Team: 14
Professor: Yugyung Lee

Name: Sneha Mishra
Class ID: 19
Email: smccr@mail.umkc.edu
MyGitHub

Technical Partner:
Name: Raju Nekadi
Class ID: 20
Email: rn8mh@mail.umkc.edu
GitHub

Objective

Linear regression uses the general linear equationY=b0+b1*X where Y is a continuous dependent variable(for example predicting the prices of houses). Logistic regression is another generalized linear model procedure using the same basic formula, but instead of the continuousY, it is regressing for the probability of a categorical outcome (for example predicting Email as spam or NOT spam. Also, another example can be using MNIST dataset in which the labels of the dataset are 10 different categories. So, the problem will be predicting the correct category of the input dataset).

Features

- Implement the Linear Regression.
- Implement Logistic Regression.
- Show the graph in TensorBoard.
- Report the accuracy changes, by changing the hyper parameter.
- Understanding the difference between Linear Regression and Logistic Regression.

Steps:

Part 1:

Problem Statement:

Implement the Linear Regression with any data set of your choice except the datasets being discussed in the class
a. Show the graph in TensorBoard

b. Plot the loss and then change the below parameter and report your view how the result changes in each case

- a.learning rate
- b.batch size
- c.optimizer
- d.activation function

Objective:

The objective for the first problem is analyse the Linear Regression Model on any Sample given Dataset.

Show the graph in TensorBoard Plot the loss and then change the below parameter and report your view how the result changes in each case a.learning rate b.batch size c.optimizer d.activation function

Approach:

In order to implement the Linear Regression model we have take Abalone Dataset from UCI Data Repository. Using this dataset we can predict the age of abalone from physical measurement. We have used the various Deep Learning libraries and packages like Keras,Tensorboard, optimizer and Activation functions to evaluate the model.

More details are given in workflow section.

Dataset used

Dataset can be found here

Code:

The source code can be found here

Output/ Workflow:

A. Loaded the dataset and creating the Panda dataframe.

```
# Linear Regression using Abalone dataset to predict Rings Value using Keras

import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout,LeakyReLU
from sklearn.model_selection import train_test_split
from keras import metrics
from keras.optimizers import Adam, RMSprop
from keras.callbacks import TensorBoard


# Creating the Dataframe using abalone.csv

abalone_data = pd.read_csv('abalone.csv')
```

B.Created the Age Column in abalone_data panda dataframe.

```
# As per problem description which require as to compute Age , lets first compute the target of problem 'Age'
# and assign it to dataset abalone_data.  Age = Rings + 1.5

abalone_data['Age'] = abalone_data['Rings']+1.5
abalone_data.drop('Rings', axis=1, inplace=True)
```

C. Feature Statistic on given dataset

```
# Feature wise statistics using builtin tools

print(abalone_data.columns)
print(abalone_data.head())

print(abalone_data.info())
print(abalone_data.describe())

# Key Insights
# All Feature are numeric except sex
# no Missing value in dataset
```

As we can see all the Features are Numeric except Sex. No Missing Data present.

D. Creating X,Y train and validation dataframe.

```
abalone_linear_regression
             Length     Diameter    ...     Shell_Weight          Age
count   4177.000000  4177.000000    ...      4177.000000  4177.000000
mean       0.523992     0.407881    ...         0.238831    11.433684
std        0.120093     0.099240    ...         0.139203     3.224169
min        0.075000     0.055000    ...         0.001500     2.500000
25%        0.450000     0.350000    ...         0.130000     9.500000
50%        0.545000     0.425000    ...         0.234000    10.500000
75%        0.615000     0.480000    ...         0.329000    12.500000
max        0.815000     0.650000    ...         1.005000    30.500000

[8 rows x 8 columns]
      Length  Diameter      ...       Viscera_Weight  Shell_Weight
1376   0.620     0.510      ...               0.2385         0.390
1225   0.345     0.255      ...               0.0370         0.050
2722   0.375     0.275      ...               0.0545         0.066
3387   0.545     0.410      ...               0.1960         0.310
2773   0.580     0.465      ...               0.2155         0.250
```

E. Normalizing the train and validation dataframe.

```
# Creating X and y

feature_cols = ['Length', 'Diameter', 'Height', 'Whole_Weight', 'Shucked_Weight', 'Viscera_Weight', 'Shell_Weight']

X = abalone_data[feature_cols]
y = abalone_data['Age']

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.3, random_state=0)

print(X_train.head())
print(y_valid.head())
```

F. Model
Definition

```python
# Normalization
def norm_stats(df1, df2):
    dfs = df1.append(df2)
    minimum = np.min(dfs)
    maximum = np.max(dfs)
    mu = np.mean(dfs)
    sigma = np.std(dfs)
    return (minimum, maximum, mu, sigma)

def z_score(col, stats):
    m, M, mu, s = stats
    df2 = pd.DataFrame()
    for c in col.columns:
        df2[c] = (col[c]-mu[c])/s[c]
    return df2

stats = norm_stats(X_train, X_valid)
arr_x_train = np.array(z_score(X_train, stats))
arr_y_train = np.array(y_train)
arr_x_valid = np.array(z_score(X_valid, stats))
arr_y_valid = np.array(y_valid)
print('Training shape:', arr_x_train.shape)
print('Validation',arr_y_train.shape)
print('Training samples: ', arr_x_train.shape[0])
print('Validation samples: ', arr_x_valid.shape[0])
```

G. Defining the Number Epoch , batch size and defining tensorboard logic for graph.

```
# Defining the Model

def model(x_size, y_size):
    t_model = Sequential()
    t_model.add(Dense(100, activation="tanh", input_shape=(x_size,)))
    t_model.add(Dropout(0.1))
    t_model.add(Dense(50, activation="relu"))
    t_model.add(Dense(20, activation="relu"))
    t_model.add(Dense(y_size))
    t_model.compile(loss='mean_squared_error', optimizer=RMSprop(lr=0.004), metrics=[metrics.mae])
    return t_model


model = model(arr_x_train.shape[1], 1)
model.summary()
```

H. Fit the Model and calculating the Train and validation score .

```
# Epoch and Batch Size
epochs = 800
batch_size = 128

# Tensoboard Logic


LOG_DIR = os.getcwd()
tensorboard = TensorBoard(log_dir='LOG_DIR', histogram_freq=0,
                          write_graph=True, write_images=True)
```

I.At the last we have plotted the loss graph

```
                         write_graph=True, write_images=True)

# Fit the Model

history = model.fit(arr_x_train, arr_y_train, batch_size=batch_size, epochs=epochs, shuffle=True, verbose=2,
                    callbacks=[tensorboard], validation_data=(arr_x_valid, arr_y_valid),)
train_score = model.evaluate(arr_x_train, arr_y_train, verbose=0)
valid_score = model.evaluate(arr_x_valid, arr_y_valid, verbose=0)

print('Train MAE: ', round(train_score[1], 4), ', Train Loss: ', round(train_score[0], 4))
print('Val MAE: ', round(valid_score[1], 4), ', Val Loss: ', round(valid_score[0], 4))

print(os.getcwd())
```

```python
# Function to Plot the Loss

def plot_loss(h):
    plt.figure()
    plt.plot(h['loss'])
    plt.plot(h['val_loss'])
    plt.title('Training vs Validation Loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Validation'])
    plt.draw()
    plt.show()
    return


plot_loss(history.history)
```

Evaluation:

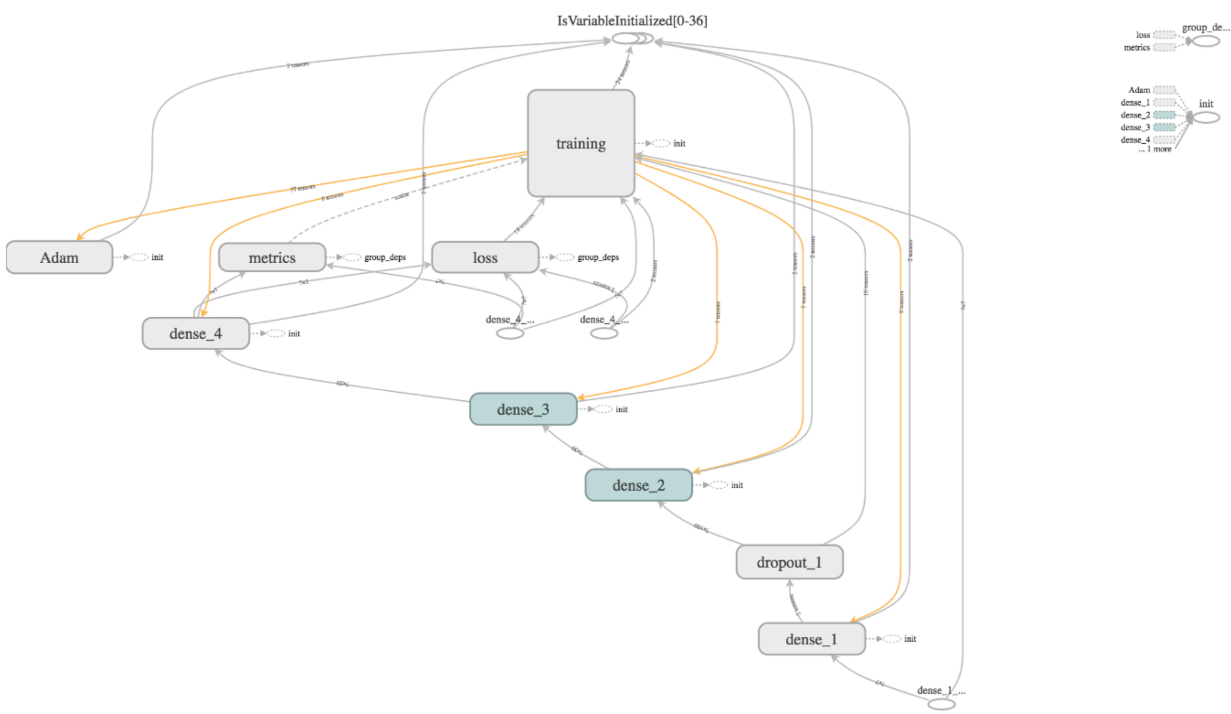
In order to evaluate the we run our programme ate get the train loss statistics. As we can see below for RMSprop optimizer, epoch=800 and batch size=128 and learning rate of 0.004 we got the minimum train
loss.

```
Epoch 800/800
 - 0s - loss: 2.5928 - mean_absolute_error: 1.1923 - val_loss: 5.6762 - val_mean_absolute_error: 1.6395
Train MAE:  1.1442 , Train Loss:  2.5337
Val MAE:  1.6395 , Val Loss:  5.6762
```

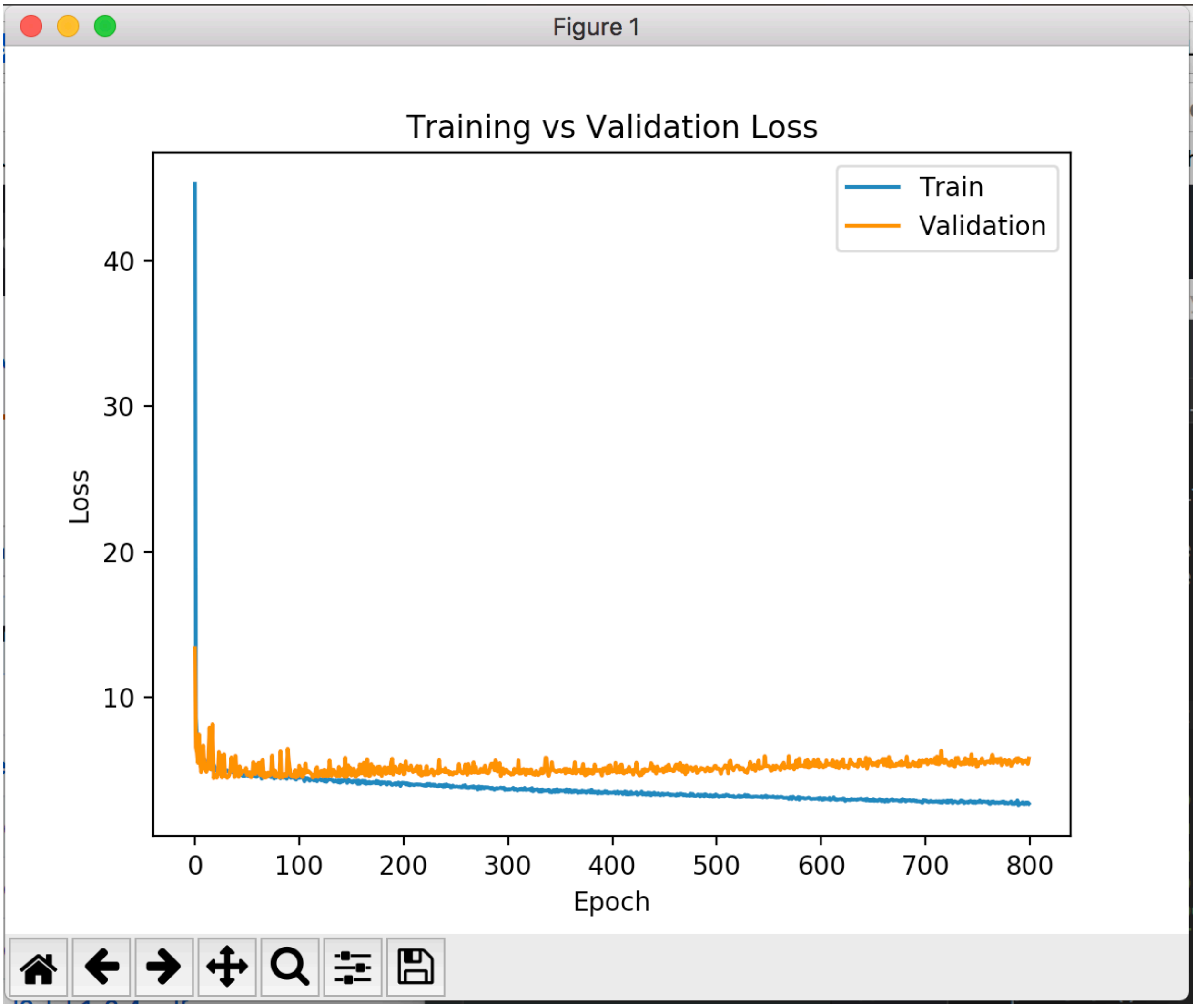
Similarly changed the optimizer ,batch size activation function and learning rate and evaluated the train loss as shown below.

Optimizer	Learning Rate	Batch Size	Activation Function	Epoch	Train Loss	Val Loss	Train Mean Absolute Error	Val Mean Absolute Error
Adam	0.002	130	sigmod,relu	200	3.9185	4.5463	1.4161	1.4906
Adam	0.001	140	tanh,sigmoid	400	3.8472	4.6826	1.3933	1.502
Adam	0.003	128	tanh,relu,	600	3.0858	5.1478	1.312	1.6404
Adam	0.004	120	tanh,relu,PReLU	800	2.5147	5.6156	1.157	1.6487
Optimizer	Learning Rate	Batch Size	Activation Function	Epoch	Train Loss	Val Loss	Train Mean Absolute Error	Val Mean Absolute Error
RMSprop	0.001	120	sigmod,relu	200	4.3324	4.5158	1.5116	1.5024
RMSprop	0.002	140	tanh,sigmoid	400	3.8254	5.2734	1.3809	1.5752
RMSprop	0.003	130	tanh,relu	600	3.2284	5.5304	1.3286	1.6822
RMSprop	0.004	128	tanh,relu,PReLU	800	2.7559	6.3351	1.2245	1.7231

Tensorboard Graph:



Loss Graph:



Evaluation Sheet Path:

https://docs.google.com/spreadsheets/d/1vfn5EH3dsXHxFmImskFtaC3yfTK6_Q_qc6txYm_RevE/edit#gid=0

Parameter:

Conclusion:

Using the evaluation above we can see that the RMSprop gives the better training loss as compared to Adam optimizer and also Epoch and learning rate playes very significant role in same.

Part 2:

Problem Statement:

Implement Logistic Regression with any data set of your choice.
a. Show the graph in TensorBoard
b. Show the Loss in TensorBoard
c. use score=model.evaluate(x_text, y_test)and then print('test accuracy', score[1])to print the accuracy
c. Change three hyper parameter and report how the accuracy changes

Objective:

Linear regression uses the general linear equationY=b0+b1*X where Y is a continuous dependent variable(for example predicting the prices of houses). Logistic regression is another generalized linear model procedure using the same basic formula, but instead of the continuousY, it is regressing for the probability of a categorical outcome (for example predicting Email as spam or NOT spam.

Dataset used

MNIST dataset is used for this problem, in which the labels of the dataset are 10 different categories.

Code Snippet:

The source code can be found here

mod2_Lab1_2.1.py | mod2_Lab1_2.2.py | mod2_Lab1_2.3.py | mod2_Lab1_2.4.py | mod2_Lab1_1.py

```python
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from keras.datasets import mnist
from keras.utils import np_utils
from keras.callbacks import TensorBoard
import matplotlib.pyplot as plt

batch_size = 128
nb_classes = 10
nb_epoch = 100

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
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
Y_Train = np_utils.to_categorical(y_train, nb_classes)
Y_Test = np_utils.to_categorical(y_test, nb_classes)

# Logistic regression model
model = Sequential()
model.add(Dense(output_dim=10, input_shape=(784,), init='normal', activation='softmax'))
model.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
tensorboard = TensorBoard(log_dir="logs/Output",histogram_freq=0, write_graph=True, write_images=True)
history = model.fit(X_train, Y_Train, nb_epoch=nb_epoch, batch_size=batch_size,callbacks=[tensorboard])
# Evaluate
evaluation = model.evaluate(X_test, Y_Test, verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (evaluation[0], evaluation[1]))
'''
```

mod2_Lab1_2.1.py | mod2_Lab1_2.2.py | mod2_Lab1_2.3.py | mod2_Lab1_2.4.py | mod2_Lab1_1.py

```python
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from keras.datasets import mnist
from keras.utils import np_utils
from keras.callbacks import TensorBoard
import tensorflow as tf

batch_size = 128
nb_classes = 10
nb_epoch = 100

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
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
Y_Train = np_utils.to_categorical(y_train, nb_classes)
Y_Test = np_utils.to_categorical(y_test, nb_classes)

# Logistic regression model
model = Sequential()
model.add(Dense(output_dim=10, input_shape=(784,), init='normal', activation='softmax'))
model.compile(optimizer='RMSprop', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
tensorboard = TensorBoard(log_dir="logs/Output",histogram_freq=0, write_graph=True, write_images=True)
model.fit(X_train, Y_Train, nb_epoch=nb_epoch, batch_size=batch_size,callbacks=[tensorboard])
# Evaluate
evaluation = model.evaluate(X_test, Y_Test, verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (evaluation[0], evaluation[1]))
```

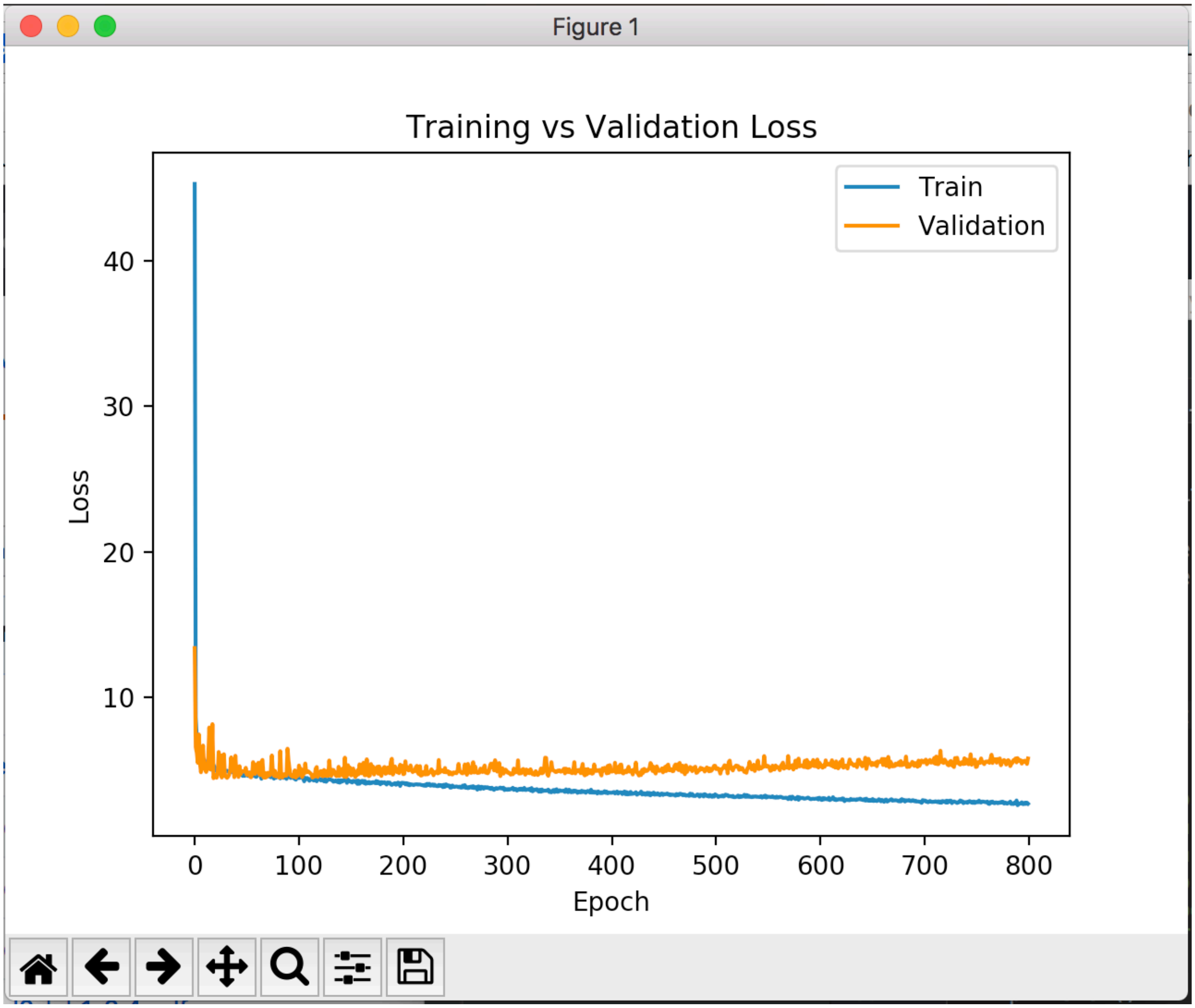
```python
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from keras.datasets import mnist
from keras.utils import np_utils
from keras.callbacks import TensorBoard
import tensorflow as tf

batch_size = 50
nb_classes = 10
nb_epoch = 20

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
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
Y_Train = np_utils.to_categorical(y_train, nb_classes)
Y_Test = np_utils.to_categorical(y_test, nb_classes)

# Logistic regression model
model = Sequential()
model.add(Dense(output_dim=10, input_shape=(784,), init='normal', activation='softmax'))
model.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
tensorboard = TensorBoard(log_dir="logs/Output",histogram_freq=0, write_graph=True, write_images=True)
model.fit(X_train, Y_Train, nb_epoch=nb_epoch, batch_size=batch_size,callbacks=[tensorboard])
# Evaluate
evaluation = model.evaluate(X_test, Y_Test, verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (evaluation[0], evaluation[1]))
```

```python
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers
from keras.datasets import mnist
from keras.utils import np_utils
from keras.callbacks import TensorBoard
import tensorflow as tf

batch_size = 50
nb_classes = 10
nb_epoch = 100

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
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
Y_Train = np_utils.to_categorical(y_train, nb_classes)
Y_Test = np_utils.to_categorical(y_test, nb_classes)

# Logistic regression model
model = Sequential()
model.add(Dense(output_dim=10, input_shape=(784,), init='normal', activation='softmax'))
model.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Train
tensorboard = TensorBoard(log_dir="logs/Output",histogram_freq=0, write_graph=True, write_images=True)
model.fit(X_train, Y_Train, nb_epoch=nb_epoch, batch_size=batch_size,callbacks=[tensorboard])
# Evaluate
evaluation = model.evaluate(X_test, Y_Test, verbose=1)
print('Summary: Loss over the test dataset: %.2f, Accuracy: %.2f' % (evaluation[0], evaluation[1]))
```

Output/ Workflow:

The output generated by the code can be found here part1 , part 2 , part 3 , part 4

The code snippet of output generated are as follows:



Parameter:

Part 1 of the problem:

batch_size = 128

nb_classes = 10

nb_epoch = 100
Optimizer='SGD'

Part 2 of the problem:

batch_size = 128
nb_classes = 10
nb_epoch = 100
Optimizer='RMSprop'

Part 3 of the problem:

batch_size = 50
nb_classes = 10
nb_epoch = 20
Optimizer='SGD'

Part 4 of the problem:

batch_size = 50
nb_classes = 10
nb_epoch = 100
Optimizer='SGD'

Conclusion:

part 1 - accuracy = 0.92 and loss = 0.28 (over test dataset)

part 2 - accuracy = 0.93 and loss = 0.3 (over test dataset)

part 3 - accuracy = 0.92 and loss = 0.3 (over test dataset)

part 4 - accuracy = 0.92 and loss = 0.27 (over test dataset)

Thus it is observed that we get slightly more accuracy when epoch is 100, optimizer is SGD, less batch size and nb class is 10 compared to others.

References:

1. https://stackoverflow.com/questions/12146914/what-is-the-difference-between-linear-regression-and-logistic-regression
2. https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8