PRACTICAL NO. :- 1

AIM : Creating a basic network and analyze its performance.

INPUT

```
#import python libraries required:

from keras.models import Sequential
from keras.layers import Dense , Activation
import numpy as np

#use numpy arrays to store inputs(x) and outputs(y):

x=np.array([[2,2],[2,4],[2,5],[4,5], [4,8], [4,6]])
y=np.array([[2],[2],[4],[5], [6], [8]])

#define the network model and its arguments

#set the number of neurons/nodes for each layer:

model = Sequential()
model.add(Dense(4, input_shape=(2,)))
model.add(Activation('linear'))
model.add(Dense(2))
model.add(Activation('linear'))

#compile the model and calculate its accuracy

model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])

#print a summay of the Keras model:

model.summary()
```

OUTPUT

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4)                 12

 activation (Activation)     (None, 4)                 0

 dense_1 (Dense)             (None, 2)                 10

 activation_1 (Activation)   (None, 2)                 0

=================================================================
Total params: 22 (88.00 Byte)
Trainable params: 22 (88.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

PRACTICAL NO. :- 2

AIM : Deploy the Confusion Matrix and simulate for Overfitting.

INPUT

```
# Deploy the Confusion matrix and simulate for Overfitting
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

# Load the Iris dataset
iris = load_iris()
X, y = iris.data, iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a simple feedforward neural network
model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_dim=4),
    keras.layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model on the training data
model.fit(X_train, y_train, epochs=200, batch_size=32, verbose=0)
```

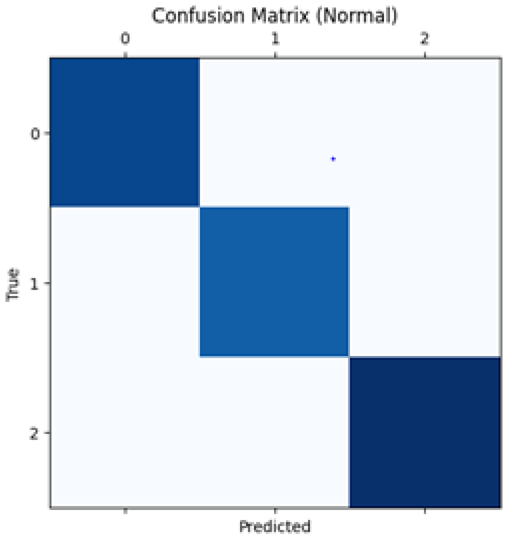
```
# Evaluate the model on the test data
y_pred = model.predict(X_test)
y_pred_classes = [tf.argmax(x).numpy() for x in y_pred]

# Generate a confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Simulate overfitting by training for more epochs
overfit_model = keras.Sequential([
    keras.layers.Dense(64, activation='relu', input_dim=4),
    keras.layers.Dense(3, activation='softmax')
])

overfit_model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

# Train the model on the training data for too many epochs
overfit_model.fit(X_train, y_train, epochs=1000, batch_size=32, verbose=0)

# Evaluate the overfit model on the test data
y_pred_overfit = overfit_model.predict(X_test)
y_pred_overfit_classes = [tf.argmax(x).numpy() for x in y_pred_overfit]

# Generate a confusion matrix for the overfit model
conf_matrix_overfit = confusion_matrix(y_test, y_pred_overfit_classes)
print("\nConfusion Matrix for Overfit Model:")
print(conf_matrix_overfit)

# Plot confusion matrices
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
```

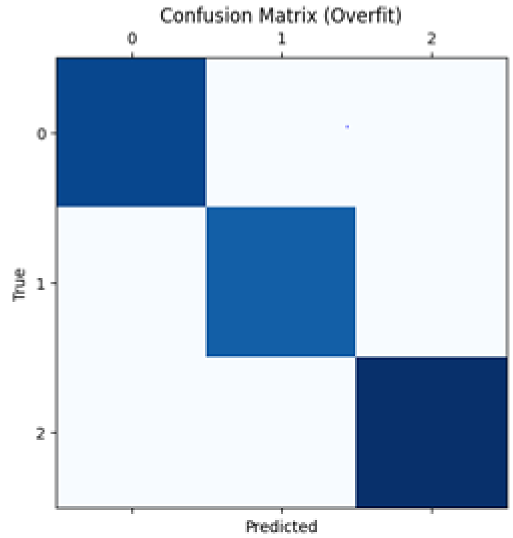
```
axes[0].matshow(conf_matrix, cmap=plt.cm.Blues, interpolation='nearest')
axes[0].set_title('Confusion Matrix (Normal)')
axes[0].set_xlabel('Predicted')
axes[0].set_ylabel('True')

axes[1].matshow(conf_matrix_overfit, cmap=plt.cm.Blues, interpolation='nearest')
axes[1].set_title('Confusion Matrix (Overfit)')
axes[1].set_xlabel('Predicted')
axes[1].set_ylabel('True')

plt.tight_layout()
plt.show()
```

OUTPUT

```
Confusion Matrix:
[[ 0  0 10]
 [ 0  0  9]
 [ 0  0 11]]
```

```
Confusion Matrix for Overfit Model:
[[10  0  0]
 [ 0  8  1]
 [ 0  0 11]]
```


Confusion Matrix (Normal)
0
1
2
True
Predicted


Confusion Matrix (Overfit)
0
1
2
True
Predicted

PRACTICAL NO. :- 3

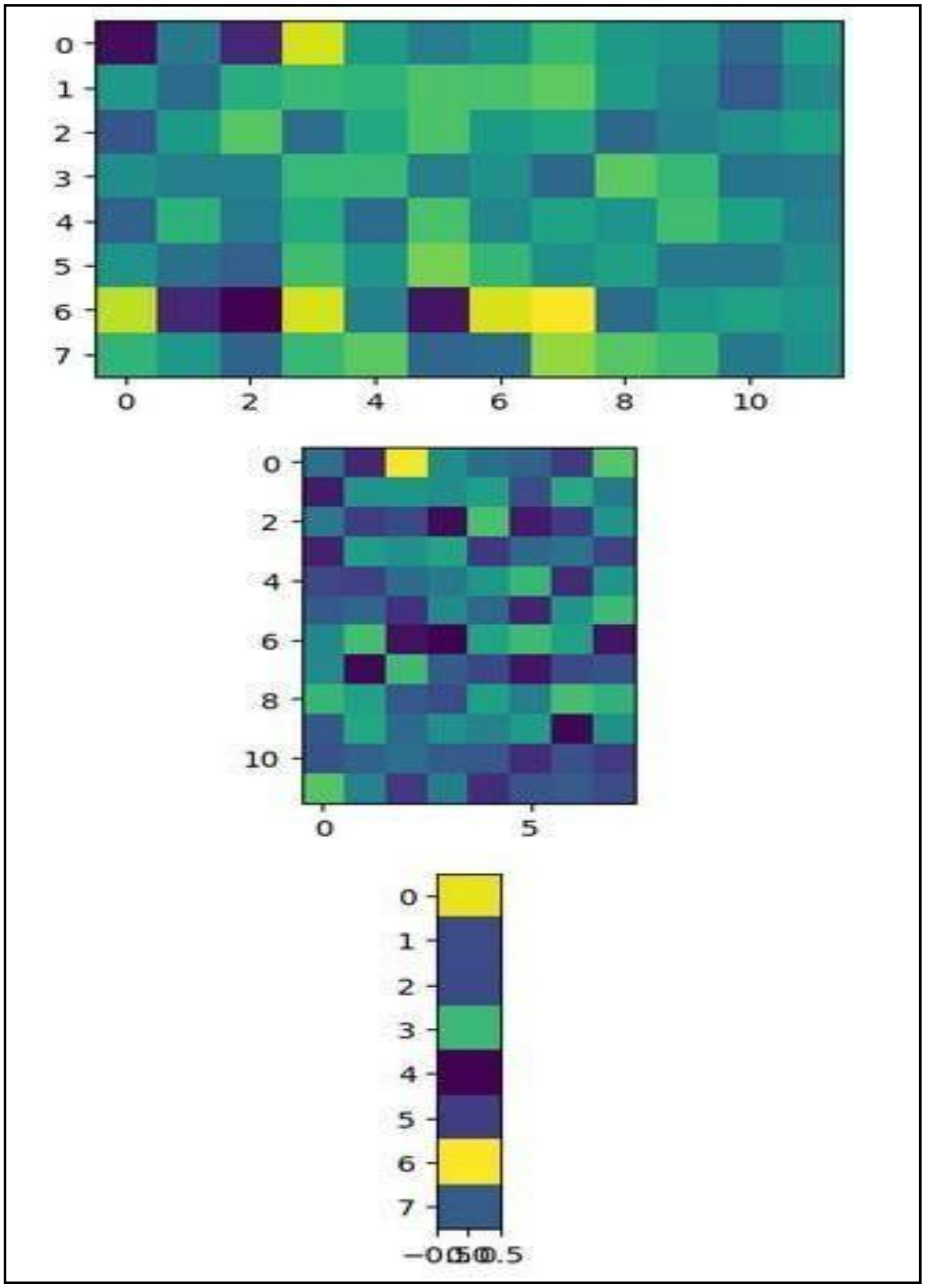
AIM : Visualizing Neural Networks.

INPUT

```
from keras.models import Sequential

from keras.layers import Dense

import numpy

from numpy import loadtxt


# fix random seed for reproducibility

numpy.random.seed(7)

# load pima indians dataset

dataset = numpy.loadtxt("/content/pima-indians-diabetes.csv", delimiter=",")


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

X = dataset[:,0:8]

Y = dataset[:,8]


# create model

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, Y, epochs=150, batch_size=10)
# evaluate the model
scores = model.evaluate(X, Y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
import numpy as np
import matplotlib.pyplot as plt

# Create a function to visualize the neural network.
def visualize_neural_network(model):
 # Get the number of layers in the neural network.
 num_layers = len(model.layers)
 # Create a figure and axes for each layer.
 fig, axes = plt.subplots(num_layers, 1, figsize=(10, 10))
 # Loop over the layers and plot the weights.
 for i, layer in enumerate(model.layers):
 # Get the weights for the layer.
 weights = layer.get_weights()
# Plot the weights.
axes[i].imshow(weights[0])

# Show the plot.
plt.show()
# Call the function to visualize the neural network.
visualize_neural_network(model)
```

OUTPUT



PRACTICAL NO. :- 4

AIM : Object Detection with pre trained RetinaNet with Keras.

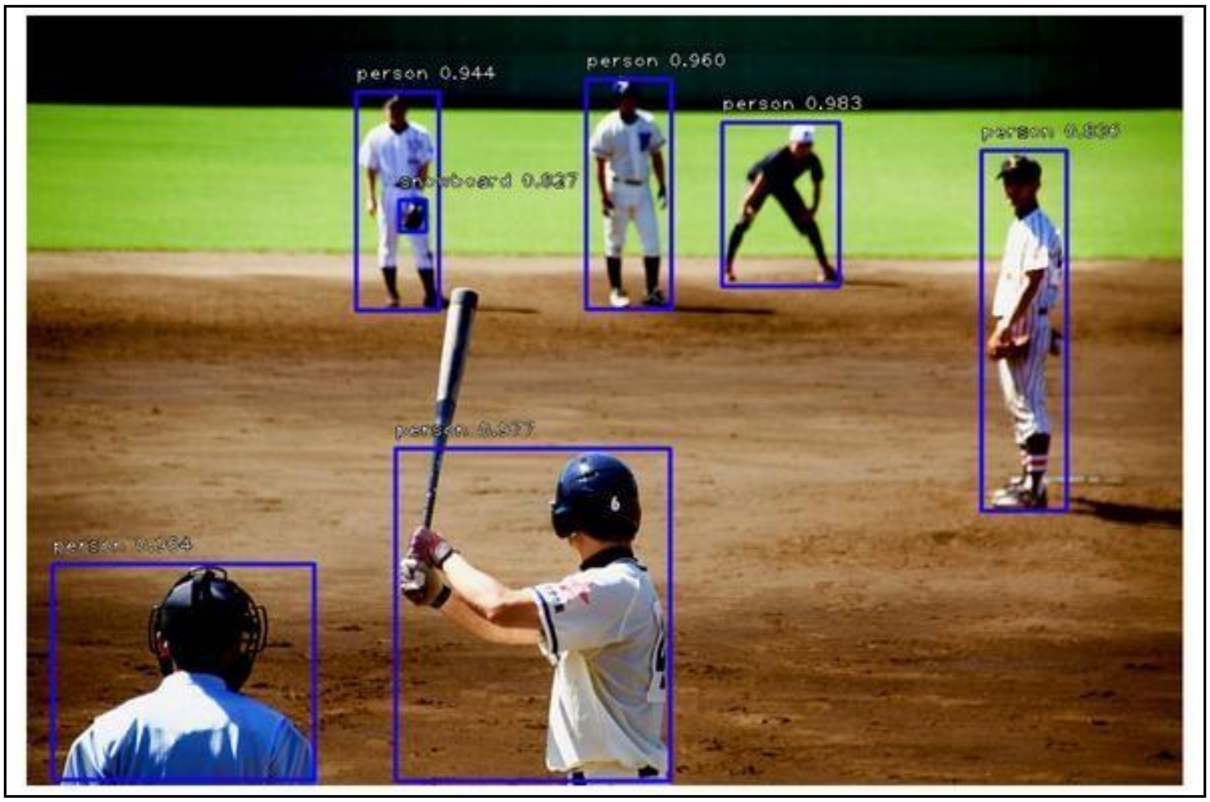
INPUT

```
!git clone https://github.com/fizyr/keras-retinanet.git

# Change to the keras-retinanet directory

%cd  keras-retinanet/

# Install dependencies

!pip install .

# Build the package

!python setup.py build_ext --inplace

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import urllib

import os

from PIL import Image

from keras_retinanet import models

from keras_retinanet.utils.image import preprocess_image, resize_image

from keras_retinanet.utils.visualization import draw_box, draw_caption

from keras_retinanet.utils.colors import label_color

# Download pretrained RetinaNet weights trained on the COCO dataset

urllib.request.urlretrieve('https://github.com/fizyr/keras-retinanet/releases/download/0.5.1/resnet50_coco_best_v2.1.0.h5',  'resnet50_coco_best_v2.1.0.h5')

# Load the model

model = models.load_model('resnet50_coco_best_v2.1.0.h5')

# Download COCO dataset labels
```

```
urllib.request.urlretrieve('https://raw.githubusercontent.com/amikelive/coco-labels/master/coco-labels-paper.txt', 'coco-labels-paper.txt')

class_labels = [label.rstrip() for label in open("coco-labels-paper.txt")]

def detect_draw_bounding_boxes(img_path, threshold=0.6):
    # Read image
    img = np.array(Image.open(img_path))
    print(f"Shape of the image: {img.shape}")
    # Remove the alpha channel from the image
    img = img[:, :, :3]
    # Preprocess and resize - mean subtraction and scaling
    img_proc = preprocess_image(img)
    img_proc, scale = resize_image(img_proc)
    print(f"Shape of the preprocessed image: {img_proc.shape}")
    boxes, scores, labels = model.predict_on_batch(np.expand_dims(img_proc, axis=0))
    # Standardize the boxes
    boxes /= scale
    for box, score, label in zip(boxes[0], scores[0], labels[0]):
        if score < threshold:
            break
        box = box.astype(np.int32)  # Box has to be integer
        color = label_color(label)
        draw_box(img, box, color=color)
        class_label = class_labels[label]
        caption = f"{class_label} {score:.3f}"
        draw_caption(img, box, caption)
    plt.axis('off')
    plt.imshow(img)
```

```
    plt.show()

# Download an example image

!wget https://c0.wallpaperflare.com/preview/814/948/832/de6l8nfk6nqltrackcl9liu6ss.jpg

# Set figure size

plt.rcParams['figure.figsize'] = [20, 10]

# Detect and draw bounding boxes on the example image

detect_draw_bounding_boxes('de6l8nfk6nqltrackcl9liu6ss.jpg')
```

OUTPUT



PRACTICAL NO. :- 5

AIM : Neural Recommender System with explicit feedback.

INPUT

```
%matplotlib inline

import matplotlib.pyplot as plt

import numpy as np

import os.path as op

from zipfile import ZipFile

try:

    from urllib.request import urlretrieve

except ImportError:  # Python 2 compat

    from urllib import urlretrieve

ML_100K_URL = "http://files.grouplens.org/datasets/movielens/ml-100k.zip"

ML_100K_FILENAME = ML_100K_URL.rsplit('/', 1)[1]

ML_100K_FOLDER = 'ml-100k'

if not op.exists(ML_100K_FILENAME):

    print('Downloading %s to %s...' % (ML_100K_URL, ML_100K_FILENAME))

    urlretrieve(ML_100K_URL, ML_100K_FILENAME)

if not op.exists(ML_100K_FOLDER):

    print('Extracting %s to %s...' % (ML_100K_FILENAME, ML_100K_FOLDER))

    ZipFile(ML_100K_FILENAME).extractall('.')


import pandas as pd

raw_ratings = pd.read_csv(op.join(ML_100K_FOLDER, 'u.data'), sep='\t',

                      names=["user_id", "item_id", "rating", "timestamp"])

raw_ratings.head()
```

	user_id	item_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

```
m_cols = ['item_id', 'title', 'release_date', 'video_release_date', 'imdb_url']

items = pd.read_csv(op.join(ML_100K_FOLDER, 'u.item'), sep='|',

                    names=m_cols, usecols=range(5), encoding='latin-1')

items.head()
```

	item_id	title	release_date	video_release_date	imdb_url
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)


```
m_cols = ['item_id', 'title', 'release_date', 'video_release_date', 'imdb_url']

items = pd.read_csv(op.join(ML_100K_FOLDER, 'u.item'), sep='|',

                    names=m_cols, usecols=range(5), encoding='latin-1')

items.head()
```

	item_id	title	release_date	video_release_date	imdb_url
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)

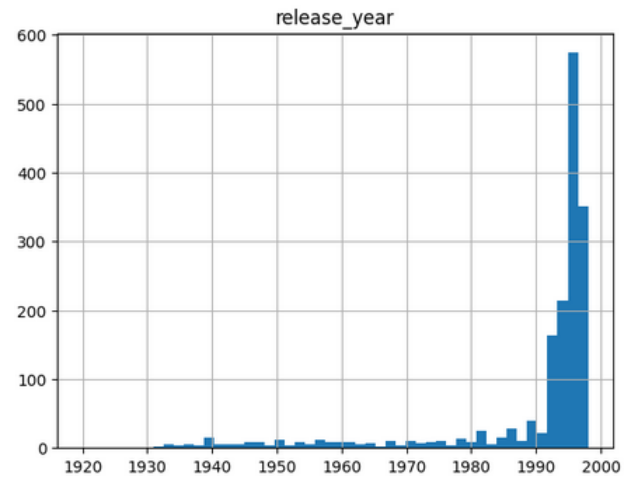

```
def extract_year(release_date):

    if hasattr(release_date, 'split'):

        components = release_date.split('-')

        if len(components) == 3:

            return int(components[2])
```

    # Missing value marker

    return 1920

items['release_year'] = items['release_date'].map(extract_year)

items.hist('release_year', bins=50);



all_ratings = pd.merge(items, raw_ratings)

all_ratings.head()

	item_id	title	release_date	video_release_date	imdb_url	release_year	user_id	rating	timestamp
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	308	4	887736532
1	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	287	5	875334088
2	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	148	4	877019411
3	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	280	4	891700426
4	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	66	3	883601324

min_user_id = all_ratings['user_id'].min()

min_user_id

1

max_user_id = all_ratings['user_id'].max()

max_user_id

943

min_item_id = all_ratings['item_id'].min()

min_item_id

1

max_item_id = all_ratings['item_id'].max()

max_item_id

1682

all_ratings['rating'].describe()

```
count    100000.000000
mean          3.529860
std           1.125674
min           1.000000
25%           3.000000
50%           4.000000
75%           4.000000
max           5.000000
Name: rating, dtype: float64
```

popularity = all_ratings.groupby('item_id').size().reset_index(name='popularity')

items = pd.merge(popularity, items)

items.nlargest(10, 'popularity')

	item_id	popularity	title	release_date	video_release_date	imdb_url	release_year
49	50	583	Star Wars (1977)	01-Jan-1977	NaN	http://us.imdb.com/M/title-exact?Star%20Wars%2...	1977
257	258	509	Contact (1997)	11-Jul-1997	NaN	http://us.imdb.com/Title?Contact+(1997/I)	1997
99	100	508	Fargo (1996)	14-Feb-1997	NaN	http://us.imdb.com/M/title-exact?Fargo%20(1996)	1997
180	181	507	Return of the Jedi (1983)	14-Mar-1997	NaN	http://us.imdb.com/M/title-exact?Return%20of%2...	1997
293	294	485	Liar Liar (1997)	21-Mar-1997	NaN	http://us.imdb.com/Title?Liar+Liar+(1997)	1997
285	286	481	English Patient, The (1996)	15-Nov-1996	NaN	http://us.imdb.com/M/title-exact?English%20Pat...	1996
287	288	478	Scream (1996)	20-Dec-1996	NaN	http://us.imdb.com/M/title-exact?Scream%20(1996)	1996
0	1	452	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995
299	300	431	Air Force One (1997)	01-Jan-1997	NaN	http://us.imdb.com/M/title-exact?Air+Force+One...	1997
120	121	429	Independence Day (ID4) (1996)	03-Jul-1996	NaN	http://us.imdb.com/M/title-exact?Independence%...	1996

items["title"][181]

```
'GoodFellas (1990)'
```

indexed_items = items.set_index('item_id')

indexed_items["title"][181]

```
'Return of the Jedi (1983)'
```

all_ratings = pd.merge(popularity, all_ratings)

all_ratings.describe()

	item_id	popularity	video_release_date	release_year	user_id	rating	timestamp
count	100000.000000	100000.000000	0.0	100000.000000	100000.00000	100000.000000	1.000000e+05
mean	425.530130	168.071900	NaN	1987.950100	462.48475	3.529860	8.835289e+08
std	330.798356	121.784558	NaN	14.169558	266.61442	1.125674	5.343856e+06
min	1.000000	1.000000	NaN	1920.000000	1.00000	1.000000	8.747247e+08
25%	175.000000	71.000000	NaN	1986.000000	254.00000	3.000000	8.794487e+08
50%	322.000000	145.000000	NaN	1994.000000	447.00000	4.000000	8.828269e+08
75%	631.000000	239.000000	NaN	1996.000000	682.00000	4.000000	8.882600e+08
max	1682.000000	583.000000	NaN	1998.000000	943.00000	5.000000	8.932866e+08

all_ratings.head()

	item_id	popularity	title	release_date	video_release_date	imdb_url	release_year	user_id	rating	timestamp
0	1	452	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	308	4	887736532
1	1	452	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	287	5	875334088
2	1	452	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	148	4	877019411
3	1	452	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	280	4	891700426
4	1	452	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	1995	66	3	883601324

```
from sklearn.model_selection import train_test_split

ratings_train, ratings_test = train_test_split(

    all_ratings, test_size=0.2, random_state=0)

user_id_train = np.array(ratings_train['user_id'])

item_id_train = np.array(ratings_train['item_id'])

rating_train = np.array(ratings_train['rating'])

user_id_test = np.array(ratings_test['user_id'])

item_id_test = np.array(ratings_test['item_id'])

rating_test = np.array(ratings_test['rating'])

from tensorflow.keras.layers import Embedding, Flatten, Dense, Dropout

from tensorflow.keras.layers import Dot

from tensorflow.keras.models import Model

# For each sample we input the integer identifiers

# of a single user and a single item

class RegressionModel(Model):

    def__init_(self, embedding_size, max_user_id, max_item_id):
```

```
        super().__init__()
        self.user_embedding = Embedding(output_dim=embedding_size,
                                input_dim=max_user_id + 1,
                                input_length=1,
                                name='user_embedding')
        self.item_embedding = Embedding(output_dim=embedding_size,
                                input_dim=max_item_id + 1,
                                input_length=1,
                                name='item_embedding')
        # The following two layers don't have parameters.
        self.flatten = Flatten()
        self.dot = Dot(axes=1)
    def call(self, inputs):
        user_inputs = inputs[0]
        item_inputs = inputs[1]
        user_vecs = self.flatten(self.user_embedding(user_inputs))
        item_vecs = self.flatten(self.item_embedding(item_inputs))
        y = self.dot([user_vecs, item_vecs])
        return y
model = RegressionModel(64, max_user_id, max_item_id)
model.compile(optimizer="adam", loss='mae')
# Useful for debugging the output shape of model
initial_train_preds = model.predict([user_id_train, item_id_train])
initial_train_preds.shape
# %load solutions/compute_errors.py
squared_differences = np.square(initial_train_preds[:,0] - rating_train)
```

absolute_differences = np.abs(initial_train_preds[:,0] - rating_train)

print("Random init MSE: %0.3f" % np.mean(squared_differences))

print("Random init MAE: %0.3f" % np.mean(absolute_differences))

You may also use sklearn metrics to do so using scikit-learn:

from sklearn.metrics import mean_absolute_error, mean_squared_error

print("Random init MSE: %0.3f" % mean_squared_error(initial_train_preds, rating_train))

print("Random init MAE: %0.3f" % mean_absolute_error(initial_train_preds, rating_train))

```
Random init MSE: 0.634
Random init MAE: 0.583
Random init MSE: 0.634
Random init MAE: 0.583
```

%%time

Training the model

history = model.fit([user_id_train, item_id_train], rating_train,

batch_size=64, epochs=10, validation_split=0.1,

shuffle=True)

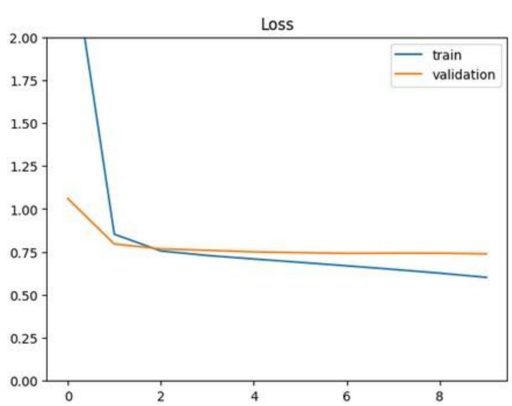
plt.plot(history.history['loss'], label='train')

plt.plot(history.history['val_loss'], label='validation')

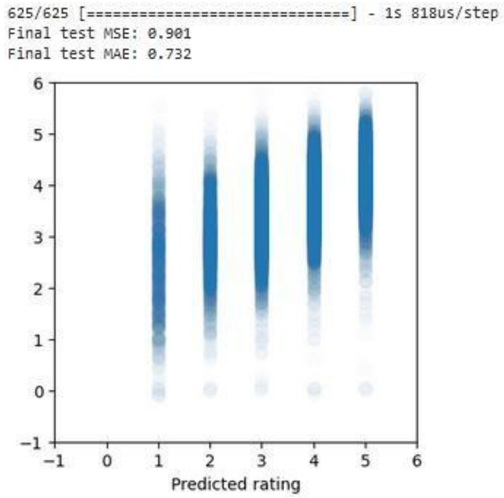
plt.ylim(0, 2)

plt.legend(loc='best')

plt.title('Loss');

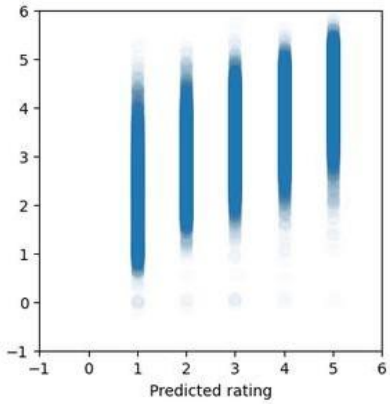


```python
def plot_predictions(y_true, y_pred):
    plt.figure(figsize=(4, 4))
    plt.xlim(-1, 6)
    plt.xlabel("True rating")
    plt.ylim(-1, 6)
    plt.xlabel("Predicted rating")
    plt.scatter(y_true, y_pred, s=60, alpha=0.01)
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
test_preds = model.predict([user_id_test, item_id_test])
print("Final test MSE: %0.3f" % mean_squared_error(test_preds, rating_test))
print("Final test MAE: %0.3f" % mean_absolute_error(test_preds, rating_test))
plot_predictions(rating_test, test_preds)
```




```python
train_preds = model.predict([user_id_train, item_id_train])
print("Final train MSE: %0.3f" % mean_squared_error(train_preds, rating_train))
print("Final train MAE: %0.3f" % mean_absolute_error(train_preds, rating_train))
plot_predictions(rating_train, train_preds)
```

```
2500/2500 [==============================] - 3s 1ms/step
Final train MSE: 0.634
Final train MAE: 0.583
```



weights and shape

weights = model.get_weights()

[w.shape for w in weights]

```
[(944, 64), (1683, 64)]
```

user_embeddings = weights[0]

item_embeddings = weights[1]

item_id = 181

print(f"Title for item_id={item_id}: {indexed_items['title'][item_id]}")

```
Title for item_id=181: Return of the Jedi (1983)
```

#Title for item_id=181: Return of the Jedi (1983)

print(f"Embedding vector for item_id={item_id}")

print(item_embeddings[item_id])

print("shape:", item_embeddings[item_id].shape)

```
Embedding vector for item_id=181
[ 0.36479577  0.21194077  0.33381793  0.46690923 -0.4615068   0.48184955
  0.4044678   0.4764477  -0.00213348  0.37826908 -0.33176094  0.3457052
 -0.29503956  0.48978835  0.07782511 -0.39882183  0.45955232 -0.14887835
  0.37896287 -0.161989    0.03519277 -0.04485312 -0.21840616 -0.03596817
 -0.33676705  0.33954418 -0.19010073 -0.50189286 -0.42458612 -0.29089347
  0.52908903  0.48058343  0.49947718  0.2880021   0.49476317 -0.25513735
  0.5599122   0.1521692   0.3346225   0.12454159  0.42250696  0.36925885
 -0.36237472  0.12584884  0.13845988 -0.24071056  0.24648649 -0.37613618
 -0.27428904  0.3505782   0.20540197  0.1128564   0.03566684 -0.37215853
  0.20527889 -0.04000176  0.23127277 -0.27025488  0.04636623 -0.30994517
 -0.4090801  -0.06630444  0.16991279 -0.09228854]
shape: (64,)
```

EPSILON = 1e-07 # to avoid division by 0.

```
def cosine(x, y):
    # TODO: implement me!
    return 0.

# %load solutions/similarity.py
EPSILON = 1e-07

def cosine(x, y):
    dot_products = np.dot(x, y.T)
    norm_products = np.linalg.norm(x) * np.linalg.norm(y)
    return dot_products / (norm_products + EPSILON)

def print_similarity(item_a, item_b, item_embeddings, titles):
    print(titles[item_a])
    print(titles[item_b])
    similarity = cosine(item_embeddings[item_a],
                        item_embeddings[item_b])
    print(f"Cosine similarity: {similarity:.3}")

print_similarity(50, 181, item_embeddings, indexed_items["title"])
```

```
Star Wars (1977)
Return of the Jedi (1983)
Cosine similarity: 0.916
```

```
print_similarity(181, 288, item_embeddings, indexed_items["title"])
```

```
Return of the Jedi (1983)
Scream (1996)
Cosine similarity: 0.717
```

```
print_similarity(181, 1, item_embeddings, indexed_items["title"])
```

```
Return of the Jedi (1983)
Toy Story (1995)
Cosine similarity: 0.809
```

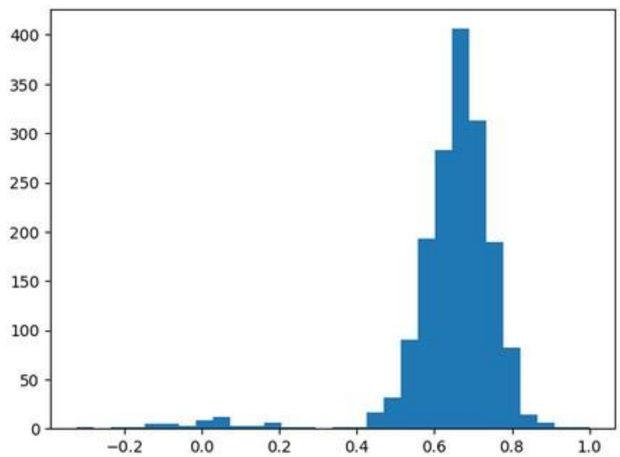
```
print_similarity(181, 181, item_embeddings, indexed_items["title"])
```

```
Return of the Jedi (1983)
Return of the Jedi (1983)
Cosine similarity: 1.0
```

```
def cosine_similarities(item_id, item_embeddings):
    """Compute similarities between item_id and all items embeddings"""
    query_vector = item_embeddings[item_id]
    dot_products = item_embeddings @ query_vector


    query_vector_norm = np.linalg.norm(query_vector)
    all_item_norms = np.linalg.norm(item_embeddings, axis=1)
    norm_products = query_vector_norm * all_item_norms
    return dot_products / (norm_products + EPSILON)
similarities = cosine_similarities(181, item_embeddings)
similarities
```

```
array([-0.20177297,  0.80906314,  0.7568024 , ...,  0.7657102 ,
        0.79105544,  0.6783905 ], dtype=float32)
```

```
plt.hist(similarities, bins=30);
```




```
def most_similar(item_id, item_embeddings, titles,
                 top_n=30):
    sims = cosine_similarities(item_id, item_embeddings)
    # [::-1] makes it possible to reverse the order of a numpy
    # array, this is required because most similar items have
```

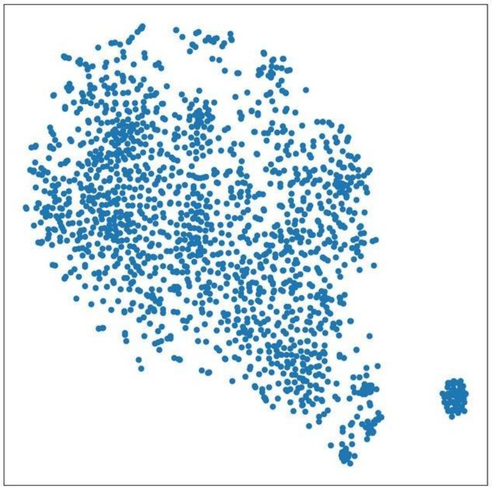
```
    # a larger cosine similarity value

    sorted_indexes = np.argsort(sims)[::-1]

    idxs = sorted_indexes[0:top_n]

    return list(zip(idxs, titles[idxs], sims[idxs]))

most_similar(50, item_embeddings, indexed_items["title"], top_n=10)
```

```
[(50, 'Star Wars (1977)', 1.0),
 (181, 'Return of the Jedi (1983)', 0.91561514),
 (172, 'Empire Strikes Back, The (1980)', 0.90764123),
 (174, 'Raiders of the Lost Ark (1981)', 0.89956915),
 (133, 'Gone with the Wind (1939)', 0.8864555),
 (404, 'Pinocchio (1940)', 0.8735312),
 (527, 'Gandhi (1982)', 0.8685317),
 (204, 'Back to the Future (1985)', 0.86583817),
 (210, 'Indiana Jones and the Last Crusade (1989)', 0.8658317),
 (8, 'Babe (1995)', 0.86524856)]
```

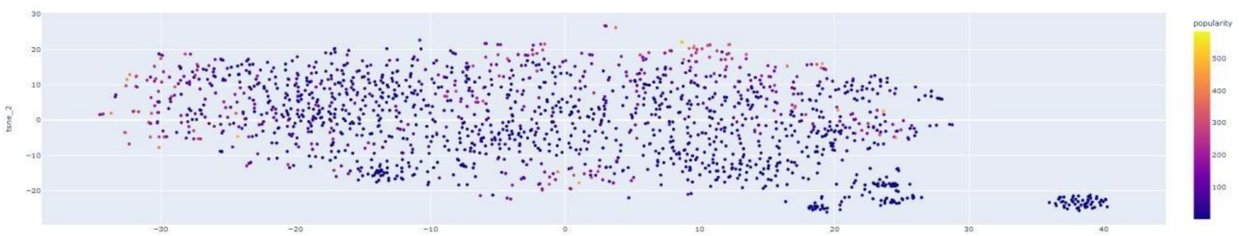
```
# items[items['title'].str.contains("Star Trek")]

most_similar(227, item_embeddings, indexed_items["title"], top_n=10)
```

```
[(227, 'Star Trek VI: The Undiscovered Country (1991)', 0.99999994),
 (230, 'Star Trek IV: The Voyage Home (1986)', 0.91736984),
 (1321, 'Open Season (1996)', 0.91401035),
 (1492, 'Window to Paris (1994)', 0.90971726),
 (228, 'Star Trek: The Wrath of Khan (1982)', 0.9030785),
 (1218, 'Friday (1995)', 0.898074),
 (82, 'Jurassic Park (1993)', 0.89470667),
 (1138, 'Best Men (1997)', 0.89300644),
 (1498, 'Farmer & Chase (1995)', 0.8898689),
 (578, 'Demolition Man (1993)', 0.88710773)]
```

```
from sklearn.manifold import TSNE

item_tsne = TSNE(perplexity=30).fit_transform(item_embeddings)

import matplotlib.pyplot as plt

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

plt.scatter(item_tsne[:, 0], item_tsne[:, 1]);

plt.xticks(()); plt.yticks(());

plt.show()
```



```
%pip install -q plotly

import plotly.express as px

tsne_df = pd.DataFrame(item_tsne, columns=["tsne_1", "tsne_2"])

tsne_df["item_id"] = np.arange(item_tsne.shape[0])

tsne_df = tsne_df.merge(items.reset_index())

px.scatter(tsne_df, x="tsne_1", y="tsne_2",

        color="popularity",

        hover_data=["item_id", "title",

                "release_year", "popularity"])
```



PRACTICAL NO.: - 6

AIM : Backpropagation in Neural Networks using Numpy.

INPUT

```
import numpy as np

# Define the sigmoid activation function and its derivative
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Define the neural network architecture
input_size = 2
hidden_size = 3
output_size = 1
learning_rate = 0.1

# Initialize weights and biases
np.random.seed(0)
input_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
output_data = np.array([[0], [1], [1], [0]])

# Weights and biases initialization
weights_input_hidden = np.random.uniform(size=(input_size, hidden_size))
biases_hidden = np.zeros((1, hidden_size))
weights_hidden_output = np.random.uniform(size=(hidden_size, output_size))
biases_output = np.zeros((1, output_size))

# Training loop
epochs = 10000
for epoch in range(epochs):
    # Forward pass
    hidden_layer_input = np.dot(input_data, weights_input_hidden) + biases_hidden
    hidden_layer_output = sigmoid(hidden_layer_input)
    output_layer_input = np.dot(hidden_layer_output, weights_hidden_output) +
biases_output
    predicted_output = sigmoid(output_layer_input)

    # Compute the loss
    loss = 0.5 * np.mean((predicted_output - output_data) ** 2)

    # Backpropagation
    output_error = output_data - predicted_output
    output_delta = output_error * sigmoid_derivative(predicted_output)
```

```
    hidden_layer_error = output_delta.dot(weights_hidden_output.T)
    hidden_layer_delta = hidden_layer_error * sigmoid_derivative(hidden_layer_output)

    # Update weights and biases
    weights_hidden_output += hidden_layer_output.T.dot(output_delta) * learning_rate
    biases_output += np.sum(output_delta, axis=0, keepdims=True) * learning_rate
    weights_input_hidden += input_data.T.dot(hidden_layer_delta) * learning_rate
    biases_hidden += np.sum(hidden_layer_delta, axis=0, keepdims=True) *
learning_rate

    if epoch % 1000 == 0:
        print(f"Epoch {epoch}, Loss: {loss}")

print("Training completed.")
```

Output

```
Epoch 0, Loss: 0.1714629896398691
Epoch 1000, Loss: 0.12310016740452781
Epoch 2000, Loss: 0.11202152064381576
Epoch 3000, Loss: 0.08233179820965757
Epoch 4000, Loss: 0.02650020295068723
Epoch 5000, Loss: 0.008485970497104004
Epoch 6000, Loss: 0.004449333904478412
Epoch 7000, Loss: 0.002906299373724993
Epoch 8000, Loss: 0.002124174486298295
Epoch 9000, Loss: 0.0016598326873063268
Training completed.
```

PRACTICAL NO.: - 7

AIM : Neural Recommender Systems with Implicit Feedback and the Triplet Loss

INPUT

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split

# Load your dataset or create a toy dataset
# For this example, we'll use a random toy dataset
num_users = 100
num_items = 50
embedding_dim = 50

# Generate toy implicit feedback data
np.random.seed(0)
user_ids = np.random.randint(0, num_users, 1000)
positive_items = np.random.randint(0, num_items, 1000)

# Create triplets (user, positive_item, negative_item)
def create_triplets(user_ids, positive_items, num_items):
    triplets = []
    for user, positive_item in zip(user_ids, positive_items):
        negative_item = np.random.randint(0, num_items)
        while negative_item == positive_item:
            negative_item = np.random.randint(0, num_items)
        triplets.append([user, positive_item, negative_item])
    return np.array(triplets)

triplets = create_triplets(user_ids, positive_items, num_items)

# Split the data into training and validation sets
train_triplets, val_triplets = train_test_split(triplets, test_size=0.1)

# Define the neural network model
user_input = keras.Input(shape=(1,))
positive_item_input = keras.Input(shape=(1,))
negative_item_input = keras.Input(shape=(1,))

embedding_layer = layers.Embedding(num_users, embedding_dim, input_length=1)
user_embedding = embedding_layer(user_input)
positive_item_embedding = embedding_layer(positive_item_input)
negative_item_embedding = embedding_layer(negative_item_input)

# Define the triplet loss layer as a custom layer
```

```
class TripletLossLayer(layers.Layer):
    def __init__(self, margin=0.2, **kwargs):
        super(TripletLossLayer, self).__init__(**kwargs)
        self.margin = margin

    def call(self, inputs):
        user_embedding, positive_item_embedding, negative_item_embedding = inputs
        positive_distance = tf.reduce_sum(tf.square(user_embedding -
positive_item_embedding), axis=1)
        negative_distance = tf.reduce_sum(tf.square(user_embedding -
negative_item_embedding), axis=1)
        loss = tf.maximum(0.0, positive_distance - negative_distance + self.margin)
        return loss

triplet_loss_layer = TripletLossLayer()([user_embedding, positive_item_embedding,
negative_item_embedding])

model = keras.Model(inputs=[user_input, positive_item_input, negative_item_input],
outputs=triplet_loss_layer)

# Compile the model
model.compile(optimizer="adam", loss="mean_absolute_error")

# Training
batch_size = 64
num_epochs = 10

model.fit(
    [train_triplets[:, 0], train_triplets[:, 1], train_triplets[:, 2]],
    np.zeros(len(train_triplets)),
    batch_size=batch_size,
    epochs=num_epochs,
    validation_data=(
        [val_triplets[:, 0], val_triplets[:, 1], val_triplets[:, 2]],
        np.zeros(len(val_triplets)),
    ),
)
```

Output

```
Epoch 1/10
15/15 [==============================] - 1s 24ms/step - loss: 0.2000 -
val_loss: 0.2000
Epoch 2/10
15/15 [==============================] - 0s 9ms/step - loss: 0.1998 -
val_loss: 0.2000
Epoch 3/10
15/15 [==============================] - 0s 8ms/step - loss: 0.1997 -
val_loss: 0.2000
Epoch 4/10
15/15 [==============================] - 0s 5ms/step - loss: 0.1995 -
val_loss: 0.2000
Epoch 5/10
15/15 [==============================] - 0s 4ms/step - loss: 0.1993 -
val_loss: 0.2000
Epoch 6/10
15/15 [==============================] - 0s 5ms/step - loss: 0.1991 -
val_loss: 0.2000
Epoch 7/10
15/15 [==============================] - 0s 5ms/step - loss: 0.1988 -
val_loss: 0.2000
Epoch 8/10
15/15 [==============================] - 0s 5ms/step - loss: 0.1985 -
val_loss: 0.2000
Epoch 9/10
15/15 [==============================] - 0s 4ms/step - loss: 0.1981 -
val_loss: 0.2000
Epoch 10/10
15/15 [==============================] - 0s 5ms/step - loss: 0.1977 -
val_loss: 0.2000
<keras.src.callbacks.History at 0x7eb78a2e9540>
```

PRACTICAL NO.: - 8

AIM : Fully Convolutional Neural Networks.

INPUT

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define a basic Fully Convolutional Neural Network
def create_fully_convolutional_network(input_shape, num_classes):
    model = keras.Sequential()

    # Encoder
    model.add(layers.Input(shape=input_shape))
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Middle
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))

    # Decoder
    model.add(layers.UpSampling2D((2, 2)))
    model.add(layers.Conv2D(num_classes, (1, 1), activation='softmax', padding='valid'))

    return model

# Define input shape and number of classes
input_shape = (256, 256, 3)  # Input image dimensions (e.g., 256x256 RGB image)
num_classes = 21  # Number of segmentation classes

# Create the FCN model
fcn_model = create_fully_convolutional_network(input_shape, num_classes)

# Compile the model with an appropriate loss and optimizer
fcn_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Summary of the model architecture
fcn_model.summary()
```

Output

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 64)      1792

 max_pooling2d (MaxPooling2  (None, 128, 128, 64)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 128)     73856

 up_sampling2d (UpSampling2  (None, 256, 256, 128)     0
 D)

 conv2d_2 (Conv2D)           (None, 256, 256, 21)      2709

=================================================================
Total params: 78357 (306.08 KB)
Trainable params: 78357 (306.08 KB)
Non-trainable params: 0 (0.00 Byte)
```

PRACTICAL NO.: - 9

AIM : ConvNets for Classification and Localization.

INPUT

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Define a Localization CNN model for classification and localization
def create_localization_cnn(input_shape, num_classes, num_coords):
    input_tensor = layers.Input(shape=input_shape)

    # Convolutional layers for feature extraction
    x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(input_tensor)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)
    x = layers.Conv2D(256, (3, 3), activation='relu', padding='same')(x)
    x = layers.MaxPooling2D((2, 2))(x)

    # Flatten the feature map for classification
    flat = layers.Flatten()(x)

    # Classification head
    classification = layers.Dense(num_classes, activation='softmax',
name='classification')(flat)
    # Localization head
    localization = layers.Dense(num_coords, activation='linear', name='localization')(flat)

    return keras.Model(inputs=input_tensor, outputs=[classification, localization])

# Define input shape, number of classes, and number of coordinates (e.g., x, y)
input_shape = (224, 224, 3)  # Input image dimensions (e.g., 224x224 RGB image)
num_classes = 10  # Number of classes for classification
num_coords = 4  # Number of coordinates (e.g., x, y, width, height) for localization

# Create the Localization CNN model
localization_cnn = create_localization_cnn(input_shape, num_classes, num_coords)
# Compile the model with appropriate loss functions and optimizers
localization_cnn.compile(
    optimizer='adam',
    loss={'classification': 'categorical_crossentropy', 'localization': 'mean_squared_error'},
    loss_weights={'classification': 1.0, 'localization': 1.0} )

# Summary of the model architecture
localization_cnn.summary()
```

Output

```
Model: "model_2"
________________________________________________________________________________________________________________________
 Layer (type)              Output Shape              Param #   Connected to
========================================================================================================================
 input_35 (InputLayer)      [(None, 224, 224, 3)]      0        []

 conv2d_3 (Conv2D)          (None, 224, 224, 64)       1792      ['input_35[0][0]']

 max_pooling2d_1 (MaxPoolin  (None, 112, 112, 64)       0        ['conv2d_3[0][0]']
 g2D)

 conv2d_4 (Conv2D)          (None, 112, 112, 128)       73856
['max_pooling2d_1[0][0]']

 max_pooling2d_2 (MaxPoolin  (None, 56, 56, 128)        0        ['conv2d_4[0][0]']
 g2D)

 conv2d_5 (Conv2D)          (None, 56, 56, 256)        295168
['max_pooling2d_2[0][0]']

 max_pooling2d_3 (MaxPoolin  (None, 28, 28, 256)        0        ['conv2d_5[0][0]']
 g2D)

 flatten (Flatten)          (None, 200704)             0        ['max_pooling2d_3[0][0]']

 classification (Dense)      (None, 10)                2007050   ['flatten[0][0]']

 localization (Dense)        (None, 4)                 802820    ['flatten[0][0]']

========================================================================================================================
Total params: 3180686 (12.13 MB)
Trainable params: 3180686 (12.13 MB)
Non-trainable params: 0 (0.00 Byte)
```

PRACTICAL NO.: - 10

AIM : Text Classification and Word Vectors.

INPUT

```
import numpy as np
from gensim.models import Word2Vec
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Step 1: Sample data (replace with your dataset)
positive_reviews = ["This movie is fantastic!", "I loved it.", "Great film."]
negative_reviews = ["Terrible movie.", "Hated it.", "Awful film."]

# Label the data: 1 for positive and 0 for negative
labels = [1] * len(positive_reviews) + [0] * len(negative_reviews)
reviews = positive_reviews + negative_reviews

# Step 2: Load pre-trained Word2Vec embeddings
# Download pre-trained Word2Vec embeddings from a source like GloVe or Word2Vec
# In this example, we'll use dummy Word2Vec embeddings for illustration purposes.

word_vectors = {
    "this": np.array([0.1, 0.2, 0.3]),
    "movie": np.array([0.2, 0.3, 0.4]),
    "is": np.array([0.3, 0.4, 0.5]),
    "fantastic!": np.array([0.4, 0.5, 0.6]),
    "terrible": np.array([-0.3, -0.4, -0.5]),
    "hated": np.array([-0.4, -0.5, -0.6]),
}

# Step 3: Convert text data to numerical representations using word vectors
def text_to_vector(text):
    words = text.split()
    vectors = [word_vectors[word] for word in words if word in word_vectors]
    if not vectors:
        return np.zeros(3)  # Return a zero vector if no known words are present
    return np.mean(vectors, axis=0)

X = np.array([text_to_vector(review) for review in reviews])

# Step 4: Split the data and train a classification model (e.g., Naive Bayes)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.2,
random_state=42)

# Train a simple Naive Bayes classifier
```

```
classifier = MultinomialNB()
classifier.fit(X_train, y_train)

# Make predictions
y_pred = classifier.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Output

```
Accuracy: 0.0
```

PRACTICAL NO.: - 11

AIM: Character Level Language Model (GPU required).

INPUT

```
import torch
import torch.nn as nn
import torch.optim as optim

# Define the data
text = "Your training data goes here..."

# Create a character-level vocabulary
vocab = set(text)
vocab_size = len(vocab)
char_to_idx = {char: idx for idx, char in enumerate(vocab)}
idx_to_char = {idx: char for char, idx in char_to_idx.items()}

# Hyperparameters
hidden_size = 100
num_layers = 2
learning_rate = 0.01
num_epochs = 100

# Define the model
class CharRNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super(CharRNN, self).__init__()  # Corrected super call
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.LSTM(hidden_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, input_size)

    def forward(self, x, hidden):
        out = self.embedding(x)
        out, hidden = self.rnn(out, hidden)
        out = self.fc(out)
        return out, hidden

model = CharRNN(vocab_size, hidden_size, num_layers)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training loop
for epoch in range(num_epochs):
    hidden = (torch.zeros(num_layers, 1, hidden_size), torch.zeros(num_layers, 1,
hidden_size))
    total_loss = 0
```

```
    for i in range(0, len(text) - 100, 100):
        input_seq = text[i:i+100]
        target_seq = text[i+1:i+101]

        input_tensor = torch.tensor([char_to_idx[c] for c in input_seq], dtype=torch.long)
        target_tensor = torch.tensor([char_to_idx[c] for c in target_seq], dtype=torch.long)

        output, hidden = model(input_tensor.view(1, -1), hidden)
        loss = criterion(output.view(1, -1, vocab_size), target_tensor.view(1, -1))

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    if (epoch + 1) % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {total_loss:.4f}')

# Generating text
with torch.no_grad():
    seed_text = "Your seed text..."
    generated_text = seed_text
    hidden = torch.zeros(num_layers, 1, hidden_size), torch.zeros(num_layers, 1,
hidden_size)

    for char in seed_text:
        if char in char_to_idx:
            input_char = torch.tensor(char_to_idx[char], dtype=torch.long)
            output, hidden = model(input_char.view(1, -1), hidden)
        else:
            # Handle characters not in the vocabulary, such as whitespace or special
characters.
            # You can choose to skip or replace them as needed.
            continue

    for _ in range(1000):
        output_softmax = torch.softmax(output.view(-1), dim=0)
        predicted_idx = torch.multinomial(output_softmax, 1)
        predicted_char = idx_to_char[predicted_idx.item()]
        generated_text += predicted_char
        input_char = predicted_idx
        output, hidden = model(input_char.view(1, -1), hidden)

    print(generated_text)
```

Output

```
Epoch [10/100], Loss: 0.0000
Epoch [20/100], Loss: 0.0000
Epoch [30/100], Loss: 0.0000
Epoch [40/100], Loss: 0.0000
Epoch [50/100], Loss: 0.0000
Epoch [60/100], Loss: 0.0000
Epoch [70/100], Loss: 0.0000
Epoch [80/100], Loss: 0.0000
Epoch [90/100], Loss: 0.0000
Epoch [100/100], Loss: 0.0000
Your seed text... Yed iaeYegea s.tss.un uson.hhgitneti.gou tssa.u gidoi aaed
nadutahYY.ingtsui.ditYase.sentggnnag .groeYu.YheeiYrasgnitengau. au.egsrite
tetnnYo.ehr.asrYYnYhYe.ohrisgheae.ihYiidsnYsti r
drhaians.uouYdssgtrY.dr.isggeodsauYud.tnhteea edYeuduYin.Y uggYdhoar
ndsttitie.odtstaYnsrnarsudunohhh hYnYi Ygiirnru.ni gthaon.o.rne  g tgnnr r
eiairugenhtrn ttadsrtYtgdesoguseushhoe ha.sgnutndYgdu r asYaoaoss g ..Yinn idi ha
.d.tnus .idsei.. sa oaYe.dr i.eds.nu. gauhse
aenniohooeursururreaoghiueoih.uguaiginuedatgtdd.ennnid sei.rdnd seY
rr.ugns.suda.tishd doru rianhah.dggs.do.iirsneonaYg oshsrtYsndunuusggdd.gtorh
rYnedttti.aYY.eh.eoiYsgngeYgahensuhhotgidt.uetdtsstneusotdnuaeanYasao
tgneehihtYd. .ieshnYdthtin.nshnrdth
.nhdoinghhttuu.dsuhtggoguiud.ogoYgagd...egtsioig.Yngutre hds gdrnYs dgoano
dhuge..n...otnYtghegeeooda.uhr. rdnogahrouho.a...tt.shg. iisassireurrsagadiou. snaeot
dt n diegg.naasa usihhton .oh ds adrrushn.nderndiYggYe
oudghnYrgrhdgositnYusYtYsunddosnhdsgguun urho..hasagsrearootnegounaon nsniet
```