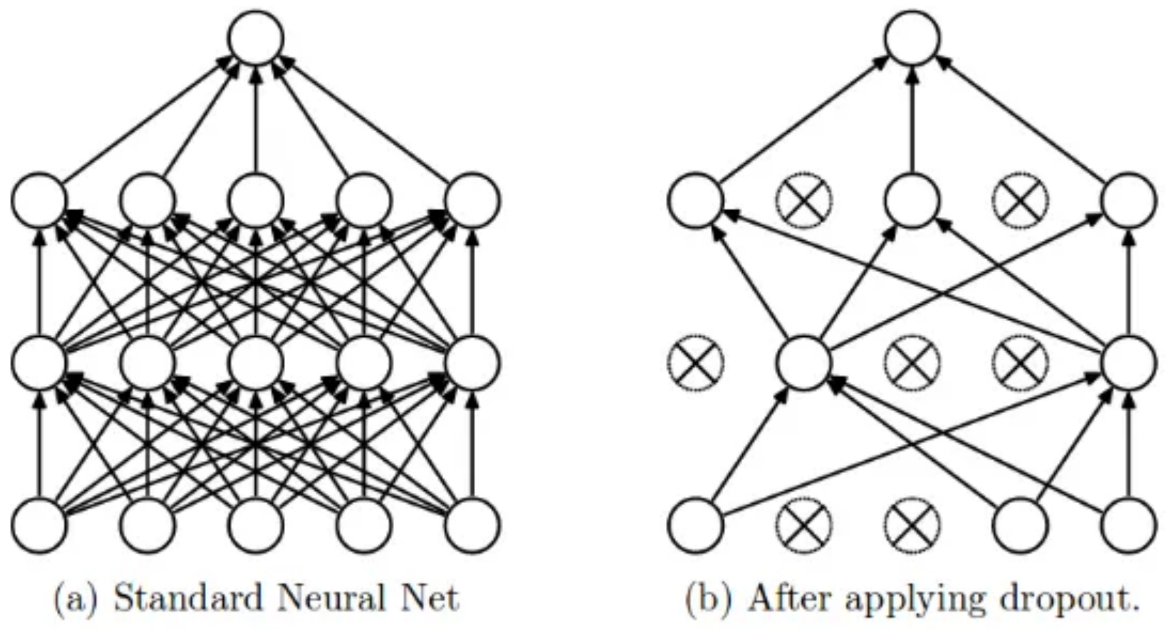
> *Consider a scenario where there are two students who are taught by same teacher. `Student A` learns eerything by heart(Cram) and the other `Student B` learns by generalising the concept.*

> Now , there was test where the questions were `based on what was taught`. `Student A = 90%` and `Student B = 72%`

> Again, there was a test but `concept based` and the result was `Student A = 65%` and `Student B = 78%`

> This is how Generalization is useful.

> Similarly, our model while being trained learns random fluctuations in the data, not the key concept and this might not predict the output when fed with any outside data. This is called ***Overfitting***.
> So, each time an input is passed while training we drop-out certain nodes in order to build `Co-Adaptation`.



Optimization and Accuracy Calculation:

```
# Cross entropy loss function measures differences between actual output and predicted output
cross_entropy = tf.losses.softmax_cross_entropy(y_train_node, y_train_prediction)

# Adam optimizer function will try to minimize loss (cross_entropy) but changing the 3 layers' variable # learning rate of 0
optimizer = tf.train.AdamOptimizer(0.005).minimize(cross_entropy)

# Function to calculate the accuracy of the actual result vs the predicted result
def calculate_accuracy(actual, predicted):
actual = np.argmax(actual, 1)
predicted = np.argmax(predicted, 1)
return (100 * np.sum(np.equal(predicted, actual)) / predicted.shape[0])

num_epochs = 100

import time
with tf.Session() as session:
    tf.global_variables_initializer().run()
    for epoch in range(num_epochs):



          start_time = time.time()
          _, cross_entropy_score = session.run([optimizer, cross_entropy, feed_dict={X_train_node: raw_X_train, y_train_node
          if epoch % 10 == 0:
              timer = time.time() - start_time
              print('Epoch: {}'.format(epoch), 'Current loss: {0:.4f}'.format(cross_entropy_score), 'Elapsed time: {0:.2f} s
              final_y_test = y_test_node.eval()
              final_y_test_prediction = y_test_prediction.eval()
              final_accuracy = calculate_accuracy(final_y_test, final_y_test_prediction)
              print("Current accuracy: {0:.2f}%".format(final_accuracy))

      final_y_test = y_test_node.eval()
      final_y_test_prediction = y_test_prediction.eval()
      final_accuracy = calculate_accuracy(final_y_test, final_y_test_prediction)
      print("Final accuracy: {0:.2f}%".format(final_accuracy))

final_fraud_y_test = final_y_test[final_y_test[:, 1] == 1]
final_fraud_y_test_prediction = final_y_test_prediction[final_y_test[:, 1] == 1]
final_fraud_accuracy = calculate_accuracy(final_fraud_y_test, final_fraud_y_test_prediction)
print('Final fraud specific accuracy: {0:.2f}%'.format(final_fraud_accuracy))
```

Output:

```
Percent of fraudulent transactions:  0.001727485630620034
Epoch: 0 Current loss: 1.3735 Elapsed time: 0.33 seconds
Current accuracy: 99.82%
Epoch: 10 Current loss: 1.3726 Elapsed time: 0.27 seconds
Current accuracy: 4.83%
Epoch: 20 Current loss: 1.3423 Elapsed time: 0.26 seconds
Current accuracy: 66.99%
Epoch: 30 Current loss: 1.2015 Elapsed time: 0.27 seconds
Current accuracy: 98.11%
Epoch: 40 Current loss: 1.0138 Elapsed time: 0.26 seconds
Current accuracy: 97.90%
Epoch: 50 Current loss: 0.9110 Elapsed time: 0.26 seconds
Current accuracy: 99.64%
Epoch: 60 Current loss: 0.8561 Elapsed time: 0.26 seconds
Current accuracy: 99.78%
Epoch: 70 Current loss: 0.8316 Elapsed time: 0.26 seconds
Current accuracy: 99.73%
Epoch: 80 Current loss: 0.8157 Elapsed time: 0.26 seconds
Current accuracy: 99.67%
Epoch: 90 Current loss: 0.8030 Elapsed time: 0.26 seconds
Current accuracy: 99.57%
Final accuracy: 99.52%
Final fraud specific accuracy: 83.81%
```