|  |  |
| --- | --- |
| **Ex. No: 1** | **Implementing Logic Gates using Perceptron Algorithm** |
|  |

**Aim:** To write a program using python to implement logic gates using perceptron algorithm.

**Algorithm:**

1. Import necessary packages.
2. Assign different weights and bias for each logic gate.
3. Perform the Required Operations.
4. Display the Output

**Program :**

import numpy as np

def step(x):

return 1 if x >= 0.5 else 0

### Not Gate

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

weights = np.array([-1])

bias = np.array([1])

out = X@weights + bias

print(out)

### Tautology

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

weights = np.array([[0],[0]])

bias = np.array([1])

out = X@weights + bias

print(out)

### AND Gate

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

weights = np.array([[1],[1]])

bias = np.array([-1])

out = X@weights + bias

y = np.array(list(map(step, out))).reshape(4,1)

print(y)

### OR Gate

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

weights = np.array([[1],[1]])

bias = np.array([0])

out = X@weights + bias

y = np.array(list(map(step, out))).reshape(4,1)

print(y)

### NOR Gate

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

weights = np.array([[-1],[-1]])

bias = np.array([1])

out = X@weights + bias

y = np.array(list(map(step, out))).reshape(4,1)

print(y)

### NAND Gate

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

weights = np.array([[-1],[-1]])

bias = np.array([2])

out = X@weights + bias

y = np.array(list(map(step, out))).reshape(4,1)

print(y)

### XOR

def step(x):

return 1 if x>0 else 0

step = np.vectorize(step)

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

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

w1 = np.array([[1,-1],[-1,1]])

w2 = np.array([[1],[1]])

y\_cap = step(step(x@w1)@w2)

y\_cap

### XNOR

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

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

w1 = np.array([[1,-1],[1,-1]])

w2 = np.array([[1],[1]])

bias = np.array([[-1,1]])

y\_cap = step(step(x@w1 + bias)@w2)

y\_cap

**Result:** Thus, using Python, the logic gates have been implemented.

|  |  |
| --- | --- |
| **Ex. No: 2** | **MLP and Linear Regression** |
|  |

**Aim:** To write a program in TensorFlow for MLP and Linear Regression.

**Algorithm:**

1. Import the necessary packages.
2. Load the Datasets.
3. Compile the model.
4. Evaluate the model.
5. Print the metrices.

.

**Program :**

### MNIST

import warnings

warnings.filterwarnings("ignore")

import tensorflow as tf

import tensorflow.keras.datasets.mnist as mnist

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

model = tf.keras.models.Sequential([

tf.keras.layers.Normalization(),

tf.keras.layers.Flatten(input\_shape=(28,28)),

tf.keras.layers.Dense(128, activation="relu"),

tf.keras.layers.Dense(10)

])

model.compile(

optimizer = tf.keras.optimizers.Adam(0.001),

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=[tf.keras.metrics.SparseCategoricalAccuracy()]

)

model.fit(x=x\_train,y=y\_train,batch\_size=64,epochs=10)

model.evaluate(x=x\_test,y=y\_test)

### Linear Regression Using Tensorflow

learning\_rate = 0.01

training\_epochs = 1000

np.random.seed(69)

x = np.random.random\_sample(500).reshape(-1,1)

y = x\*2 + 0.5\*np.random.random(500).reshape(-1,1)

#### Training Loop

model = tf.keras.models.Sequential([tf.keras.layers.Dense(1)])

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss='mean\_absolute\_error')

model.fit(x,y,epochs=training\_epochs,batch\_size=128,verbose=False)

y\_pred = model.predict(x, verbose=False)

plt.scatter(x,y)

plt.plot(x, y\_pred, c="red")

print("Training Cost :", model.evaluate(x,y,verbose=False))

print("Weight :", model.get\_weights()[0])

print("Bias :", model.get\_weights()[1])

**Result:** Thus, using TensorFlow library, MLP and Linear Regression have been implemented.

|  |  |
| --- | --- |
| **Ex. No: 3** | **Ablation Studies** |
|  |

**Aim:** To Perform Ablation Studies on a Neural Network Architecture.

**Algorithm:**

1. Import the necessary packages.
2. Find a Vector dataset from Open Source.
3. Train the designed architecture with initial set of parameters
4. Change the training methods with various hyperparameters and record the observations.
5. Make comments about the inferences made from the table.

**Program :**

### Wine Dataset

import pandas as pd

import warnings

warnings.filterwarnings("ignore")

df = pd.read\_csv("Wine.csv")

print("Shape :", df.shape)

df.head()

df.info()

### Pre Processing

# One-Hot Quality column

df = pd.concat([df, pd.get\_dummies(df["quality"], drop\_first=True)], axis=1)

df = df.drop(["quality"], axis=1)

y = df["color"]

x = df.drop(["color"], axis=1)

# Label Encode color column

from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()

y = label.fit\_transform(df["color"])

# Train-Test Split

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=69)

### Model Building

import keras

from keras.models import Sequential

from keras.layers import Normalization

from keras.layers import Dense

architecture = [

Sequential([Normalization(),

Dense(32, activation="relu"),

Dense(16, activation="relu"),

Dense(1, activation="sigmoid")]),

Sequential([Normalization(),

Dense(64, activation="relu"),

Dense(32, activation="relu"),

Dense(16, activation="relu"),

Dense(1, activation="sigmoid")]),

Sequential([Normalization(),

Dense(128, activation="relu"),

Dense(32, activation="relu"),

Dense(1, activation="sigmoid")])

]

optimizer = [keras.optimizers.Adam(learning\_rate=0.01), keras.optimizers.Adam(learning\_rate=0.001),

keras.optimizers.SGD(learning\_rate=0.01)]

epochs = [5,10,20]

output = pd.DataFrame(columns=["No of Layers","No of Params","Optimizer","Learning Rate","Epochs","Accuracy"])

for arch in architecture:

for opti in optimizer:

for epoch in epochs:

model = arch

model.compile(optimizer=opti, loss="BinaryCrossentropy", metrics="Accuracy")

model.fit(x\_train, y\_train, epochs=epoch, verbose=False)

y\_pred = model.evaluate(x\_test, y\_test, verbose=False)

output = output.append(pd.DataFrame(

{"No of Layers" : [len(model.layers)],

"No of Params" : [model.count\_params()],

"Optimizer" : [opti.get\_config()["name"]],

"Learning Rate" : [opti.learning\_rate.numpy()],

"Epochs" : [epoch],

"Accuracy" : [y\_pred[1]]}))

**Result:** Thus, using TensorFlow library, Ablation Studies on neural network have been performed.

|  |  |
| --- | --- |
| **Ex. No: 4** | **Regularization Techniques and Linear Regression** |
|  |

**Aim:** To perform Regularization Techniques and implement Linear Regression using PyTorch.

**Algorithm:**

1. Import the necessary packages.
2. Load the Dataset.
3. Perform Pre-Processing steps on the dataset before fitting to the model.
4. Compare the Base Model with various Regularization methods and record the observations.
5. Compare the observations
6. Plot the Inference for better Understanding

.

**Program :**

### Wine Dataset

import pandas as pd

df = pd.read\_csv("Wine.csv")

print("Shape :", df.shape)

df.head()

df.info()

### Pre Processing

# One-Hot Quality column

df = pd.concat([df, pd.get\_dummies(df["quality"], drop\_first=True)], axis=1)

df = df.drop(["quality"], axis=1)

y = df["color"]

x = df.drop(["color"], axis=1)

# Label Encode color column

from sklearn.preprocessing import LabelEncoder

label = LabelEncoder()

y = label.fit\_transform(df["color"])

# Train-Test Split

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

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=69)

### Model Building

import keras

from keras.models import Sequential

from keras.layers import Normalization

from keras.layers import Dense, Dropout

# Plot

import matplotlib.pyplot as plt

def plot\_history(hist):

plt.plot(hist.history['loss'], label = 'loss')

plt.plot(hist.history['val\_loss'], label='val loss')

plt.title("Loss vs Val\_Loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

# Logging

output = pd.DataFrame(columns=["Model", "Accuracy", "Loss"])

### Base Model

model = Sequential([Normalization(),

Dense(64, activation="relu"),

Dense(16, activation="relu"),

Dense(1, activation="sigmoid")])

optimizer = keras.optimizers.Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss="BinaryCrossentropy", metrics="Accuracy")

hist = model.fit(x\_train, y\_train, epochs=10, verbose=False, validation\_data=(x\_test,y\_test))

Loss, Accuracy = model.evaluate(x\_test, y\_test, verbose=False)

print("Validation Result")

print("Loss :", Loss)

print("Accuracy :", Accuracy)

plot\_history(hist)

output = output.append(pd.DataFrame(pd.DataFrame({"Model":["Base Model"],

"Accuracy" :[Accuracy],

"Loss" : [Loss]})), ignore\_index=True)

### Base Model with L1 Regularization

model = Sequential([Normalization(),

Dense(64, activation="relu", kernel\_regularizer='l1'),

Dense(16, activation="relu", kernel\_regularizer='l1'),

Dense(1, activation="sigmoid")])

optimizer = keras.optimizers.Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss="BinaryCrossentropy", metrics="Accuracy")

hist = model.fit(x\_train, y\_train, epochs=10, verbose=False, validation\_data=(x\_test,y\_test))

Loss, Accuracy = model.evaluate(x\_test, y\_test, verbose=False)

print("Validation Result")

print("Loss :", Loss)

print("Accuracy :", Accuracy)

plot\_history(hist)

output = output.append(pd.DataFrame(pd.DataFrame({"Model":["L1"],

"Accuracy" :[Accuracy],

"Loss" : [Loss]})), ignore\_index=True)

### Base Model with L2 Regularization

model = Sequential([Normalization(),

Dense(64, activation="relu", kernel\_regularizer='l2'),

Dense(16, activation="relu", kernel\_regularizer='l2'),

Dense(1, activation="sigmoid")])

optimizer = keras.optimizers.Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss="BinaryCrossentropy", metrics="Accuracy")

hist = model.fit(x\_train, y\_train, epochs=10, verbose=False, validation\_data=(x\_test,y\_test))

Loss, Accuracy = model.evaluate(x\_test, y\_test, verbose=False)

print("Validation Result")

print("Loss :", Loss)

print("Accuracy :", Accuracy)

plot\_history(hist)

output = output.append(pd.DataFrame(pd.DataFrame({"Model":["L2"],

"Accuracy" :[Accuracy],

"Loss" : [Loss]})), ignore\_index=True)

### Base Model with Dropout

model = Sequential([Normalization(),

Dense(64, activation="relu"),

Dropout(0.2),

Dense(16, activation="relu"),

Dropout(0.5),

Dense(1, activation="sigmoid")])

optimizer = keras.optimizers.Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss="BinaryCrossentropy", metrics="Accuracy")

hist = model.fit(x\_train, y\_train, epochs=10, verbose=False, validation\_data=(x\_test,y\_test))

Loss, Accuracy = model.evaluate(x\_test, y\_test, verbose=False)

print("Validation Result")

print("Loss :", Loss)

print("Accuracy :", Accuracy)

plot\_history(hist)

output = output.append(pd.DataFrame(pd.DataFrame({"Model":["Dropout"],

"Accuracy" :[Accuracy],

"Loss" : [Loss]})), ignore\_index=True)

### Base Model with Early Stopping

model = Sequential([Normalization(),

Dense(64, activation="relu"),

Dense(16, activation="relu"),

Dense(1, activation="sigmoid")])

optimizer = keras.optimizers.Adam(learning\_rate=0.001)

model.compile(optimizer=optimizer, loss="BinaryCrossentropy", metrics="Accuracy")

callback = keras.callbacks.EarlyStopping(monitor='loss', patience=5)

hist = model.fit(x\_train, y\_train, epochs=100, callbacks=[callback], verbose=False, validation\_data=(x\_test,y\_test))

Loss, Accuracy = model.evaluate(x\_test, y\_test, verbose=False)

print("Validation Result")

print("Loss :", Loss)

print("Accuracy :", Accuracy)

plot\_history(hist)

output = output.append(pd.DataFrame(pd.DataFrame({"Model":["Early Stopping"],

"Accuracy" :[Accuracy],

"Loss" : [Loss]})), ignore\_index=True)

### Linear Regression

import torch

import numpy as np

import matplotlib.pyplot as plt

x = torch.arange(-5,5,0.2).view(-1,1)

y = x\*0.69 + torch.rand(x.size())

# Initialize Weights

w = torch.randn(1, requires\_grad = True)

b = torch.randn(1, requires\_grad = True)

# Training Loop

def predict():

return (x\*w + b)

def calc\_loss(y\_pred):

return (((y - y\_pred)\*\* 2).sum() / 2\*x.size()[0])

def train(epoch = 10000, lr = 0.0000001):

global w, b

for \_ in range(epoch):

# Predicition

y\_pred = predict()

# Loss

loss = calc\_loss(y\_pred)

# Calculate Grad

loss.backward()

with torch.no\_grad():

w -= w.grad \* lr

b -= b.grad \* lr

w.grad.zero\_()

b.grad.zero\_()

train()

print("Loss :", calc\_loss(predict()))

plt.scatter(x, y)

plt.plot(x.detach().numpy(), predict().detach().numpy(), c="red")

plt.legend(["Ground Truth", "Predict"][::-1])

plt.show()

**Result:** Thus, Regularization techniques and Linear Regression have been implemented.

|  |  |
| --- | --- |
| **Ex. No: 5** | **Ablation Studies - CNN** |
|  |

**Aim:** Perform Ablation Studies on a CNN Architecture.

**Algorithm:**

1. Import the necessary packages.
2. Load The Dataset.
3. Pre-Process the datasets with the respected methods.
4. Train an ANN and find the accuracy.
5. Tweak the Architecture among various options.
6. Tabulate the observations.
7. Plot the Inference for better understanding.

**Program :**

import numpy as np

import pandas as pd

import random

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

from sklearn.metrics import classification\_report

from tensorflow.python import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Flatten, Conv2D, Dropout, MaxPooling2D, BatchNormalization

from IPython.display import SVG

from keras.utils.vis\_utils import model\_to\_dot

from keras.utils import plot\_model

import seaborn as sns

from keras.utils import np\_utils

import matplotlib.pyplot as plt

%matplotlib inline

## Parameters

IMG\_ROWS = 28

IMG\_COLS = 28

NUM\_CLASSES = 10

TEST\_SIZE = 0.1

RANDOM\_STATE = 2018

#Model

NO\_EPOCHS = 150

PATIENCE = 20

VERBOSE = 1

BATCH\_SIZE = 512

PATH="/home/ai\_ds-b1/Downloads/"

train\_file = PATH+"train.csv"

test\_file = PATH+"test.csv"

train\_df = pd.read\_csv(train\_file)

test\_df = pd.read\_csv(test\_file)

print("MNIST train - rows:",train\_df.shape[0]," columns:", train\_df.shape[1])

print("MNIST test - rows:",test\_df.shape[0]," columns:", test\_df.shape[1])

# data preprocessing

def data\_preprocessing(raw, hasLabel=True):

start\_pixel = 0

if(hasLabel):

start\_pixel = 1

if(hasLabel):

out\_y = np\_utils.to\_categorical(raw.label, NUM\_CLASSES)

else:

out\_y = None

num\_images = raw.shape[0]

x\_as\_array = raw.values[:,start\_pixel:]

x\_shaped\_array = x\_as\_array.reshape(num\_images, IMG\_ROWS, IMG\_COLS, 1)

out\_x = x\_shaped\_array / 255

return out\_x, out\_y

# prepare the data

X, y = data\_preprocessing(train\_df)

X\_test, y\_test = data\_preprocessing(test\_df,hasLabel=False)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, test\_size=TEST\_SIZE, random\_state=RANDOM\_STATE)

print("MNIST train - rows:",X\_train.shape[0]," columns:", X\_train.shape[1:4])

print("MNIST valid - rows:",X\_val.shape[0]," columns:", X\_val.shape[1:4])

print("MNIST test - rows:",X\_test.shape[0]," columns:", X\_test.shape[1:4])

# Model

model = Sequential()

# Add convolution 2D

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

kernel\_initializer='he\_normal',input\_shape=(IMG\_ROWS, IMG\_COLS, 1)))

# model.add(BatchNormalization())

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

model.add(Conv2D(32,kernel\_size=5,strides=2,padding='same',activation='relu'))

model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, kernel\_size=(3, 3), strides=2,padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.4))

model.add(Dense(NUM\_CLASSES, activation='softmax'))

# Compile the model

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

### Inspect the model

Let's check the model we initialized.

model.summary()

NO\_EPOCHS = 5

from keras.callbacks import EarlyStopping, ModelCheckpoint

earlystopper = EarlyStopping(monitor='loss', patience=PATIENCE, verbose=VERBOSE)

checkpointer = ModelCheckpoint('best\_model.h5',

monitor='val\_acc',

verbose=VERBOSE,

save\_best\_only=True,

save\_weights\_only=True)

history = model.fit(X\_train, y\_train,

batch\_size=BATCH\_SIZE,

epochs=NO\_EPOCHS,

verbose=1,

validation\_data=(X\_val, y\_val),

callbacks=[earlystopper, checkpointer])

print("run model - predict validation set")

score = model.evaluate(X\_val, y\_val, verbose=0)

print(f'Last validation loss: {score[0]}, accuracy: {score[1]}')

score = model\_optimal.evaluate(X\_val, y\_val, verbose=0)

print(f'Best validation loss: {score[0]}, accuracy: {score[1]}')

pred\_y = np.argmax(model.predict(X\_val),axis=1)

y\_val = np.argmax(y\_val,axis=1)

from sklearn.metrics import confusion\_matrix

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

sns.heatmap(confusion\_matrix(y\_val,pred\_y),annot=True)

**Result:** Thus, using TensorFlow library, Ablation Studies on CNN have been performed.

|  |  |
| --- | --- |
| **Ex. No: 6** | **CNN vs ANN - MNIST** |
|  |

**Aim:** To write a program in TensorFlow to compare results between ANN and CNN on MNIST.

**Algorithm:**

1. Import the necessary packages.
2. Load the Dataset.
3. Pre-Process the datasets with the respected methods.
4. Implement ANN and CNN.
5. Compare the results.
6. Plot the Inference.

**Program :**

import pandas as pd

import numpy as np

import tensorflow as tf

import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten

### ANN

data = keras.datasets.mnist

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

print(x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape)

x\_train = x\_train.reshape(60000, 784)

x\_test = x\_test.reshape(10000, 784)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

x\_train = scaler.fit\_transform(x\_train)

x\_test = scaler.transform(x\_test)

model = Sequential()

model.add(Dense(128, activation='relu', input\_shape = (784,)))

model.add(Dense(64, activation='relu'))

model.add(Dense(32, activation='relu'))

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

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

model.summary()

model\_history = model.fit(x\_train, y\_train, epochs=10, verbose=0)

print("Accuracy : ", model.evaluate(x\_test, y\_test, verbose=0)[1])

### CNN

data = keras.datasets.mnist

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

print(x\_train.shape, y\_train.shape, x\_test.shape, y\_test.shape)

model = Sequential()

model.add(Conv2D(32, kernel\_size=(5, 5), activation='relu', input\_shape=(28, 28, 1)))

model.add(MaxPooling2D((2, 2)))

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

model.add(MaxPooling2D((2, 2)))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

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

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

model.summary()

model\_history = model.fit(x\_train, y\_train, epochs=10, verbose=0, batch\_size=512)

print("Accuracy : ", model.evaluate(x\_test, y\_test, verbose=0)[1])

**Result:** Thus, using TensorFlow library, CNN and ANN have been implemented and compared.

|  |  |
| --- | --- |
| **Ex. No: 7** | **Sentiment analysis using RNN** |
|  |

**Aim:** To write a program in TensorFlow to perform Sentiment analysis using RNN.

**Algorithm:**

1. Import the necessary packages.
2. Load IMDB dataset.
3. Pre-Process the Datasets
4. Train a RNN model with the set of Hyper-Parameters.
5. Display the accuracy.
6. Plot the Accuracy metrices.

**Program :**

import numpy as np

import pandas as pd

import tensorflow as tf

import keras

(train\_x, train\_y) , (test\_x, test\_y) = keras.datasets.imdb.load\_data(num\_words=10000)

print("Shape")

print("Train X : ", train\_x.shape)

print("Train Y : ", train\_y.shape)

print("Test X : ", test\_x.shape)

print("Test Y : ", test\_y.shape)

print("Maximum Value of word Index :", max([max(i) for i in train\_x]))

print("Maximum Length of word:", max([len(i) for i in train\_x]))

from keras.preprocessing.sequence import pad\_sequences

train\_x = pad\_sequences(train\_x, maxlen=269)

test\_x = pad\_sequences(test\_x, maxlen=269)

from keras.models import Sequential

rnn = Sequential()

rnn.add(keras.layers.Embedding(10000,32,input\_length =269))

rnn.add(keras.layers.SimpleRNN(16, activation="relu"))

rnn.add(keras.layers.Dense(1))

rnn.add(keras.layers.Activation("sigmoid"))

print(rnn.summary())

rnn.compile(loss="binary\_crossentropy", optimizer="rmsprop", metrics=["accuracy"])

history = rnn.fit(train\_x, train\_y, epochs = 10, batch\_size=128, verbose = 1)

score = rnn.evaluate(test\_x, test\_y, verbose=0)

print("Accuracy", score[1]\*100)

**Result:** Thus, using TensorFlow library, RNN have been implemented to find sentiment.

|  |  |
| --- | --- |
| **Ex. No: 8** | **LSTM vs GRU** |
|  |

**Aim:** Load a text sequence and a vector sequence dataset of your choice and apply LSTM and GRU model to compare the accuracy and time complexity for both models.

**Algorithm:**

1. Import the necessary packages.
2. Load text sequence and vector sequence dataset.
3. Apply Pre-Processing techniques.
4. Train LSTM and GRU model.
5. Compare the accuracy and time complexity.

**Program :**

## Using Text Sequence

import pandas as pd

import tensorflow as tf

import numpy as np

Data = pd.read\_csv("training.csv")

Test = pd.read\_csv('test.csv')

Data.head()

Test.head()

def Pre(xyz):

Input = [i for i in xyz]

stop\_words = ['a','an','i','is','the','am','are','has','have','you','she','it','he','him','her','had','that','there','where','when','why'\

'while','though','this','can','go','so']

I1 = [i1.split() for i1 in Input]

I2 = [ [i for i in j if i not in stop\_words] for j in I1]

I1 =[]

for i in I2:

for j in i:

if j not in I1:

I1.append(j)

return I1

Inpt = Data['text']

out = Data['label']

Input = [i for i in Data['text']]

from sklearn.preprocessing import OneHotEncoder

OE = OneHotEncoder()

Out = OE.fit\_transform(np.array(Data['label']).reshape(-1,1)).toarray()

stop\_words = ['a','an','i','is','the','am','are','has','have','you','she','it','he','him','her','had','that','there','where','when','why'\

'while','though','this','can','go','so']

I1 = [i1.split() for i1 in Input]

I2 = [ [i for i in j if i not in stop\_words] for j in I1]

I1 =[]

for i in I2:

for j in i:

if j not in I1:

I1.append(j)

Size = 300

Input = pd.DataFrame(data = 0 , index =[i for i in range(len(I2))],columns=I1[:Size])

for i in range(len(I2)):

for j in range(len(I2[i])):

Input.iloc[i][I2[i][j]] = 1

import tensorflow as tf

from keras.layers import LSTM,Dense,GRU

from keras.models import Sequential

from keras.losses import CategoricalCrossentropy

model = Sequential()

model.add(LSTM(units=100,activation='relu',input\_shape=(1,Size),return\_sequences=True))

model.add(Dense(Out.shape[1],activation='softmax'))

model.compile(optimizer='adam',loss=CategoricalCrossentropy(),metrics=['accuracy'])

model.summary()

model.fit(Input.values.reshape(-1,1,Size),Out.reshape(-1,1,6),epochs=300)

Input\_test = Pre(Test['text'])

Input\_t = pd.DataFrame(data = 0 , index =[i for i in range(len(I2))],columns=I1[:Size])

for i in range(len(Input\_test)):

for j in range(len(Input\_test[i])):

Input\_t.iloc[i][Input\_test[i][j]] = 1

Out = OE.fit\_transform(np.array(Test['label']).reshape(-1,1)).toarray()

model = Sequential()

model.add(GRU(units=100,activation='relu',input\_shape=(1,Size),return\_sequences=True))

model.add(Dense(Out.shape[1],activation='softmax'))

model.compile(optimizer='adam',loss=CategoricalCrossentropy(),metrics=['accuracy'])

model.summary()

model.fit(Input.values.reshape(-1,1,Size),Out.reshape(-1,1,6),epochs=300)

from sklearn.datasets import load\_digits

data = load\_digits()

inp.shape

inp = data['data']

out\_vec = OE.fit\_transform(data['target'].reshape(-1,1)).toarray()

model = Sequential()

model.add(LSTM(units=100,activation='relu',input\_shape=(1,inp.shape[-1]),return\_sequences=True))

model.add(Dense(out\_vec.shape[-1],activation='softmax'))

model.compile(optimizer='adam',loss=CategoricalCrossentropy(),metrics=['accuracy'])

model.fit(inp.reshape(inp.shape[0],1,inp.shape[1]),out\_vec.reshape(out\_vec.shape[0],1,out\_vec.shape[1]),epochs=100)

model = Sequential()

model.add(GRU(units=100,activation='relu',input\_shape=(1,inp.shape[-1]),return\_sequences=True))

model.add(Dense(out\_vec.shape[-1],activation='softmax'))

model.compile(optimizer='adam',loss=CategoricalCrossentropy(),metrics=['accuracy'])

model.fit(inp.reshape(inp.shape[0],1,inp.shape[1]),out\_vec.reshape(out\_vec.shape[0],1,out\_vec.shape[1]),epochs=100)

## END

**Result:** Thus using TensorFlow library, LSTM and GRU have been implemented and compared.

|  |  |
| --- | --- |
| **Ex. No: 9** | **Text Generation using Gated Recurrent Unit Networks** |
|  |

**Aim:** To Generation text using Gated Recurrent Unit Networks.

**Algorithm:**

Step 1: Importing the required libraries.

Step 2: Loading the data into a string.

Step 3: Creating a mapping from each unique character in the text to a unique number.

Step 4: Pre-processing the data.

Step 5: Building the GRU network.

Step 6: Defining some helper functions which will be used during the training of the network.

a) Helper function to sample the next character:

b) Helper function to generate text after each epoch

c) Helper function to save the model after each epoch in which loss decreases

d) Helper function to reduce the learning rate each time the learning plateaus

Step 7: Training the GRU model (Take 15 epochs, batch size is individual decision (for e.g 128))

Step 8: Generating new and random text.

**Program:**

import numpy as np

import tensorflow as tf

from keras.models import Sequential

from keras.layers import Dense, Activation

from keras.layers import LSTM, GRU

from keras.optimizers import RMSprop

from keras.callbacks import LambdaCallback

from keras.callbacks import ModelCheckpoint

from keras.callbacks import ReduceLROnPlateau

import random

import sys

with open('poems.txt', 'r') as file:

text = file.read()

print(text)

vocabulary = sorted(list(set(text)))

char\_to\_indices = dict((c, i) for i, c in enumerate(vocabulary))

indices\_to\_char = dict((i, c) for i, c in enumerate(vocabulary))

print(vocabulary)

max\_length = 100

steps = 5

sentences = []

next\_chars = []

for i in range(0, len(text) - max\_length, steps):

sentences.append(text[i: i + max\_length])

next\_chars.append(text[i + max\_length])

X = np.zeros((len(sentences), max\_length, len(vocabulary)), dtype = np.bool)

y = np.zeros((len(sentences), len(vocabulary)), dtype = np.bool)

for i, sentence in enumerate(sentences):

for t, char in enumerate(sentence):

X[i, t, char\_to\_indices[char]] = 1

y[i, char\_to\_indices[next\_chars[i]]] = 1

model = Sequential()

model.add(GRU(128, input\_shape =(max\_length, len(vocabulary))))

model.add(Dense(len(vocabulary)))

model.add(Activation('softmax'))

optimizer = RMSprop(lr = 0.01)

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

def sample\_index(preds, temperature = 1.0):

preds = np.asarray(preds).astype('float64')

preds = np.log(preds) / temperature

exp\_preds = np.exp(preds)

preds = exp\_preds / np.sum(exp\_preds)

probas = np.random.multinomial(1, preds, 1)

return np.argmax(probas)

def on\_epoch\_end(epoch, logs):

print()

print('----- Generating text after Epoch: % d' % epoch)

start\_index = random.randint(0, len(text) - max\_length - 1)

for diversity in [0.2, 0.5, 1.0, 1.2]:

print('----- diversity:', diversity)

generated = ''

sentence = text[start\_index: start\_index + max\_length]

generated += sentence

print('----- Generating with seed: "' + sentence + '"')

sys.stdout.write(generated)

for i in range(400):

x\_pred = np.zeros((1, max\_length, len(vocabulary)))

for t, char in enumerate(sentence):

x\_pred[0, t, char\_to\_indices[char]] = 1.

preds = model.predict(x\_pred, verbose = 0)[0]

next\_index = sample\_index(preds, diversity)

next\_char = indices\_to\_char[next\_index]

generated += next\_char

sentence = sentence[1:] + next\_char

sys.stdout.write(next\_char)

sys.stdout.flush()

print\_callback = LambdaCallback(on\_epoch\_end = on\_epoch\_end)

filepath = "weights.hdf5"

checkpoint = ModelCheckpoint(filepath, monitor ='loss',

verbose = 1, save\_best\_only = True,

mode ='min')

reduce\_alpha = ReduceLROnPlateau(monitor ='loss', factor = 0.2,

patience = 1, min\_lr = 0.001)

callbacks = [print\_callback, checkpoint, reduce\_alpha]

model.fit(X, y, batch\_size = 128, epochs = 15, callbacks = callbacks)

def generate\_text(length, diversity):

start\_index = random.randint(0, len(text) - max\_length - 1)

generated = ''

sentence = text[start\_index: start\_index + max\_length]

generated += sentence

for i in range(length):

x\_pred = np.zeros((1, max\_length, len(vocabulary)))

for t, char in enumerate(sentence):

x\_pred[0, t, char\_to\_indices[char]] = 1.

preds = model.predict(x\_pred, verbose = 0)[0]

next\_index = sample\_index(preds, diversity)

next\_char = indices\_to\_char[next\_index]

generated += next\_char

sentence = sentence[1:] + next\_char

return generated

print(generate\_text(500, 0.2))

**Result:** Thus, using TensorFlow library, GRU have been implemented to generate text.

|  |  |
| --- | --- |
| **Ex. No: 10** | **Object Detection** |
|  |

**Aim:** To use convolutional neural network (CNN) for object detection from CIFAR-10 dataset.

**Algorithm:**

1. Import the necessary packages.
2. Load CIFAR-10 dataset.
3. Pre-process the Datasets
4. Train a CNN model.
5. Display the accuracy.

**Program :**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

(x\_train, y\_train), (x\_test, y\_test) = keras.datasets.cifar10.load\_data()

x\_train = x\_train/255

x\_test = x\_test/255

y\_train = keras.utils.to\_categorical(y\_train)

y\_test = keras.utils.to\_categorical(y\_test)

# Model Building

model = keras.Sequential(

[

keras.layers.Conv2D(filters=32,kernel\_size=(5,5),padding="same",activation="relu",input\_shape=(32,32,3)),

keras.layers.MaxPooling2D((2,2)),

keras.layers.Conv2D(filters=64,kernel\_size=(5,5),padding="same",activation="relu"),

keras.layers.MaxPooling2D((2,2)),

keras.layers.Conv2D(filters=128,kernel\_size=(5,5),padding="same",activation="relu"),

keras.layers.MaxPooling2D((2,2)),

keras.layers.Flatten(),

keras.layers.Dense(512,activation="relu"),

keras.layers.Dense(128,activation="relu"),

keras.layers.Dense(10,activation="softmax")

])

# Compile Model

opt = keras.optimizers.SGD(learning\_rate=0.001, momentum=0.9)

model.compile(optimizer=opt, loss="categorical\_crossentropy", metrics=["acc"])

# Train Model

model\_history = model.fit(x\_train,y\_train,epochs=5, batch\_size=4, validation\_data=(x\_test,y\_test))

# Accuracy

\_, accuracy = model.evaluate(x\_test,y\_test,verbose=0)

print("Accuracy :", accuracy)

model.summary()

**Result:** Thus, using TensorFlow library, CNN have been implemented for object detection.

|  |  |
| --- | --- |
| **Ex. No: 11** | **Fraud Detection** |
|  |

**Aim:** Credit card fraud detection using creditcard.csv dataset and using random forest. Print the amount details for Normal Transaction, plot the Correlation Matrix, print accuracy, precision, recall and F1 score, print confusion matrix.

**Algorithm:**

1. Import the necessary packages.
2. Load Credit Card Dataset.
3. Pre-Process the datasets,
4. Train a random forest classifier.
5. Display the accuracy, precision, recall, F1 score and confusion matrix.

**Program :**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

df = pd.read\_csv("creditcard.csv")

print("Shape :", df.shape)

df.head()

# Columns

df.columns

df.describe()

### Amount details for Normal Transaction

df[df["Class"] == 0]["Amount"].describe()

### DownSampling

df["Class"].value\_counts()

from sklearn.utils import resample

zero\_df = resample(df[df["Class"] == 0], n\_samples=492)

sample\_df = pd.concat([df[df["Class"] == 1], zero\_df], ignore\_index=True)

### Correlation Matrix

sample\_df.corr()

### Random Forest

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

x = sample\_df.iloc[:,:-1].values

y = sample\_df.iloc[:, -1].values

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, random\_state=69, test\_size=0.2)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

### Classification Report

from sklearn.metrics import classification\_report

print(classification\_report(y\_test, y\_pred))

from sklearn.metrics import confusion\_matrix

print(confusion\_matrix(y\_test, y\_pred))

**Result:** Thus, using Sklearn library, fraud detection have been implemented.

|  |  |
| --- | --- |
| **Ex. No: 12** | **Autoencoder** |
|  |

**Aim:** To implement vanilla autoencoder on MNIST dataset and calculate the loss vs. epoch curve for training and validation set.

**Algorithm:**

1. Import the necessary packages.
2. Load MNIST dataset.
3. Train the autoencoder model.
4. Display the accuracy.

**Program :**

from tensorflow.keras.datasets import mnist

from tensorflow.keras.layers import Dense, Input, Flatten, Reshape, LeakyReLU as LR, Activation, Dropout

from tensorflow.keras.models import Model, Sequential

from IPython import display

from matplotlib import pyplot as plt

import numpy as np

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

x\_train = x\_train/255.0

x\_test = x\_test/255.0

plt.imshow(x\_train[0], cmap = "gray")

plt.show()

LATENT\_SIZE = 32

encoder = Sequential([

Flatten(input\_shape = (28, 28)),

Dense(512),

LR(),

Dropout(0.5),

Dense(256),

LR(),

Dropout(0.5),

Dense(128),

LR(),

Dropout(0.5),

Dense(64),

LR(),

Dropout(0.5),

Dense(LATENT\_SIZE),

LR()

])

decoder = Sequential([

Dense(64, input\_shape = (LATENT\_SIZE,)),

LR(),

Dropout(0.5),

Dense(128),

LR(),

Dropout(0.5),

Dense(256),

LR(),

Dropout(0.5),

Dense(512),

LR(),

Dropout(0.5),

Dense(784),

Activation("sigmoid"),

Reshape((28, 28))

])

img = Input(shape = (28, 28))

latent\_vector = encoder(img)

output = decoder(latent\_vector)

model = Model(inputs = img, outputs = output)

model.compile("nadam", loss = "binary\_crossentropy")

EPOCHS = 10

for epoch in range(EPOCHS):

fig, axs = plt.subplots(4, 4)

rand = x\_test[np.random.randint(0, 10000, 16)].reshape((4, 4, 1, 28, 28))

display.clear\_output()

for i in range(4):

for j in range(4):

axs[i, j].imshow(model.predict(rand[i, j])[0], cmap = "gray")

axs[i, j].axis("off")

plt.subplots\_adjust(wspace = 0, hspace = 0)

plt.show()

print("-----------", "EPOCH", epoch, "-----------")

model.fit(x\_train, x\_train)

**Result:** Thus, using TensorFlow library, Autoencoder has been implemented.

|  |  |
| --- | --- |
| **Ex. No: 13** | **Anomaly Detection System** |
|  |

**Aim:** To develop an anomaly detection system, which is trained by the normal data only but will be used to identify the normal data and the anomaly data during testing.

**Algorithm:**

1. Import the necessary packages.
2. Load MNIST as normal data and Fashion MNIST as anomaly data.
3. Pre-process the data.
4. Train a CNN Autoencoder model.
5. Now test the trained model with the anomaly data and display the outputs.
6. Display the accuracy for anomaly data.

**Program :**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

from keras import datasets

digit = datasets.mnist.load\_data()

fashion = datasets.fashion\_mnist.load\_data()

# Train-test Split - DIGIT

x\_train = digit[0][0]

x\_test = digit[1][0]

# Fashion - Test

fashion\_test = fashion[1][0]

# DIGIT

x\_train = x\_train / 255

x\_test = x\_test / 255

x\_train = x\_train.reshape(len(x\_train), 28, 28, 1)

x\_test = x\_test.reshape(len(x\_test), 28, 28, 1)

# FASHION

fashion\_test = fashion\_test / 255

fashion\_test = fashion\_test.reshape(len(fashion\_test), 28, 28, 1)

def plot\_image(array, sample\_size = 5):

index = 1

plt.figure(figsize=(20, 4))

for i in np.random.choice(array.shape[0], size = sample\_size):

ax = plt.subplot(2, 10, index)

index += 1

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

plt.gray()

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

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

plt.show()

plot\_image(x\_train)

plot\_image(fashion\_test)

### Convolutional Autoencoder

import keras

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

x = keras.layers.Conv2D(16, (3, 3), activation='relu', padding='same')(input\_img)

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

x = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)

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

x = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)

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

x = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)

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

x = keras.layers.Conv2D(8, (3, 3), activation='relu', padding='same')(x)

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

x = keras.layers.Conv2D(16, (3, 3), activation='relu')(x)

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

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

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

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

autoencoder.summary()

autoencoder.fit(x\_train, x\_train, epochs=25, batch\_size=64, validation\_data=(x\_test, x\_test))

digit\_predict = autoencoder.predict(x\_test)

fashion\_predict = autoencoder.predict(fashion\_test)

# Regenerated Image

plot\_image(digit\_predict)

plot\_image(fashion\_predict)

# Fixing Threshold

reconstruction\_error\_digit = []

reconstruction\_error\_fashion = []

for i in x\_test[:100]:

error = autoencoder.evaluate([i], [i], verbose=0)[0]

reconstruction\_error\_digit.append(error)

for i in fashion\_test[:100]:

error = autoencoder.evaluate([i], [i], verbose=0)[0]

reconstruction\_error\_fashion.append(error)

reconstruction\_error\_digit = np.array(reconstruction\_error\_digit)

reconstruction\_error\_fashion = np.array(reconstruction\_error\_fashion)

threshold = [reconstruction\_error\_digit.mean() - 2\*reconstruction\_error\_digit.std(), reconstruction\_error\_digit.mean() + 2\*reconstruction\_error\_digit.std()]

def detect\_anomaly(image):

error = autoencoder.evaluate([image], [image], verbose=0)[0]

if error >= threshold[0] and error <= threshold[1]:

print("Noice !!!")

return 0

else:

print("Anomaly Detected")

return 1

accuracy = 0

for i in fashion\_test[10:20]:

accuracy += detect\_anomaly(i)

print("Accuracy : ", accuracy)

**Result:** Thus, using TensorFlow library, Anomaly detection system has been implemented.