**INDEX**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Practical no.** | **Practical Aim** | **Page no.** | **Date** | **Sign** |
| 1. | Performing matrix multiplication and finding eigenvectors and eigenvalues using TensorFlow | 4 |  |  |
| 2. | Solving XOR problem using deep feed forward network | 6 |  |  |
| 3. | Implementing deep neural network for performing binary classification task | 10 |  |  |
| 4. | Implementation of convolutional neural network to predict numbers from number images | 12 |  |  |
| 5a. | Using deep feed forward network with two hidden layers for performing multi-class classification and predicting the class | 15 |  |  |
| 5b. | Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class | 17 |  |  |
| 5c. | Using a deep feed forward network with two hidden layers for performing linear regression and predicting values | 19 |  |  |
| 6a. | Evaluating feed forward deep network for regression using KFold cross validation | 21 |  |  |
| 6b. | Evaluating feed forward deep network for multiclass Classification using KFold cross-validation | 23 |  |  |
| 7 | Demonstrate recurrent neural network that learns to perform sequence analysis for stock price | 25 |  |  |
| 8 | Performing encoding and decoding of images using deep autoencoder | 31 |  |  |
| 9 | Denoising of images using autoencoder | 35 |  |  |

**Practical no.1**

# Aim : Performing matrix multiplication and finding eigenvectors and eigenvalues using TensorFlow

**Input :**

import tensorflow as tf

print("Dilip Deepak Jaiswar, 05, M.Sc IT (Part II) Sem 4, Sathaye College")

print("Matrix Multiplication Demo")

x=tf.constant([1,2,3,4,5,6],shape=[2,3])

print(x)

y=tf.constant([7,8,9,10,11,12],shape=[3,2])

print(y)

z=tf.matmul(x,y)

print("Product:",z)

e\_matrix\_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

print("Matrix A:\n{}\n\n".format(e\_matrix\_A))

eigen\_values\_A,eigen\_vectors\_A=tf.linalg.eigh(e\_matrix\_A)

print("Eigen Vectors:\n{}\n\nEigen Values:\n{}\n".format(eigen\_vectors\_A,eigen\_values\_A))

**Practical no.2**

**Aim :  Solving XOR problem using deep feed forward network.**

**Input :**

import numpy as np

from keras.layers import Dense

from keras.models import Sequential

print("Dilip Deepak Jaiswar, 05, M.Sc IT (Part II) Sem 4, Sathaye College")

model=Sequential()

model.add(Dense(units=2,activation='relu',input\_dim=2))

model.add(Dense(units=1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy'])

print(model.summary())

print(model.get\_weights())

X=np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])

Y=np.array([0.,1.,1.,0.])

model.fit(X,Y,epochs=1000,batch\_size=4)

print(model.get\_weights())

print(model.predict(X,batch\_size=4))

**Practical no. 3**

**Aim :  Implementing deep neural network for performing binary classification task.**

**Input :**

# Load libraries

import numpy as np

from keras.datasets import imdb

from keras.preprocessing.text import Tokenizer

from keras import models

from keras import layers

print ("Dilip Deepak Jaiswar, Sathaye College, 05")

# Set random seed

np.random.seed(0)

# Set the number of features we want

number\_of\_features = 1000

# Load data and target vector from movie review data

(train\_data, train\_target), (test\_data, test\_target) = imdb.load\_data(num\_words=number\_of\_features)

# Convert movie review data to one-hot encoded feature matrix

tokenizer = Tokenizer(num\_words=number\_of\_features)

train\_features = tokenizer.sequences\_to\_matrix(train\_data, mode='binary')

test\_features = tokenizer.sequences\_to\_matrix(test\_data, mode='binary')# Start neural network

network = models.Sequential()

# Add fully connected layer with a ReLU activation function

network.add(layers.Dense(units=16, activation='relu', input\_shape=(number\_of\_features,)))

# Add fully connected layer with a ReLU activation function

network.add(layers.Dense(units=16, activation='relu'))

# Add fully connected layer with a sigmoid activation function

network.add(layers.Dense(units=1, activation='sigmoid'))

# Compile neural network

network.compile(loss='binary\_crossentropy', # Cross-entropy

                optimizer='rmsprop', # Root Mean Square Propagation

                metrics=['accuracy']) # Accuracy performance metric

# Train neural network

history = network.fit(train\_features, # Features

                      train\_target, # Target vector

                      epochs=3, # Number of epochs

                      verbose=1, # Print description after each epoch

                      batch\_size=100, # Number of observations per batch

                      validation\_data=(test\_features, test\_target)) # Data for

evaluation

**Practical no. 4**

**Aim :  Implementation of convolutional neural network to predict numbers from number images.**

**Input :**

from keras.datasets import mnist

from keras.models import Sequential

from keras.layers import Dense,Conv2D,Flatten

import matplotlib.pyplot as plt

print ("Dilip Deepak Jaiswar, Sathaye College, 05")

#download mnist data and split into train and test sets

(X\_train,Y\_train),(X\_test,Y\_test)=mnist.load\_data()

#plot the first image in the dataset

plt.imshow(X\_train[0])

plt.show()

print(X\_train[0].shape)

X\_train=X\_train.reshape(60000,28,28,1)

X\_test=X\_test.reshape(10000,28,28,1)

Y\_train[0]

print(Y\_train[0])

model=Sequential()

#add model layers

#learn image features

model.add(Conv2D(64,kernel\_size=3,activation='relu',input\_shape=(28,28,1)))

model.add(Conv2D(32,kernel\_size=3,activation='relu'))

model.add(Flatten())

model.add(Dense(10,activation='softmax'))

model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics=['accuracy'])

#train

print(model.predict(X\_test[:4]))

#actual results for 1st 4 images in the test set

print(Y\_test[:4])

**Practical no. 5**

**Aim : (5a) Using deep feed forward network with two hidden layers for performing multi-class classification and predicting the class.**

**Input :**

from keras.models import Sequential

from keras.layers import Dense

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import MinMaxScaler

print ("Dilip Deepak Jaiswar, Sathaye College, 05")

X,Y=make\_blobs(n\_samples=100,centers=2,n\_features=2,random\_state=1)

scalar=MinMaxScaler()

scalar.fit(X)

X=scalar.transform(X)

model=Sequential()

model.add(Dense(4,input\_dim=2,activation='relu'))

model.add(Dense(4,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy',optimizer='adam')

model.fit(X,Y,epochs=500)

Xnew,Yreal=make\_blobs(n\_samples=3,centers=2,n\_features=2,random\_state=1)

Xnew=scalar.transform(Xnew)

Ynew=model.predict(Xnew)

for i in range(len(Xnew)):

  print("X=%s,Predicted=%s,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))

**Aim : (5b) Using a deep feed forward network with two hidden layers for performing classification and predicting the probability of class.**

**Input :**

from keras.models import Sequential

from keras.layers import Dense

from sklearn.datasets import make\_blobs

from sklearn.preprocessing import MinMaxScaler

print ("Dilip Deepak Jaiswar, Sathaye College, 05")

X,Y=make\_blobs(n\_samples=100,centers=2,n\_features=2,random\_state=1)

scalar=MinMaxScaler()

scalar.fit(X)

X=scalar.transform(X)

model=Sequential()

model.add(Dense(4,input\_dim=2,activation='relu'))

model.add(Dense(4,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary\_crossentropy',optimizer='adam')

model.fit(X,Y,epochs=500)

Xnew,Yreal=make\_blobs(n\_samples=3,centers=2,n\_features=2,random\_state=1)

Xnew=scalar.transform(Xnew)

Yclass=model(Xnew)

Ynew=model(Xnew)

for i in range(len(Xnew)):

 print("X=%s,Predicted\_probability=%s,Predicted\_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))

**5c) : Using a deep feed forward network with two hidden layers for performing linear regression and predicting values.**

**Input :**

from keras.models import Sequential

from keras.layers import Dense

from sklearn.datasets import make\_regression

from sklearn.preprocessing import MinMaxScaler

print("Dilip Deepak Jaiswar, Satahye College, 05")

X,Y=make\_regression(n\_samples=100,n\_features=2,noise=0.1,random\_state=1)

scalarX,scalarY=MinMaxScaler(),MinMaxScaler()

scalarX.fit(X)

scalarY.fit(Y.reshape(100,1))

X=scalarX.transform(X)

Y=scalarY.transform(Y.reshape(100,1))

model=Sequential()

model.add(Dense(4,input\_dim=2,activation='relu'))

model.add(Dense(4,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.compile(loss='mse',optimizer='adam')

model.fit(X,Y,epochs=1000,verbose=0)

Xnew,a=make\_regression(n\_samples=3,n\_features=2,noise=0.1,random\_state=1)

Xnew=scalarX.transform(Xnew)

Ynew=model.predict(Xnew)

for i in range(len(Xnew)):

 print("X=%s,Predicted=%s"%(Xnew[i],Ynew[i]))

**Practical no. 6**

**Aim : (6a) Evaluating feed forward deep network for regression using KFold cross validation**

**Input :**

# MLP for Pima Indians Dataset with 10-fold cross validation

from keras.models import Sequential

from keras.layers import Dense

from sklearn.model\_selection import StratifiedKFold

import numpy

print ("Dilip Deepak Jaiswar, Sathaye college, 05")

# fix random seed for reproducibility

seed = 7

numpy.random.seed(seed)

# load pima indians dataset

dataset = numpy.loadtxt("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv", delimiter=",")

# split into input (X) and output (Y) variables

X = dataset[:,0:8]

Y = dataset[:,8]

# define 10-fold cross validation test harness

kfold = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=seed)

cvscores = []

for train, test in kfold.split(X, Y):

 model = Sequential()

 model.add(Dense(12, input\_dim=8, activation='relu'))

 model.add(Dense(8, activation='relu'))

 model.add(Dense(1, activation='sigmoid'))

 # Compile model

 model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

 # Fit the model

 model.fit(X[train], Y[train], epochs=150, batch\_size=10, verbose=0)

 # evaluate the model

 scores = model.evaluate(X[test], Y[test], verbose=0)

 print("%s: %.2f%%" % (model.metrics\_names[1], scores[1]\*100))

 cvscores.append(scores[1] \* 100)

print("%.2f%% (+/- %.2f%%)" % (numpy.mean(cvscores), numpy.std(cvscores)))

**Aim : (6b) Evaluating feed forward deep network for multiclass Classification using KFold cross-validation**

**Input :**

# multi-class classification with Keras

import pandas

from keras.models import Sequential

from keras.layers import Dense

from keras.wrappers.scikit\_learn import KerasClassifier

from keras.utils import np\_utils

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import KFold

from sklearn.preprocessing import LabelEncoder

from sklearn.pipeline import Pipeline

print ("Dilip Deepak Jaiswar, Sathaye College, 05")

# load dataset

dataframe = pandas.read\_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv",

header=None)

dataset = dataframe.values

X = dataset[:,0:4].astype(float)

Y = dataset[:,4]

# encode class values as integers

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

# convert integers to dummy variables (i.e. one hot encoded)

dummy\_y = np\_utils.to\_categorical(encoded\_Y)

# define baseline model

def baseline\_model():

  # create model

 model = Sequential()

 model.add(Dense(8, input\_dim=4, activation='relu'))

 model.add(Dense(3, activation='softmax'))

 # Compile model

 model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

 return model

estimator = KerasClassifier(build\_fn=baseline\_model, epochs=20, batch\_size=5, verbose=0)

kfold = KFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(estimator, X, dummy\_y, cv=kfold)

print("Baseline: %.2f%% (%.2f%%)" % (results.mean()\*100, results.std()\*100))

**Practical no. 7**

**Aim : Demonstrate recurrent neural network that learns to perform sequence analysis for stock price.**

**Input :**

#Importing the libraries

from nsepy import get\_history as gh

import datetime as dt

from matplotlib import pyplot as plt

from sklearn import model\_selection

from sklearn.metrics import confusion\_matrix

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

print ("Dilip Deepak Jaiswar, Sathaye COllege, 05")

#Setting start and end dates and fetching the historical data

start = dt.datetime(2016,1,1)

end = dt.datetime(2021,12,31)

stk\_data = gh(symbol='SBIN',start=start,end=end)

#Visualizing the fetched data

plt.figure(figsize=(14,14))

plt.plot(stk\_data['Close'])

plt.title('Historical Stock Value')

plt.xlabel('Date')

plt.ylabel('Stock Price')

plt.show()

#Data Preprocessing

stk\_data['Date'] = stk\_data.index

data2 = pd.DataFrame(columns = ['Date', 'Open', 'High', 'Low', 'Close'])

data2['Date'] = stk\_data['Date']

data2['Open'] = stk\_data['Open']

data2['High'] = stk\_data['High']

data2['Low'] = stk\_data['Low']

data2['Close'] = stk\_data['Close']

train\_set = data2.iloc[:, 1:2].values

sc = MinMaxScaler(feature\_range = (0, 1))

training\_set\_scaled = sc.fit\_transform(train\_set)

X\_train = []

y\_train = []

for i in range(60, 1482):

 X\_train.append(training\_set\_scaled[i-60:i, 0])

 y\_train.append(training\_set\_scaled[i, 0])

X\_train, y\_train = np.array(X\_train), np.array(y\_train)

X\_train = np.reshape(X\_train, (X\_train.shape[0], X\_train.shape[1], 1))

#Defining the LSTM Recurrent Model

regressor = Sequential()

regressor.add(LSTM(units = 50, return\_sequences = True, input\_shape = (X\_train.shape[1], 1)))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50, return\_sequences = True))

regressor.add(Dropout(0.2))

regressor.add(LSTM(units = 50))

regressor.add(Dropout(0.2))

regressor.add(Dense(units = 1))

#Compiling and fitting the model

regressor.compile(optimizer = 'adam', loss = 'mean\_squared\_error')

regressor.fit(X\_train, y\_train, epochs = 5, batch\_size = 32)

#Fetching the test data and preprocessing

testdataframe = gh(symbol='SBIN',start=dt.datetime(2019,1,1),end=dt.datetime(2019,9,18))

testdataframe['Date'] = testdataframe.index

testdata = pd.DataFrame(columns = ['Date', 'Open', 'High', 'Low', 'Close'])

testdata['Date'] = testdataframe['Date']

testdata['Open'] = testdataframe['Open']

testdata['High'] = testdataframe['High']

testdata['Low'] = testdataframe['Low']

testdata['Close'] = testdataframe['Close']

real\_stock\_price = testdata.iloc[:, 1:2].values

dataset\_total = pd.concat((data2['Open'], testdata['Open']), axis = 0)

inputs = dataset\_total[len(dataset\_total) - len(testdata) - 60:].values

inputs = inputs.reshape(-1,1)

inputs = sc.transform(inputs)

X\_test = []

for i in range(60, 235):

 X\_test.append(inputs[i-60:i, 0])

X\_test = np.array(X\_test)

X\_test = np.reshape(X\_test, (X\_test.shape[0], X\_test.shape[1], 1))

#Making predictions on the test data

predicted\_stock\_price = regressor.predict(X\_test)

predicted\_stock\_price = sc.inverse\_transform(predicted\_stock\_price)

#Visualizing the prediction

plt.figure(figsize=(20,10))

plt.plot(real\_stock\_price, color = 'green', label = 'SBI Stock Price')

plt.plot(predicted\_stock\_price, color = 'red', label = 'Predicted SBI Stock Price')

plt.title('SBI Stock Price Prediction')

plt.xlabel('Trading Day')

plt.ylabel('SBI Stock Price')

plt.legend()

plt.show()

**Practical no. 8**

**Aim : Performing encoding and decoding of images using deep autoencoder.**

**Input :**

%matplotlib inline

%config InlineBackend.figure\_format = 'retina'

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

from \_\_future\_\_ import print\_function

from keras.models import Model

from keras.layers import Dense, Input

from keras.datasets import mnist

from keras.regularizers import l1

print ("Dilip Deepak Jaiswar, Sathaye college, 05")

def plot\_autoencoder\_outputs(autoencoder, n, dims):

  decoded\_imgs = autoencoder.predict(x\_test)

  # number of example digits to show

  n = 5

  plt.figure(figsize=(10, 4.5))

  for i in range(n):

     # plot original image

     ax = plt.subplot(2, n, i + 1)

     plt.imshow(x\_test[i].reshape(\*dims))

     plt.gray()

     ax.get\_xaxis().set\_visible(False)

     ax.get\_yaxis().set\_visible(False)

     if i == n/2:

        ax.set\_title('Original Images')

        # plot reconstruction

        ax = plt.subplot(2, n, i + 1 + n)

        plt.imshow(decoded\_imgs[i].reshape(\*dims))

        plt.gray()

        ax.get\_xaxis().set\_visible(False)

        ax.get\_yaxis().set\_visible(False)

        if i == n/2:

          ax.set\_title('Reconstructed Images')

        plt.show()

(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

print(x\_test.shape)

input\_size = 784

hidden\_size = 128

code\_size = 32

input\_img = Input(shape=(input\_size,))

hidden\_1 = Dense(hidden\_size, activation='relu')(input\_img)

code = Dense(code\_size, activation='relu')(hidden\_1)

hidden\_2 = Dense(hidden\_size, activation='relu')(code)

output\_img = Dense(input\_size, activation='sigmoid')(hidden\_2)

autoencoder = Model(input\_img, output\_img)

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

autoencoder.fit(x\_train, x\_train, epochs=3)

plot\_autoencoder\_outputs(autoencoder, 5, (28, 28))

**Practical no. 9**

**Aim : Denoising of images using autoencoder.**

**Input :**

import keras

from keras.datasets import mnist

from keras import layers

import numpy as np

from keras.callbacks import TensorBoard

import matplotlib.pyplot as plt

(X\_train,\_),(X\_test,\_)=mnist.load\_data()

X\_train=X\_train.astype('float32')/255.

X\_test=X\_test.astype('float32')/255.

X\_train=np.reshape(X\_train,(len(X\_train),28,28,1))

X\_test=np.reshape(X\_test,(len(X\_test),28,28,1))

noise\_factor=0.5

X\_train\_noisy=X\_train+noise\_factor\*np.random.normal(loc=0.0,scale=1.0,size=X\_train.shape)

X\_test\_noisy=X\_test+noise\_factor\*np.random.normal(loc=0.0,scale=1.0,size=X\_test.shape)

X\_train\_noisy=np.clip(X\_train\_noisy,0.,1.)

X\_test\_noisy=np.clip(X\_test\_noisy,0.,1.)

n=10

plt.figure(figsize=(20,2))

for i in range(1,n+1):

ax=plt.subplot(1,n,i)

plt.imshow(X\_test\_noisy[i].reshape(28,28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

input\_img=keras.Input(shape=(28,28,1))

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(input\_img)

x=layers.MaxPooling2D((2,2),padding='same')(x)

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)

encoded=layers.MaxPooling2D((2,2),padding='same')(x)

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(encoded)

x=layers.UpSampling2D((2,2))(x)

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)

x=layers.UpSampling2D((2,2))(x)

decoded=layers.Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)

autoencoder=keras.Model(input\_img,decoded)

autoencoder.compile(optimizer='adam',loss='binary\_crossentropy')

autoencoder.fit(X\_train\_noisy,X\_train,

epochs=3,

batch\_size=128,

shuffle=True,

validation\_data=(X\_test\_noisy,X\_test),

callbacks=[TensorBoard(log\_dir='/tmo/tb',histogram\_freq=0,write\_graph=False)])

predictions=autoencoder.predict(X\_test\_noisy)

m=10

plt.figure(figsize=(20,2))

for i in range(1,m+1):

ax=plt.subplot(1,m,i)

plt.imshow(predictions[i].reshape(28,28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()