**Practical 5**

**a)** **A. Using deep feed forward network with two hidden**

**layers for performing multiclass classification and**

**predicting the class.**

**Code:**

# 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

# load dataset

dataframe = pandas.read\_csv("iris.data", 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=200, batch\_size=5, verbose=0)

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

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

print("\n")

print("7\_Govind Saini \n")

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

**. B. Using a deep feed forward network with two hidden**

**layers for performing classification and predicting the**

**probability of class.**

**Code:**

import numpy as np

from sklearn.datasets import load\_wine

from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from keras.layers import Dense, Input, concatenate, Dropout

from keras.models import Model

from tensorflow.keras import optimizers

dataset = load\_wine()

# Number of sub-networks

ensemble\_num = 10

# 80% size of original (training) dataset

bootstrap\_size = 0.8

# 80% for training, 20% for test

training\_size = 0.8

# Number of neurons in hidden layer

num\_hidden\_neurons = 10

# Percentage of weights dropped out before softmax output (this prevents overfitting)

dropout = 0.25

# Number of epochs (complete training episodes over the training set) to run

epochs = 100

# Mini batch size for better convergence

batch = 10

# Get the holdout training and test set

temp = []

scaler = MinMaxScaler()

# One hot encode the target classes

one\_hot = OneHotEncoder()

dataset['data'] = scaler.fit\_transform(dataset['data'])

dataset['target'] = one\_hot.fit\_transform(np.reshape(dataset['target'], (-1, 1))).toarray()

for i in range(len(dataset.data)):

    temp.append([dataset['data'][i], np.array(dataset['target'][i])])

# Shuffle the row of data and targets

temp = np.array(temp)

np.random.shuffle(temp)

# Holdout training and test stop index

stop = int(training\_size \* len(dataset.data))

train\_X = np.array([x for x in temp[:stop, 0]])

train\_Y = np.array([x for x in temp[:stop, 1]])

test\_X = np.array([x for x in temp[stop:, 0]])

test\_Y = np.array([x for x in temp[stop:, 1]])

# Now build the ensemble neural network

# First, let's build the individual sub-networks, each as a Keras functional model.

sub\_net\_outputs = []

sub\_net\_inputs = []

for i in range(ensemble\_num):

    # Two hidden layers to keep it simple

    # Specify input shape to the shape of the training set

    net\_input = Input(shape=(train\_X.shape[1],))

    sub\_net\_inputs.append(net\_input)

    y = Dense(num\_hidden\_neurons)(net\_input)

    y = Dense(num\_hidden\_neurons)(y)

    y = Dropout(dropout)(y)

    # sub\_nets contains the output tensors

    sub\_net\_outputs.append(y)

# Now concatenate the output tensors

y = concatenate(sub\_net\_outputs)

# Final softmax output layer

y = Dense(train\_Y[0].shape[0], activation='softmax')(y)

# Now build the whole functional model

model = Model(inputs=sub\_net\_inputs, outputs=y)

model.compile(optimizer='rmsprop', loss='categorical\_crossentropy')

print("\n7, Govind Saini \n")

print("Begin training...")

# Train the model

model.fit([train\_X] \* ensemble\_num, train\_Y, validation\_data=([test\_X] \* ensemble\_num, test\_Y),

          epochs=epochs, batch\_size=batch)

print("Training complete...")

np.set\_printoptions(precision=2, suppress=True)

for i in range(len(test\_X)):

    print("Prediction: " + str(model.predict(test\_X[i].reshape(1, test\_X[i].shape[0]) \* ensemble\_num)) +

          " | True: " + str(test\_Y[i]))

**C. Using a deep feed forward network with two hidden**

**layers for performing linear regression and predicting**

**values.**

**Code:**

from pandas import read\_csv

from keras.models import Sequential

from keras.layers import Dense

from keras.wrappers.scikit learn import KerasRegressor

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

from sklearn.model\_selection import KFold

# load dataset

dataframe = read\_csv("housing.csv", delim\_whitespace=True, header=None) dataset dataframe.values

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

X = dataset[:,0:13]

Y = dataset[:,13]

# define base model def baseline\_model():

# create model

model = Sequential()

model.add(Dense (13, input\_dim=13, kernel\_initializer='normal', activation='relu')) model.add(Dense (1, kernel\_initializer='normal'))

# Compile model

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

return model

# evaluate model

estimator = Keras Regressor (build\_fn-baseline\_model, epochs-100, batch\_size=5, verbose=0) kfold KFold(n\_splits=10)

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

print('\n')

print("7\_Govind Saini \n")

print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))

**practical 6**

**A. Evaluating feed forward deep network for regression**

**using K-Fold cross validation**

**. B. Evaluating feed forward deep network for multiclass**

**Classification using K-Fold cross-validation**.

A code:

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense

from keras.wrappers.scikit learn import KerasRegressor from sklearn.model\_selection import cross\_val\_score

from sklearn.model selection import KFold

from sklearn.preprocessing import StandardScaler from sklearn.pipeline import Pipeline

# load dataset

dataframe = pd.read\_csv("ames\_iowa\_housing.csv",

delim\_whitespace=True, header=None)

dataset = dataframe.values

X = dataset [:, 0:13]

Y = dataset [:, 1:13]

# define Accuracy model def accuracy\_model():

# create model

model = Sequential()

model.add(Dense (15, input\_dim=13,

kernel\_initializer='normal', activation='relu'))

model.add(Dense (13, kernel\_initializer='normal', activation='relu')) model.add(Dense (1, kernel\_initializer='normal'))

# Compile model

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

return model

estimators = []

estimators.append(('standardize', StandardScaler())) estimators.append(('mlp', Keras Regressor(

build\_fn=accuracy\_model, epochs=10, batch\_size=5)))

pipeline = Pipeline (estimators)

kfold KFold (n\_splits=10)

results = cross\_val\_score (pipeline, X, Y, cv=kfold)

print('\n')

print("7\_Govind Saini \n")

print("Accuracy: %.2f (%.2f) MSE" % (results.mean(), results.std()))

**b code:**

import pandas

from keras.models import Sequential

ww www

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

ww

from sklearn.model\_selection import KFold

from sklearn.preprocessing import LabelEncoder from sklearn.pipeline import Pipeline

# load dataset

dataframe

=

pandas.read\_csv("iris.data", 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=200, batch\_size=5, verbose=0) <fold = KFold (n\_splits=10, shuffle=True)

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

print('\n')

print("7\_Govind Saini \n")

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

**practical 7**

**Implementation of convolutional neural network to**

**predict numbers from number images**

**code:**

from keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten import matplotlib.pyplot as plt

# 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("7\_Govind Saini")

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 = to\_categorical (Y\_train)

Y\_test = to\_categorical (Y\_test)

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",

# train

metrics=['accuracy'])

model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=1)

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

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

print(Y\_test[:4])

P**practical 8**

**Demonstrate recurrent neural network that learns to**

**perform sequence analysis for stock price.**

#Importing the libraries

from nsepy import get\_history as gh import datetime as dt

from matplotlib import pyplot as plt

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

#Setting start and end dates and fetching the historical data

start = dt.datetime(2013,1,1)

end = dt.datetime (2018, 12,31)

stk\_data

gh(symbol='SBIN', start-start, end-end)

print("7\_Govind Saini")

print("\n")

#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-60:1, 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 = 15, 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

N

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

N

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-60:1, 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 9**

**Performing encoding and decoding of images using**

**deep Autoencoder.**

import keras

from keras import layers

from keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

# This is the size of our encoded representations

# 32 floats -> compression of factor 24.5, assuming the input is 784 floats encoding\_dim = 32

# This is our input image

input\_img = keras. Input (shape=(784,))

# "encoded" is the encoded representation of the input

encoded = layers. Dense (encoding\_dim, activation='relu') (input\_img)

# "decoded" is the lossy reconstruction of the input

decoded = layers. Dense (784, activation='sigmoid') (encoded)

# This model maps an input to its reconstruction autoencoder = keras.Model(input\_img, decoded)

# This model maps an input to its encoded representation encoder = keras.Model (input\_img, encoded)

# This is our encoded (32-dimensional) input encoded\_input = keras. Input (shape=(encoding\_dim,)) # Retrieve the last layer of the autoencoder model decoder\_layer = autoencoder.layers[-1]

# Create the decoder model

decoder = keras.Model (encoded\_input, decoder\_layer (encoded\_input))

decoder = keras. Model (encoded\_input, decoder\_layer (encoded\_input))

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

(x\_train, \_), (x\_test, ) = mnist.load\_data() x\_train = x\_train.astype('float32') / 255. x\_test = x\_test.astype('float32') / 255.

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("7\_GovindSaini \n")

print(x\_train.shape)

print(x\_test.shape)

autoencoder.fit(x\_train, x\_train,

epochs=13,

batch\_size=256,

shuffle=True, validation\_data=(x\_test, x\_test))

# Encode and decode some digits Note that we take them from the \*test\* set encoded\_imgs = encoder.predict(x\_test)

decoded\_imgs= decoder.predict(encoded\_imgs)

# Use Matplotlib (don't ask)

n = # How many digits we will display plt.figure(figsize=(20, 4)) for i in range(n):

# Display original

ax = plt.subplot(2, n, i + 1) plt.imshow(x\_test[i].reshape(28, 28))

plt.gray()

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

# Display reconstruction

ax plt.subplot(2, n, i + 1 + n) plt.imshow(decoded\_imgs[i].reshape(28, 28))

plt.gray()

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

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

plt.show()

|  |  |
| --- | --- |
|  |  |

**Practical 10**

**Denoising of images using Autoencoder.**

**Code:**

mport keras

from keras.datasets import mnist from keras import layers

import numpy as np

from keras.callbacks import TensorBoard

import matplotlib.pyplot as plt

print("7\_GovindSaini")

(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.)

# Use Matplotlib (don't ask) How many digits we will display

n = 10

plt.figure(figsize=(20, 2)) for i in range(1, n+1):

# Display reconstruction

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

=

X =

keras.Input (shape=(28, 28, 1))

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)

# Create the decoder model

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

autoencoder.compile(optimizer='adam', loss='binary\_crossentropy') autoencoder.fit(x\_train\_noisy, X\_train,

epochs=3,

batch\_size=328,

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))

# Display original

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()

|  |  |
| --- | --- |
|  |  |