Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :......................   Date of Completion:.......................

Group B Deep Learning

Assignment No: 1

Title of the Assignment: Linear regression by using Deep Neural network: Implement Boston housing price.prediction problem by Linear regression using Deep Neural network. Use Boston House price prediction dataset.

Objective of the Assignment: Students should be able to perform Linear regression by using Deep Neural network on Boston House Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Linear Regression
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Linear Regression
2. Example of Linear Regression
3. Concept of Deep Neural Network
4. How Deep Neural Network Work
5. Code Explanation with Output

What is Linear Regression?

Linear regression is a statistical approach that is commonly used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the variables and uses mathematical methods to estimate the coefficients that best fit the data.

Deep neural networks are a type of machine learning algorithm that are modeled after the structure and function of the human brain. They consist of multiple layers of interconnected neurons that process data and learn from it to make predictions or classifications.

Linear regression using deep neural networks combines the principles of linear regression with the power of deep learning algorithms. In this approach, the input features are passed through one or more layers of neurons to extract features and then a linear regression model is applied to the output of the last layer to make predictions. The weights and biases of the neural network are adjusted during training to optimize the performance of the model.

This approach can be used for a variety of tasks, including predicting numerical values, such as stock prices or housing prices, and classifying data into categories, such as detecting whether an image contains a particular object or not. It is often used in fields such as finance, healthcare, and image recognition.

Example Of Linear Regression

A suitable example of linear regression using deep neural network would be predicting the price of a house based on various features such as the size of the house, the number of bedrooms, the location, and the age of the house.

In this example, the input features would be fed into a deep neural network, consisting of multiple layers of interconnected neurons. The first few layers of the network would learn to extract features from the input data, such as identifying patterns and correlations between the input features.

The output of the last layer would then be passed through a linear regression model, which would use the learned features to predict the price of the house.

During training, the weights and biases of the neural network would be adjusted to minimize the difference between the predicted price and the actual price of the house. This process is known as gradient descent, and it involves iteratively adjusting the model's parameters until the optimal values are reached.

Once the model is trained, it can be used to predict the price of a new house based on its features. This approach can be used in the real estate industry to provide accurate and reliable estimates of house prices, which can help both buyers and sellers make informed decisions.

Concept of Deep Neural Network-

A deep neural network is a type of machine learning algorithm that is modeled after the structure and function of the human brain. It consists of multiple layers of interconnected nodes, or artificial neurons, that process data and learn from it to make predictions or classifications.

Each layer of the network performs a specific type of processing on the data, such as identifying patterns or correlations between features, and passes the results to the next layer. The layers closest to the input are known as the "input layer", while the layers closest to the output are known as the "output layer".

The intermediate layers between the input and output layers are known as "hidden layers". These layers are responsible for extracting increasingly complex features from the input data, and can be deep (i.e., containing many hidden layers) or shallow (i.e., containing only a few hidden layers).

Deep neural networks are trained using a process known as backpropagation, which involves adjusting the weights and biases of the nodes based on the error between the predicted output and the actual output. This process is repeated for multiple iterations until the model reaches an optimal level of accuracy.

Deep neural networks are used in a variety of applications, such as image and speech recognition, natural language processing, and recommendation systems. They are capable of learning from vast amounts of data and can automatically extract features from raw data, making them a powerful tool for solving complex problems in a wide range of domains.

How Deep Neural Network Work-

Boston House Price Prediction is a common example used to illustrate how a deep neural network can work for regression tasks. The goal of this task is to predict the price of a house in Boston based on various features such as the number of rooms, crime rate, and accessibility to public transportation.

Here's how a deep neural network can work for Boston House Price Prediction:

1. **Data preprocessing:** The first step is to preprocess the data. This involves normalizing the input features to have a mean of 0 and a standard deviation of 1, which helps the network learn more efficiently. The dataset is then split into training and testing sets.
2. **Model architecture:** A deep neural network is then defined with multiple layers. The first layer is the input layer, which takes in the normalized features. This is followed by several hidden layers, which can be deep or shallow. The last layer is the output layer, which predicts the house price.
3. **Model training:** The model is then trained using the training set. During training, the weights and biases of the nodes are adjusted based on the error between the predicted output and the actual output. This is done using an optimization algorithm such as stochastic gradient descent.
4. **Model evaluation:** Once the model is trained, it is evaluated using the testing set. The

performance of the model is measured using metrics such as mean squared error or mean absolute error.

5. **Model prediction:** Finally, the trained model can be used to make predictions on new data, such as predicting the price of a new house in Boston based on its features.
6. By using a deep neural network for Boston House Price Prediction, we can obtain accurate predictions based on a large set of input features. This approach is scalable and can be used for other regression tasks as well.

Boston House Price Prediction Dataset-

Boston House Price Prediction is a well-known dataset in machine learning and is often used to demonstrate regression analysis techniques. The dataset contains information about 506 houses in Boston, Massachusetts, USA. The goal is to predict the median value of owner-occupied homes in thousands of dollars.

The dataset includes 13 input features, which are:

CRIM: per capita crime rate by town

ZN: proportion of residential land zoned for lots over 25,000 sq.ft.

INDUS: proportion of non-retail business acres per town

CHAS: Charles River dummy variable (1 if tract bounds river; 0 otherwise)

NOX: nitric oxides concentration (parts per 10 million)

RM: average number of rooms per dwelling

AGE: proportion of owner-occupied units built prior to 1940

DIS: weighted distances to five Boston employment centers

RAD: index of accessibility to radial highways

TAX: full-value property-tax rate per $10,000

PTRATIO: pupil-teacher ratio by town

B: 1000(Bk - 0.63)^2 where Bk is the proportion of black people by town

LSTAT: % lower status of the population

The output variable is the median value of owner-occupied homes in thousands of dollars (MEDV).

To predict the median value of owner-occupied homes, a regression model is trained on the dataset. The model can be a simple linear regression model or a more complex model, such as a deep neural network.

After the model is trained, it can be used to predict the median value of owner-occupied homes based on the input features. The model's accuracy can be evaluated using metrics such as mean squared error or mean absolute error.

Boston House Price Prediction is a example of regression analysis and is often used to teach machine learning concepts. The dataset is also used in research to compare the performance of different regression models.

Source Code with Explanation-

```
#Importing the pandas for data processing and numpy for numerical computing
import numpy as np
import pandas as pd
# Importing the Boston Housing dataset from the sklearn
from sklearn.datasets import load_boston
boston = load_boston()
#Converting the data into pandas dataframe
data = pd.DataFrame(boston.data)
#First look at the data
data.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

```
#Adding the feature names to the dataframe
data.columns = boston.feature_names
#Adding the target variable to the dataset
data['PRICE'] = boston.target
#Looking at the data with names and target variable
data.head(n=10)
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2
5	0.02985	0.0	2.18	0.0	0.458	6.430	58.7	6.0622	3.0	222.0	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0.0	0.524	6.012	66.6	5.5605	5.0	311.0	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0.0	0.524	6.172	96.1	5.9505	5.0	311.0	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0.0	0.524	5.631	100.0	6.0821	5.0	311.0	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0.0	0.524	6.004	85.9	6.5921	5.0	311.0	15.2	386.71	17.10	18.9

```
#Shape of the data
print(data.shape)
#Checking the null values in the dataset
data.isnull().sum()
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
PRICE      0
dtype: int64
#Checking the statistics of the data
data.describe()
# This is sometimes very useful, for example if you look at the CRIM the max is 
88.97 and 75% of the value is below 3.677083 and
# mean is 3.613524 so it means the max values is actually an outlier or there are
```

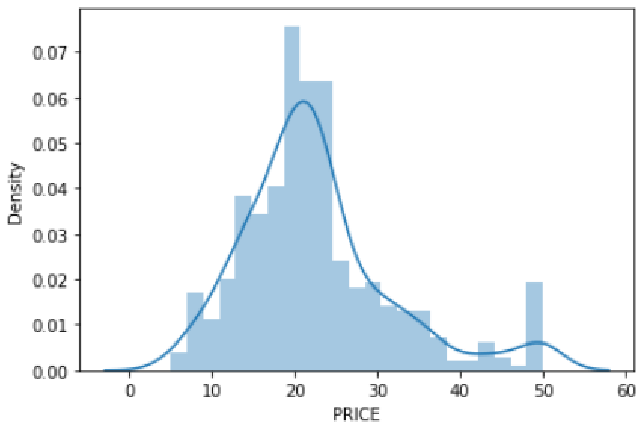
```
outliers present in the column
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  PRICE    506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

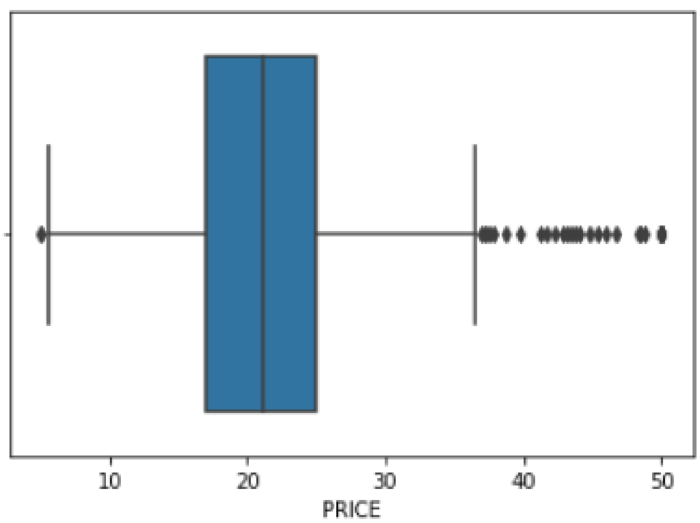
```
#checking the distribution of the target variable
import seaborn as sns
sns.distplot(data.PRICE)
#The distribution seems normal, has not be the data normal we would have perform
log transformation or took to square root of the data to make the data normal.
# Normal distribution is need for the machine learning for better predictiblity
of the model
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44d082c670>
```



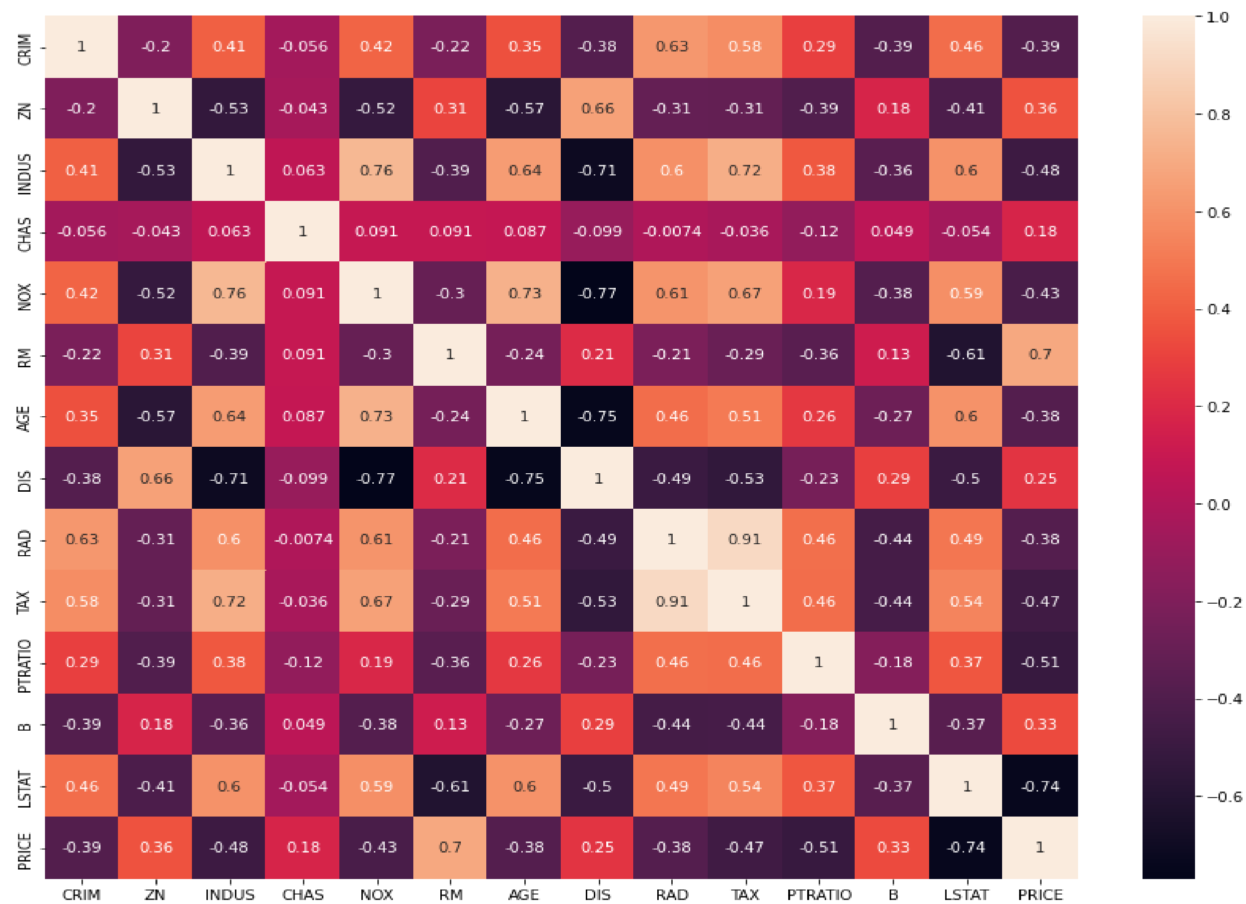

```
#Distribution using box plot
sns.boxplot(data.PRICE)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f44d077ed60>
```

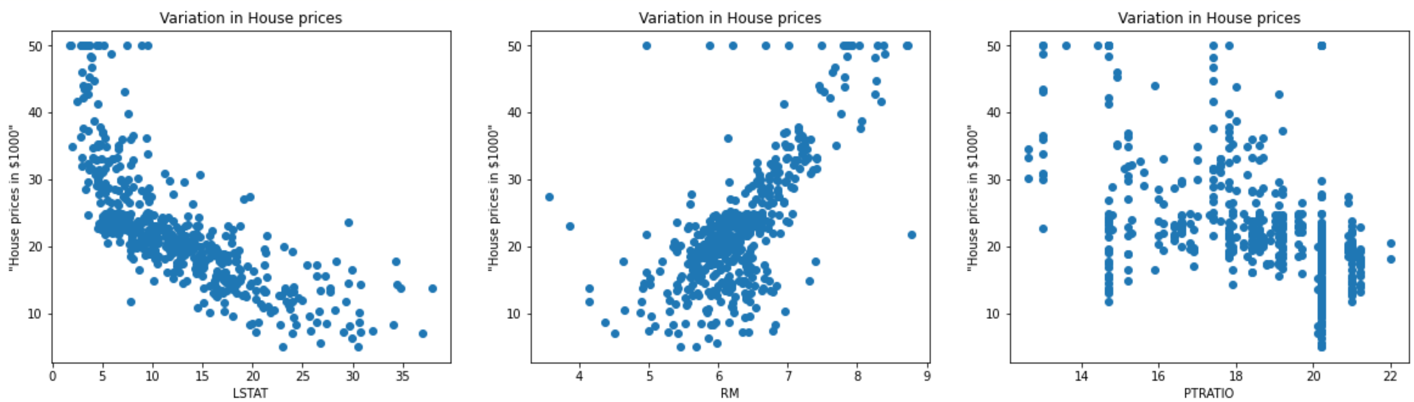



```
#Checking the correlation of the independent feature with the dependent feature
# Correlation is a statistical technique that can show whether and how strongly
pairs of variables are related.An intelligent correlation analysis can lead to a
greater understanding of your data
#checking Correlation of the data
correlation = data.corr()
correlation.loc['PRICE']
CRIM       -0.388305
```

```
ZN          0.360445
INDUS      -0.483725
CHAS        0.175260
NOX        -0.427321
RM          0.695360
AGE        -0.376955
DIS         0.249929
RAD        -0.381626
TAX        -0.468536
PTRATIO    -0.507787
B           0.333461
LSTAT      -0.737663
PRICE       1.000000
Name: PRICE, dtype: float64
# plotting the heatmap
import matplotlib.pyplot as plt
fig,axes = plt.subplots(figsize=(15,12))
sns.heatmap(correlation,square = True,annot = True)
# By looking at the correlation plot LSAT is negatively correlated with -0.75 and
RM is positively correlated to the price and PTRATIO is correlated negatively
with -0.51
```



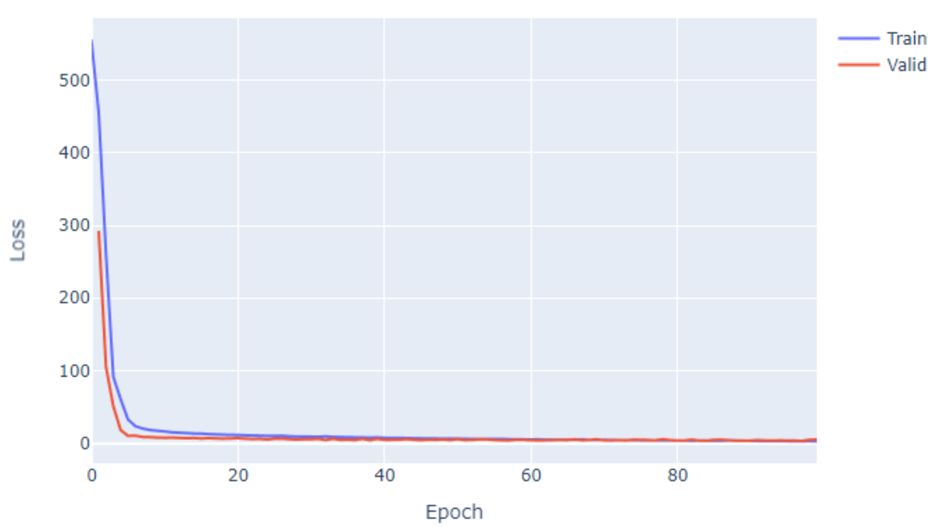
```
# Checking the scatter plot with the most correlated features
plt.figure(figsize = (20,5))
features = ['LSTAT','RM','PTRATIO']
for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = data[col]
    y = data.PRICE
    plt.scatter(x, y, marker='o')
    plt.title("Variation in House prices")
    plt.xlabel(col)
    plt.ylabel('"House prices in $1000"')
```




```
# Splitting the dependent feature and independent feature
#X = data[['LSTAT','RM','PTRATIO']]
X = data.iloc[:,:-1]
y= data.PRICE

# In order to provide a standardized input to our neural network, we need the
perform the normalization of our dataset.
# This can be seen as an step to reduce the differences in scale that may arise
from the existent features.
# We perform this normalization by subtracting the mean from our data and
dividing it by the standard deviation.
# One more time, this normalization should only be performed by using the mean
and standard deviation from the training set,
# in order to avoid any information leak from the test set.

mean = X_train.mean(axis=0)
std = X_train.std(axis=0)

X_train = (X_train - mean) / std
X_test = (X_test - mean) / std
#Linear Regression
from sklearn.linear_model import LinearRegression
```

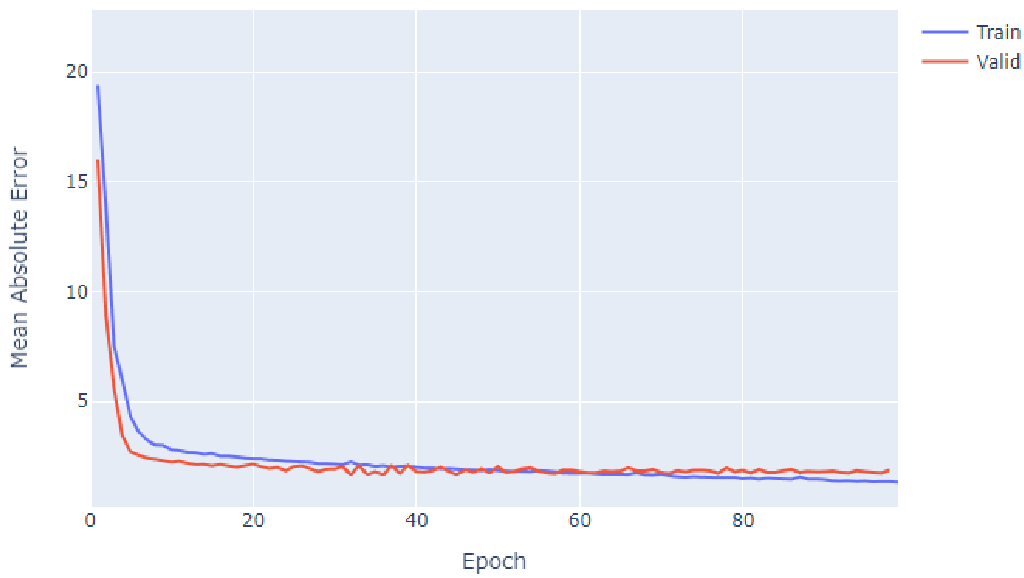
```
regressor = LinearRegression()
#Fitting the model
regressor.fit(X_train,y_train)
# Model Evaluation

#Prediction on the test dataset
y_pred = regressor.predict(X_test)
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)

from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
# Neural Networks
#Scaling the dataset
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
# Due to the small amount of presented data in this dataset, we must be careful
to not create an overly complex model,
# which could lead to overfitting our data. For this, we are going to adopt an
architecture based on two Dense layers,
# the first with 128 and the second with 64 neurons, both using a ReLU activation
function.
# A dense layer with a linear activation will be used as output layer.
# In order to allow us to know if our model is properly learning, we will use a
mean squared error loss function and to report the performance of it we will
adopt the mean average error metric.
# By using the summary method from Keras, we can see that we have a total of
10,113 parameters, which is acceptable for us.

#Creating the neural network model
import keras
from keras.layers import Dense, Activation,Dropout
from keras.models import Sequential

model = Sequential()
model.add(Dense(128,activation  = 'relu',input_dim =13))
model.add(Dense(64,activation  = 'relu'))
```

```
model.add(Dense(32,activation  = 'relu'))
model.add(Dense(16,activation  = 'relu'))
model.add(Dense(1))
#model.compile(optimizer='adam', loss='mse', metrics=['mae'])
model.compile(optimizer = 'adam',loss ='mean_squared_error',metrics=['mae'])
!pip install ann_visualizer
!pip install graphviz
from ann_visualizer.visualize import ann_viz;
#Build your model here
ann_viz(model, title="DEMO ANN");
history = model.fit(X_train, y_train, epochs=100, validation_split=0.05)
# By plotting both loss and mean average error, we can see that our model was
capable of learning patterns in our data without overfitting taking place (as
shown by the validation set curves)
from plotly.subplots import make_subplots
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Scattergl(y=history.history['loss'],
                    name='Train'))

fig.add_trace(go.Scattergl(y=history.history['val_loss'],
                    name='Valid'))


fig.update_layout(height=500, width=700,
                  xaxis_title='Epoch',
                  yaxis_title='Loss')
fig.show()
```




```
fig = go.Figure()
```

```
fig.add_trace(go.Scattergl(y=history.history['mae'],
                    name='Train'))

fig.add_trace(go.Scattergl(y=history.history['val_mae'],
                    name='Valid'))


fig.update_layout(height=500, width=700,
                  xaxis_title='Epoch',
                  yaxis_title='Mean Absolute Error')
fig.show()
```




```
#Evaluation of the model
y_pred = model.predict(X_test)
mse_nn, mae_nn = model.evaluate(X_test, y_test)

print('Mean squared error on test data: ', mse_nn)
print('Mean absolute error on test data: ', mae_nn)
```

4/4 [==============================] - 0s 4ms/step - loss: 10.5717 - mae: 2.2670

Mean squared error on test data: 10.571733474731445

Mean absolute error on test data: 2.2669904232025146

```
#Comparison with traditional approaches
#First let's try with a simple algorithm, the Linear Regression:
from sklearn.metrics import mean_absolute_error

lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
```

```
y_pred_lr = lr_model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)

print('Mean squared error on test data: ', mse_lr)
print('Mean absolute error on test data: ', mae_lr)
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print(r2)
0.8812832788381159
```

```
# Predicting RMSE the Test set results
from sklearn.metrics import mean_squared_error
rmse = (np.sqrt(mean_squared_error(y_test, y_pred)))
print(rmse)
3.320768607496587
# Make predictions on new data
import sklearn
new_data = sklearn.preprocessing.StandardScaler().fit_transform(([[0.1, 10.0,
5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]]))
prediction = model.predict(new_data)
print("Predicted house price:", prediction)
1/1 [==============================] - 0s 70ms/step
Predicted house price: [[11.104753]]
```

#new_data = sklearn.preprocessing.StandardScaler().fit_transform(([[0.1, 10.0, 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]])) is a line of code that standardizes the input features of a new data point.

In this specific case, we have a new data point represented as a list of 13 numeric values ([0.1, 10.0, 5.0, 0, 0.4, 6.0, 50, 6.0, 1, 400, 20, 300, 10]) that represents the values for the 13 features of the Boston House Price dataset.

The StandardScaler() function from the sklearn.preprocessing module is used to standardize the data. Standardization scales each feature to have zero mean and unit variance, which is a common preprocessing step in machine learning to ensure that all features contribute equally to the model.

The fit_transform() method is used to fit the scaler to the data and apply the standardization transformation. The result is a new data

```
point with standardized feature values.
```

Conclusion- In this way we can Predict the Boston House Price using Deep Neural Network.

Assignment Question

1. What is Linear Regression?
2. What is a Deep Neural Network?
3. What is the concept of standardization?
4. Why split data into train and test?
5. Write Down Application of Deep Neural Network?

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group B Deep Learning

Assignment No: 2A

Title of the Assignment: Binary classification using Deep Neural Networks Example: Classify movie reviews into positive" reviews and "negative" reviews, just based on the text content of the reviews.Use IMDB dataset

Objective of the Assignment: Students should be able to Classify movie reviews into positive reviews and "negative reviews on IMDB Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Classification
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Classification
2. Example of Classification
3. How Deep Neural Network Work on Classification
4. Code Explanation with Output

What is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the class of new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be used to classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, or to classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbors to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

How Deep Neural Network Work on Classification-

Deep neural networks are commonly used for classification tasks because they can automatically learn to extract relevant features from raw input data and map them to the correct output class.

The basic architecture of a deep neural network for classification consists of three main parts: an input layer, one or more hidden layers, and an output layer. The input layer receives the raw input data, which is usually preprocessed to a fixed size and format. The hidden layers are composed of neurons that apply linear transformations and nonlinear activations to the input features to extract relevant patterns and representations. Finally, the output layer produces the predicted class labels, usually as a probability distribution over the possible classes.

During training, the deep neural network learns to adjust its weights and biases in each layer to minimize the difference between the predicted output and the true labels. This is typically done by optimizing a loss function that measures the discrepancy between the predicted and true labels, using techniques such as gradient descent or stochastic gradient descent.

One of the key advantages of deep neural networks for classification is their ability to learn hierarchical representations of the input data. In a deep neural network with multiple hidden layers, each layer learns to capture more complex and abstract features than the previous layer, by building on the representations learned by the earlier layers. This hierarchical structure allows deep neural networks to learn highly discriminative features that can separate different classes of input data, even when the data is highly complex or noisy.

Overall, the effectiveness of deep neural networks for classification depends on the choice of architecture, hyperparameters, and training procedure, as well as the quality and quantity of the training data. When trained properly, deep neural networks can achieve state-of-the-art performance on a wide range of classification tasks, from image recognition to natural language processing.

IMDB Dataset-The IMDB dataset is a large collection of movie reviews collected from the IMDB website, which is a popular source of user-generated movie ratings and reviews. The dataset consists of 50,000 movie reviews, split into 25,000 reviews for training and 25,000 reviews for testing.

Each review is represented as a sequence of words, where each word is represented by an integer index based on its frequency in the dataset. The labels for each review are binary, with 0 indicating a negative review and 1 indicating a positive review.

The IMDB dataset is commonly used as a benchmark for sentiment analysis and text classification tasks, where the goal is to classify the movie reviews as either positive or negative based on their text content. The dataset is challenging because the reviews are often highly subjective and can contain complex language and nuances of meaning, making it difficult for traditional machine learning approaches to accurately classify them.

Deep learning approaches, such as deep neural networks, have achieved state-of-the-art performance on the IMDB dataset by automatically learning to extract relevant features from the raw text data and map them to the correct output class. The IMDB dataset is widely used in research and education for natural language processing and machine learning, as it provides a rich source of labeled text data for training and testing deep learning models.

Source Code and Output-

```
# The IMDB sentiment classification dataset consists of 50,000 movie reviews from IMDB users that are labeled as either positive (1) or negative (0).
# The reviews are preprocessed and each one is encoded as a sequence of word indexes in the form of integers.
# The words within the reviews are indexed by their overall frequency within the dataset. For example, the integer “2” encodes the second most frequent word in the data.
# The 50,000 reviews are split into 25,000 for training and 25,000 for testing.
# Text Process word by word at diffrent timestamp ( You may use RNN LSTM GRU )
# convert input text to vector reprent input text
# DOMAIN: Digital content and entertainment industry
# CONTEXT: The objective of this project is to build a text classification model that analyses the customer's sentiments based on their reviews in the IMDB database. The model uses a complex deep learning model to build an embedding layer followed by a classification algorithm to analyse the sentiment of the customers.
# DATA DESCRIPTION: The Dataset of 50,000 movie reviews from IMDB, labelled by sentiment (positive/negative).
# Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers).
# For convenience, the words are indexed by their frequency in the dataset, meaning the for that has index 1 is the most frequent word.
# Use the first 20 words from each review to speed up training, using a max vocabulary size of 10,000.
# As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.
# PROJECT OBJECTIVE: Build a sequential NLP classifier which can use input text parameters to determine the customer sentiments.
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
#loading imdb data with most frequent 10000 words
```

```
from keras.datasets import imdb
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000) # you may take top 10,000 word frequently used review of movies other are discarded
#consolidating data for EDA Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics
data = np.concatenate((X_train, X_test), axis=0) #  axis 0 is first running vertically downwards across rows (axis 0), axis 1 is second running horizontally across columns (axis 1),
label = np.concatenate((y_train, y_test), axis=0)
X_train.shape
(25000,)
X_test.shape
(25000,)
y_train.shape
(25000,)
y_test.shape
(25000,)
print("Review is ",X_train[0]) # series of no converted word to vocabulory associated with index
print("Review is ",y_train[0])
Review is  [1, 194, 1153, 194, 8255, 78, 228, 5, 6, 1463, 4369, 5012, 134, 26, 4, 715, 8, 118, 1634, 14, 394, 20, 13, 119, 954, 189, 102, 5, 207, 110, 3103, 21, 14, 69, 188, 8, 30, 23, 7, 4, 249, 126, 93, 4, 114, 9, 2300, 1523, 5, 647, 4, 116, 9, 35, 8163, 4, 229, 9, 340, 1322, 4, 118, 9, 4, 130, 4901, 19, 4, 1002, 5, 89, 29, 952, 46, 37, 4, 455, 9, 45, 43, 38, 1543, 1905, 398, 4, 1649, 26, 6853, 5, 163, 11, 3215, 2, 4, 1153, 9, 194, 775, 7, 8255, 2, 349, 2637, 148, 605, 2, 8003, 15, 123, 125, 68, 2, 6853, 15, 349, 165, 4362, 98, 5, 4, 228, 9, 43, 2, 1157, 15, 299, 120, 5, 120, 174, 11, 220, 175, 136, 50, 9, 4373, 228, 8255, 5, 2, 656, 245, 2350, 5, 4, 9837, 131, 152, 491, 18, 2, 32, 7464, 1212, 14, 9, 6, 371, 78, 22, 625, 64, 1382, 9, 8, 168, 145, 23, 4, 1690, 15, 16, 4, 1355, 5, 28, 6, 52, 154, 462, 33, 89, 78, 285, 16, 145, 95]
Review is  0
vocab=imdb.get_word_index() # Retrieve the word index file mapping words to indices
print(vocab)
{'fawn': 34701, 'tsukino': 52006, 'nunnery': 52007, 'sonja': 16816, 'vani': 63951, 'woods': 1408, 'spiders': 16115,
y_train
array([1, 0, 0, ..., 0, 1, 0])
y_test
array([0, 1, 1, ..., 0, 0, 0])
# Function to perform relevant sequence adding on the data
# Now it is time to prepare our data. We will vectorize every review and fill it with zeros so that it
```

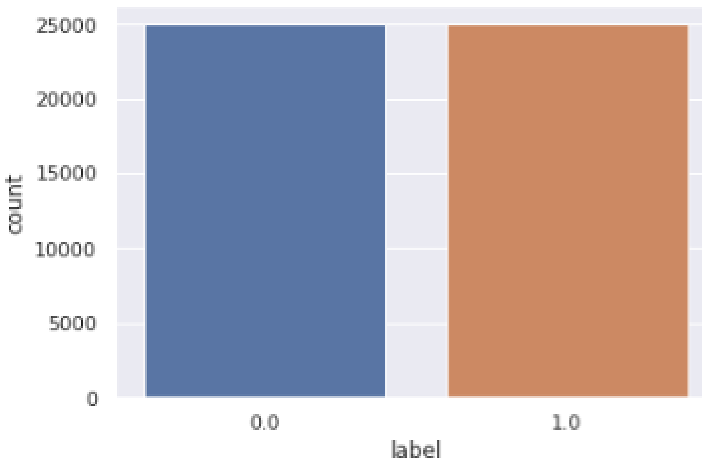
```
contains exactly 10000 numbers.
# That means we fill every review that is shorter than 500 with zeros.
# We do this because the biggest review is nearly that long and every input for our neural network needs
to have the same size.
# We also transform the targets into floats.
# sequences is name of method the review less than 10000 we perform padding overthere
# binary vectorization code:

# VECTORIZE as one cannot feed integers into a NN
# Encoding the integer sequences into a binary matrix - one hot encoder basically
# From integers representing words, at various lengths - to a normalized one hot encoded tensor (matrix)
of 10k columns
def vectorize(sequences, dimension = 10000):    # We will vectorize every review and fill it with zeros
so that it contains exactly 10,000 numbers.
 # Create an all-zero matrix of shape (len(sequences), dimension)
  results = np.zeros((len(sequences), dimension))
  for i, sequence in enumerate(sequences):
    results[i, sequence] = 1
  return results
# Now we split our data into a training and a testing set.
# The training set will contain  reviews and the testing set
# # Set a VALIDATION set

test_x = data[:10000]
test_y = label[:10000]
train_x = data[10000:]
train_y = label[10000:]
test_x.shape
(10000,)
test_y.shape
(10000,)
train_x.shape
(40000,)
train_y.shape
(40000,)
print("Categories:", np.unique(label))
print("Number of unique words:", len(np.unique(np.hstack(data))))
```

```
# The hstack() function is used to stack arrays in sequence horizontally (column wise).
Categories: [0 1]
Number of unique words: 9998
length = [len(i) for i in data]
print("Average Review length:", np.mean(length))
print("Standard Deviation:", round(np.std(length)))

# The whole dataset contains 9998 unique words and the average review length is 234 words, with a standard deviation of 173 words.
Average Review length: 234.75892
Standard Deviation: 173
# If you look at the data you will realize it has been already pre-processed.
# All words have been mapped to integers and the integers represent the words sorted by their frequency.
# This is very common in text analysis to represent a dataset like this.
# So 4 represents the 4th most used word,
# 5 the 5th most used word and so on...
# The integer 1 is reserved for the start marker,
# the integer 2 for an unknown word and 0 for padding.

# Let's look at a single training example:

print("Label:", label[0])
Label: 1
print("Label:", label[1])
Label: 0
print(data[0])
# Retrieves a dict mapping words to their index in the IMDB dataset.
index = imdb.get_word_index()  # word to index
# Create inverted index from a dictionary with document ids as keys and a list of terms as values for each document
reverse_index = dict([(value, key) for (key, value) in index.items()]) # id to word

decoded = " ".join( [reverse_index.get(i - 3, "#") for i in data[0]] )
# The indices are offset by 3 because 0, 1 and 2 are reserved indices for "padding", "start of sequence" and "unknown".
print(decoded)
# this film was just brilliant casting location scenery story direction everyone's really suited the part they
```

played and you could just imagine being there robert # is an amazing actor and now the same being director # father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film

```
#Adding sequence to data
# Vectorization is the process of converting textual data into numerical vectors and is a process that is usually applied once the text is cleaned.
data = vectorize(data)
label = np.array(label).astype("float32")
labelDF=pd.DataFrame({'label':label})
sns.countplot(x='label', data=labelDF)
```

<AxesSubplot:xlabel='label', ylabel='count'>




```
# Creating train and test data set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data,label, test_size=0.20, random_state=1)
X_train.shape
(40000, 10000)
X_test.shape
(10000, 10000)
# Let's create sequential model
from keras.utils import to_categorical
from keras import models
from keras import layers
model = models.Sequential()
# Input - Layer
# Note that we set the input-shape to 10,000 at the input-layer because our reviews are 10,000 integers long.
# The input-layer takes 10,000 as input and outputs it with a shape of 50.
model.add(layers.Dense(50, activation = "relu", input_shape=(10000, )))
```

```
# Hidden - Layers
# Please note you should always use a dropout rate between 20% and 50%. # here in our case 0.3 means 30% dropout we are using dropout to prevent overfitting.
# By the way, if you want you can build a sentiment analysis without LSTMs, then you simply need to replace it by a flatten layer:
model.add(layers.Dropout(0.3, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
model.add(layers.Dropout(0.2, noise_shape=None, seed=None))
model.add(layers.Dense(50, activation = "relu"))
# Output- Layer
model.add(layers.Dense(1, activation = "sigmoid"))
model.summary()
```

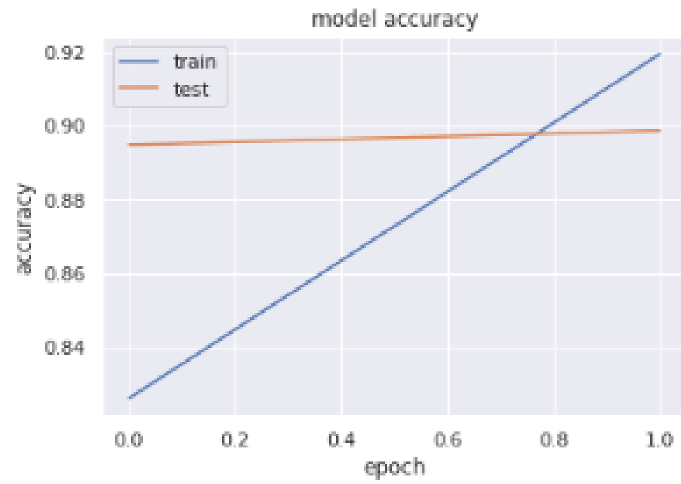
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 50)                500050

 dropout (Dropout)           (None, 50)                0

 dense_1 (Dense)             (None, 50)                2550

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 50)                2550

 dense_3 (Dense)             (None, 1)                 51

=================================================================
Total params: 505,201
Trainable params: 505,201
Non-trainable params: 0
```

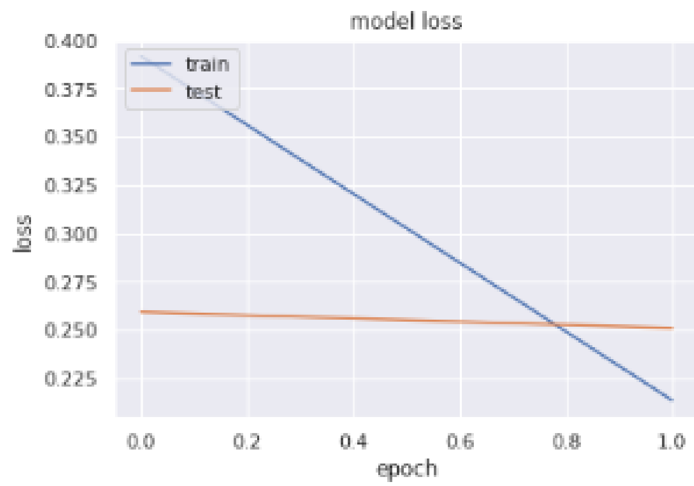
```
#For early stopping
# Stop training when a monitored metric has stopped improving.
# monitor: Quantity to be monitored.
# patience: Number of epochs with no improvement after which training will be stopped.
import tensorflow as tf
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)
# We use the “adam” optimizer, an algorithm that changes the weights and biases during training.
# We also choose binary-crossentropy as loss (because we deal with binary
```

```
classification) and accuracy as our evaluation metric.

model.compile(
 optimizer = "adam",
 loss = "binary_crossentropy",
 metrics = ["accuracy"]
)
from sklearn.model_selection import train_test_split

results = model.fit(
 X_train, y_train,
 epochs= 2,
 batch_size = 500,
 validation_data = (X_test, y_test),
 callbacks=[callback]
)
# Let's check mean accuracy of our model
print(np.mean(results.history["val_accuracy"]))
# Evaluate the model
score = model.evaluate(X_test, y_test, batch_size=500)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
20/20 [==============================] - 1s 24ms/step - loss: 0.2511 - accuracy:
0.8986
Test loss: 0.25108325481414795
Test accuracy: 0.8985999822616577
#Let's plot training history of our model.

# list all data in history
print(results.history.keys())
# summarize history for accuracy
plt.plot(results.history['accuracy'])
plt.plot(results.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(results.history['loss'])
plt.plot(results.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
```

```
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```





Conclusion- In this way we can Classify the Movie Reviews by using DNN.

Assignment Question

1. What is Binary Classification?
2. What is binary Cross Entropy?
3. What is Validation Split?
4. What is the Epoch Cycle?
5. What is Adam Optimizer?

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :......................   Date of Completion:.......................

Group B Deep Learning

Assignment No: 2B

Title of the Assignment: Multiclass classification using Deep Neural Networks: Example: Use the OCR letter recognition dataset https://archive.ics.uci.edu/ml/datasets/letter+recognition

Objective of the Assignment: Students should be able to solve Multiclass classification using Deep Neural NetworksSolve

Prerequisite:

1. Basic of programming language
2. Concept of Multi Classification
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Multi-Classification
2. Example of Multi-Classification
3. How Deep Neural Network Work on Multi-Classification
4. Code Explanation with Output

What is multiclass classification ?

Multi Classification, also known as multiclass classification or multiclass classification problem, is a type of classification problem where the goal is to assign input data to one of three or more classes or categories. In other words, instead of binary classification, where the goal is to assign input data to one of two classes (e.g., positive or negative), multiclass classification involves assigning input data to one of several possible classes or categories (e.g., animal species, types of products, etc.).

In multiclass classification, each input sample is associated with a single class label, and the goal of the model is to learn a function that can accurately predict the correct class label for new, unseen input data. Multiclass classification can be approached using a variety of machine learning algorithms, including decision trees, support vector machines, and deep neural networks.

Some examples of multiclass classification problems include image classification, where the goal is to classify images into one of several categories (e.g., animals, vehicles, buildings), and text classification, where the goal is to classify text documents into one of several categories (e.g., news topics, sentiment analysis).

Example of multiclass classification-

Here are a few examples of multiclass classification problems:

Image classification: The goal is to classify images into one of several categories. For example, an image classification model might be trained to classify images of animals into categories such as cats, dogs, and birds.

Text classification: The goal is to classify text documents into one of several categories. For example, a text classification model might be trained to classify news articles into categories such as politics, sports, and entertainment.

Disease diagnosis: The goal is to diagnose patients with one of several diseases based on their symptoms and medical history. For example, a disease diagnosis model might be trained to classify patients into categories such as diabetes, cancer, and heart disease.

Speech recognition: The goal is to transcribe spoken words into text. A speech recognition model might be trained to recognize spoken words in several languages or dialects.

Credit risk analysis: The goal is to classify loan applicants into categories such as low risk, medium risk, and high risk. A credit risk analysis model might be trained to classify loan applicants based on their credit score, income, and other factors.

In all of these examples, the goal is to assign input data to one of several possible classes or categories.

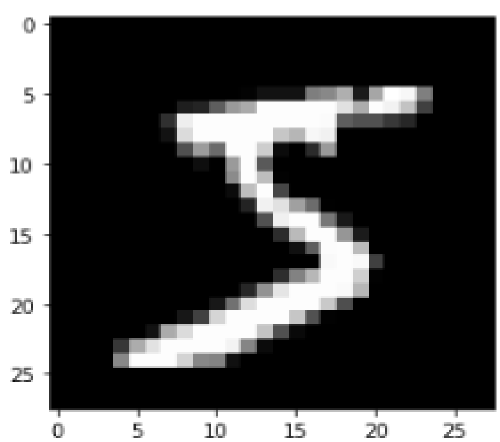
Multiclass classification is a common task in machine learning and can be approached using a variety of algorithms, including decision trees, support vector machines, and deep neural networks.

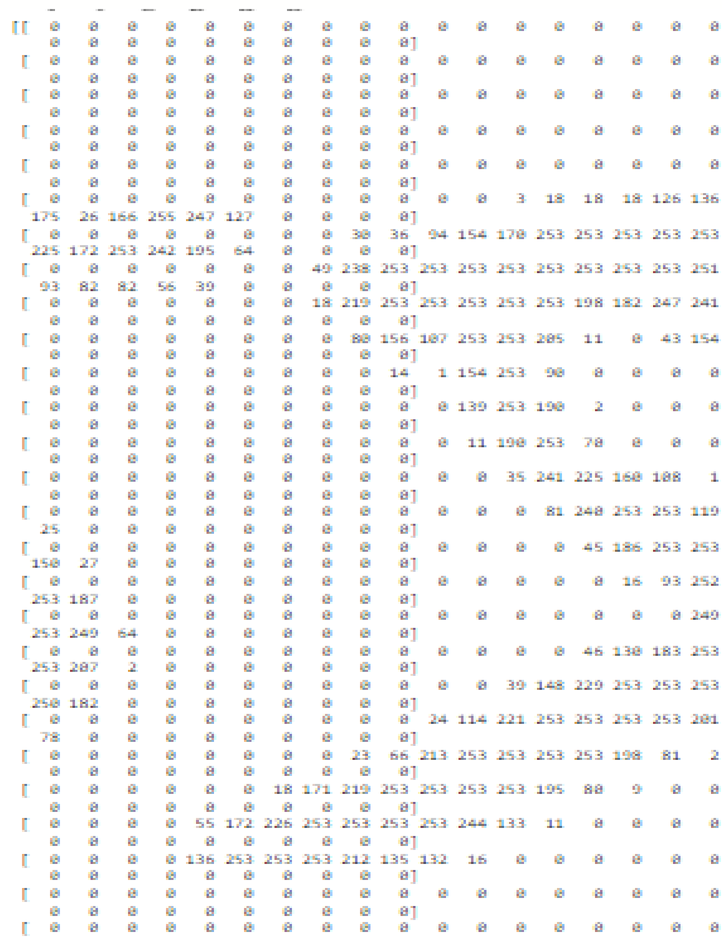
Source Code and Output

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
from sklearn import metrics
# Load the OCR dataset

# The MNIST dataset is a built-in dataset provided by Keras.
# It consists of 70,000 28x28 grayscale images, each of which displays a single handwritten digit from 0
to 9.
# The training set consists of 60,000 images, while the test set has 10,000 images.

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# X_train and X_test are our array of images while y_train and y_test are our array of labels for each
image.
# The first tuple contains the training set features (X_train) and the training set labels (y_train).
# The second tuple contains the testing set features (X_test) and the testing set labels (y_test).
# For example, if the image shows a handwritten 7, then the label will be the intger 7.

plt.imshow(x_train[0], cmap='gray')  # imshow() function which simply displays an image.
plt.show() # cmap is responsible for mapping a specific colormap to the values found in the array that
you passed as the first argument.
# This is because of the format that all the images in the dataset have:
# 1. All the images are grayscale, meaning they only contain black, white and grey.
# 2. The images are 28 pixels by 25 pixels in size (28x28).
print(x_train[0])

# image data is just an array of digits. You can almost make out a 5 from the pattern of the digits in the
array.
# Array of 28 values
```

a grayscale pixel is stored as a digit between 0 and 255 where 0 is black, 255 is white and values in between are different shades of gray.

Therefore, each value in the [28][28] array tells the computer which color to put in that position when.





```
[[   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
   175  26 166 255 247 127   0   0   0   0]
 [   0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
   225 172 253 242 195  64   0   0   0   0]
 [   0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
    93  82  82  56  39   0   0   0   0   0]
 [   0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
    25   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
   150  27   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
   253 187   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
   253 249  64   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
   253 207   2   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
   250 182   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
    78   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0]
 [   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

reformat our X_train array and our X_test array because they do not have the correct shape.

Reshape the data to fit the model

```
print("X_train shape", x_train.shape)
print("y_train shape", y_train.shape)
print("X_test shape", x_test.shape)
print("y_test shape", y_test.shape)
```

Here you can see that for the training sets we have 60,000 elements and the testing sets have 10,000 elements.

y_train and y_test only have 1 dimensional shapes because they are just the labels of each element.

x_train and x_test have 3 dimensional shapes because they have a width and height (28x28 pixels) for each element.

(60000, 28, 28) 1st parameter in the tuple shows us how much image we have 2nd and 3rd parameters are the pixel values from x to y (28x28)

The pixel value varies between 0 to 255.

(60000,) Training labels with integers from 0-9 with dtype of uint8. It has the shape (60000,).

(10000, 28, 28) Testing data that consists of grayscale images. It has the shape (10000, 28, 28) and the dtype of uint8. The pixel value varies between 0 to 255.

(10000,) Testing labels that consist of integers from 0-9 with dtype uint8. It has the shape (10000,).

```
X_train shape (60000, 28, 28)
y_train shape (60000,)
X_test shape (10000, 28, 28)
y_test shape (10000,)
# X: Training data of shape (n_samples, n_features)
# y: Training label values of shape (n_samples, n_labels)
# 2D array of height and width, 28 pixels by 28 pixels will just become 784 pixels (28 squared).
# Remember that X_train has 60,000 elemenets, each with 784 total pixels so will become shape (60000, 784).
# Whereas X_test has 10,000 elements, each with each with 784 total pixels so will become shape (10000, 784).
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32') # use 32-bit precision when training a neural network, so at one point the training data will have to be converted to 32 bit floats. Since the dataset fits easily in RAM, we might as well convert to float immediately.
x_test = x_test.astype('float32')
x_train /= 255  # Each image has Intensity from 0 to 255
x_test /= 255

# Regarding the division by 255, this is the maximum value of a byte (the input feature's type before the conversion to float32),
# so this will ensure that the input features are scaled between 0.0 and 1.0.
# Convert class vectors to binary class matrices
num_classes = 10
y_train = np.eye(num_classes)[y_train]  # Return a 2-D array with ones on the diagonal and zeros elsewhere.
y_test = np.eye(num_classes)[y_test] # f your particular categories is present then it mark as 1 else 0 in remain row
# Define the model architecture
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))  # Input cosist of 784 Neuron ie 784 input, 512 in the hidden layer
model.add(Dropout(0.2)) # DROP OUT RATIO 20%
```

```
model.add(Dense(512, activation='relu')) #returns a sequence of another vectors of dimension 512
model.add(Dropout(0.2))
model.add(Dense(num_classes, activation='softmax')) # 10 neurons ie output node in the output layer.
# Compile the model
model.compile(loss='categorical_crossentropy',  # for a multi-class classification problem
          optimizer=RMSprop(),
          metrics=['accuracy'])
# Train the model
batch_size = 128 # batch_size argument is passed to the layer to define a batch size for the inputs.
epochs = 20
history = model.fit(x_train, y_train,
              batch_size=batch_size,
              epochs=epochs,
              verbose=1, # verbose=1 will show you an animated progress bar eg. [==========]
                  validation_data=(x_test, y_test)) # Using validation_data means you are providing the
training set and validation set yourself,
 # 60000image/128=469 batch each
# Evaluate the model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
Test loss: 0.08541901409626007
Test accuracy: 0.9851999878883362
```

Conclusion- In this way we can do Multi classification using DNN.

Assignment Question

1. What is Batch Size?
2. What is Dropout?
3. What is RMSprop?
4. What is the Softmax Function?
5. What is the Relu Function?

Conclusion- In this way we can Classify the Movie Reviews by using DNN.

Assignment Question

1. What is Batch Size?
2. What is Dropout?
3. What is RMSprop?
4. What is the Softmax Function?
5. What is the Relu Function?

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group B Deep Learning

Assignment No: 3B

Title of the Assignment: Use MNIST Fashion Dataset and create a classifier to classify fashion clothing into categories.

Objective of the Assignment: Students should be able to Classify movie reviews into positive reviews and "negative reviews on IMDB Dataset.

Prerequisite:

1. Basic of programming language
2. Concept of Classification
3. Concept of Deep Neural Network

Contents for Theory:

1. What is Classification
2. Example of Classification
3. What is CNN?
4. How Deep Neural Network Work on Classification
5. Code Explanation with Output

What is Classification?

Classification is a type of supervised learning in machine learning that involves categorizing data into predefined classes or categories based on a set of features or characteristics. It is used to predict the class of new, unseen data based on the patterns learned from the labeled training data.

In classification, a model is trained on a labeled dataset, where each data point has a known class label. The model learns to associate the input features with the corresponding class labels and can then be used to classify new, unseen data.

For example, we can use classification to identify whether an email is spam or not based on its content and metadata, to predict whether a patient has a disease based on their medical records and symptoms, or to classify images into different categories based on their visual features.

Classification algorithms can vary in complexity, ranging from simple models such as decision trees and k-nearest neighbors to more complex models such as support vector machines and neural networks. The choice of algorithm depends on the nature of the data, the size of the dataset, and the desired level of accuracy and interpretability.

Example- Classification is a common task in deep neural networks, where the goal is to predict the class of an input based on its features. Here's an example of how classification can be performed in a deep neural network using the popular MNIST dataset of handwritten digits.

The MNIST dataset contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. Each image is a grayscale 28x28 pixel image, and the task is to classify each image into one of the 10 classes corresponding to the 10 digits.

We can use a convolutional neural network (CNN) to classify the MNIST dataset. A CNN is a type of deep neural network that is commonly used for image classification tasks.

What us CNN-

Convolutional Neural Networks (CNNs) are commonly used for image classification tasks, and they are designed to automatically learn and extract features from input images. Let's consider an example of using a CNN to classify images of handwritten digits.

In a typical CNN architecture for image classification, there are several layers, including convolutional layers, pooling layers, and fully connected layers. Here's a diagram of a simple CNN architecture for the digit classification task:

The input to the network is an image of size 28x28 pixels, and the output is a probability distribution over the 10 possible digits (0 to 9).

The convolutional layers in the CNN apply filters to the input image, looking for specific patterns and features. Each filter produces a feature map that highlights areas of the image that match the filter. The filters are learned during training, so the network can automatically learn which features are most relevant for the classification task.

The pooling layers in the CNN downsample the feature maps, reducing the spatial dimensions of the data. This helps to reduce the number of parameters in the network, while also making the features more robust to small variations in the input image.

The fully connected layers in the CNN take the flattened output from the last pooling layer and perform a classification task by outputting a probability distribution over the 10 possible digits.

During training, the network learns the optimal values of the filters and parameters by minimizing a loss function. This is typically done using stochastic gradient descent or a similar optimization algorithm.

Once trained, the network can be used to classify new images by passing them through the network and computing the output probability distribution.

Overall, CNNs are powerful tools for image recognition tasks and have been used successfully in many applications, including object detection, face recognition, and medical image analysis.

CNNs have a wide range of applications in various fields, some of which are:

Image classification: CNNs are commonly used for image classification tasks, such as identifying objects in images and recognizing faces.

Object detection: CNNs can be used for object detection in images and videos, which involves identifying the location of objects in an image and drawing bounding boxes around them.

Semantic segmentation: CNNs can be used for semantic segmentation, which involves partitioning an image into segments and assigning each segment a semantic label (e.g., "road", "sky", "building").

Natural language processing: CNNs can be used for natural language processing tasks, such as sentiment analysis and text classification.

Medical imaging: CNNs are used in medical imaging for tasks such as diagnosing diseases from X-rays and identifying tumors from MRI scans.

Autonomous vehicles: CNNs are used in autonomous vehicles for tasks such as object detection and lane detection.

Video analysis: CNNs can be used for tasks such as video classification, action recognition, and video captioning.

Overall, CNNs are a powerful tool for a wide range of applications, and they have been used successfully in many areas of research and industry.

How Deep Neural Network Work on Classification using CNN-

Deep neural networks using CNNs work on classification tasks by learning to automatically extract features from input images and using those features to make predictions. Here's how it works:

Input layer: The input layer of the network takes in the image data as input.

Convolutional layers: The convolutional layers apply filters to the input images to extract relevant features. Each filter produces a feature map that highlights areas of the image that match the filter.

Activation functions: An activation function is applied to the output of each convolutional layer to introduce non-linearity into the network.

Pooling layers: The pooling layers downsample the feature maps to reduce the spatial dimensions of the data.

Dropout layer: Dropout is used to prevent overfitting by randomly dropping out a percentage of the neurons in the network during training.

Fully connected layers: The fully connected layers take the flattened output from the last pooling layer and perform a classification task by outputting a probability distribution over the possible classes.

Softmax activation function: The softmax activation function is applied to the output of the last fully connected layer to produce a probability distribution over the possible classes.

Loss function: A loss function is used to compute the difference between the predicted probabilities and the actual labels.

Optimization: An optimization algorithm, such as stochastic gradient descent, is used to minimize the loss function by adjusting the values of the network parameters.

Training: The network is trained on a large dataset of labeled images, adjusting the values of the parameters to minimize the loss function.

Prediction: Once trained, the network can be used to classify new images by passing them through the network and computing the output probability distribution.

MNIST Dataset-

The MNIST Fashion dataset is a collection of 70,000 grayscale images of 28x28 pixels, representing 10 different categories of clothing and accessories. The categories include T-shirts/tops, trousers, pullovers, dresses, coats, sandals, shirts, sneakers, bags, and ankle boots.

The dataset is often used as a benchmark for testing image classification algorithms, and it is considered a more challenging version of the original MNIST dataset which contains handwritten digits. The

MNIST Fashion dataset was released by Zalando Research in 2017 and has since become a popular dataset in the machine learning community.

he MNIST Fashion dataset is a collection of 70,000 grayscale images of 28x28 pixels each. These images represent 10 different categories of clothing and accessories, with each category containing 7,000 images. The categories are as follows:

T-shirt/tops

Trousers

Pullovers

Dresses

Coats

Sandals

Shirts

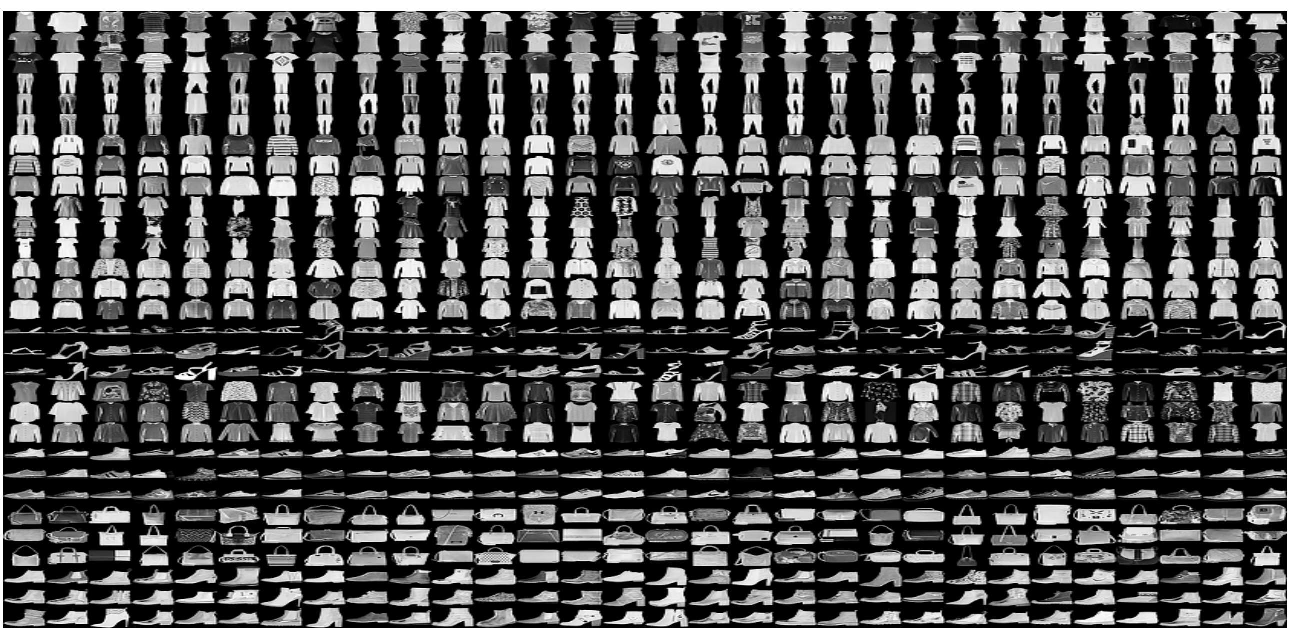
Sneakers

Bags

Ankle boots

The images were obtained from Zalando's online store and are preprocessed to be normalized and centered. The training set contains 60,000 images, while the test set contains 10,000 images. The goal of the dataset is to accurately classify the images into their respective categories.

The MNIST Fashion dataset is often used as a benchmark for testing image classification algorithms, and it is considered a more challenging version of the original MNIST dataset which contains handwritten digits. The dataset is widely used in the machine learning community for research and educational purposes.



Here are the general steps to perform Convolutional Neural Network (CNN) on the MNIST Fashion dataset:

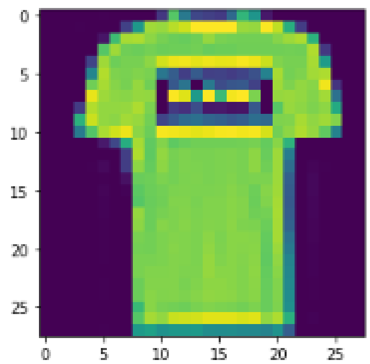
- Import the necessary libraries, including TensorFlow, Keras, NumPy, and Matplotlib.
- Load the dataset using Keras' built-in function, keras.datasets.fashion_mnist.load_data(). This will provide the training and testing sets, which will be used to train and evaluate the CNN.
- Preprocess the data by normalizing the pixel values between 0 and 1, and reshaping the images to be of size (28, 28, 1) for compatibility with the CNN.
- Define the CNN architecture, including the number and size of filters, activation functions, and pooling layers. This can vary based on the specific problem being addressed.
- Compile the model by specifying the loss function, optimizer, and evaluation metrics. Common choices include categorical cross-entropy, Adam optimizer, and accuracy metric.
- Train the CNN on the training set using the fit() function, specifying the number of epochs and batch size.
- Evaluate the performance of the model on the testing set using the evaluate() function. This will provide metrics such as accuracy and loss on the test set.
- Use the trained model to make predictions on new images, if desired, using the predict() function.

Source Code with Output-

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow import keras
import numpy as np

(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()

# There are 10 image classes in this dataset and each class has a mapping corresponding to the following labels:

#0 T-shirt/top
#1 Trouser
#2 pullover
#3 Dress
#4 Coat
#5 sandals
```

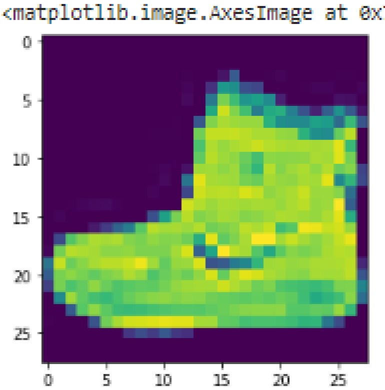
```
#6 shirt
#7 sneaker
#8 bag
#9 ankle boot
plt.imshow(x_train[1])
```

<matplotlib.image.AxesImage at 0x7f85874f3a00>




```
plt.imshow(x_train[0])
```

<matplotlib.image.AxesImage at 0x7f8584b93d00>




```
# Next, we will preprocess the data by scaling the pixel values to be between 0 and 1, and then reshaping the images to be 28x28 pixels.

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

x_train = x_train.reshape(-1, 28, 28, 1)
x_test = x_test.reshape(-1, 28, 28, 1)

# 28, 28 comes from width, height, 1 comes from the number of channels
# -1 means that the length in that dimension is inferred.
# This is done based on the constraint that the number of elements in an ndarray or Tensor when reshaped must remain the same.
```

```
# each image is a row vector (784 elements) and there are lots of such rows (let it be n, so there are 784n elements). So TensorFlow can infer that -1 is n.
# converting the training_images array to 4 dimensional array with sizes 60000, 28, 28, 1 for 0th to 3rd dimension.
x_train.shape
(60000, 28, 28)
x_test.shape
(10000, 28, 28, 1)
y_train.shape
(60000,)
y_test.shape
(10000,)
# We will use a convolutional neural network (CNN) to classify the fashion items.
# The CNN will consist of multiple convolutional layers followed by max pooling,
# dropout, and dense layers. Here is the code for the model:
model = keras.Sequential([
    keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    # 32 filters (default), randomly initialized
    # 3*3 is Size of Filter
    # 28,28,1 size of Input Image
    # No zero-padding: every output 2 pixels less in every dimension
    # in Paramter shwon 320 is value of weights: (3x3 filter weights + 32 bias) * 32 filters
    # 32*3*3=288(Total)+32(bias)= 320
    keras.layers.MaxPooling2D((2,2)),
    # It shown 13 * 13 size image with 32 channel or filter or depth.
    keras.layers.Dropout(0.25),
    # Reduce Overfitting of Training sample drop out 25% Neuron
    keras.layers.Conv2D(64, (3,3), activation='relu'),
    # Deeper layers use 64 filters
    # 3*3 is Size of Filter
    # Observe how the input image on 28x28x1 is transformed to a 3x3x64 feature map
    # 13(Size)-3(Filter Size )+1(bias)=11 Size for Width and Height with 64 Depth or filtter or channel
    # in Paramter shwon 18496 is value of weights: (3x3 filter weights + 64 bias) * 64 filters
    # 64*3*3=576+1=577*32 + 32(bias)=18496
keras.layers.MaxPooling2D((2,2)),
    # It shown 5 * 5 size image with 64 channel or filter or depth.
keras.layers.Dropout(0.25),
```

```
    keras.layers.Conv2D(128, (3,3), activation='relu'),
   # Deeper layers use 128 filters
   # 3*3 is Size of Filter
   # Observe how the input image on 28x28x1 is transformed to a 3x3x128 feature map
    # It show 5(Size)-3(Filter Size )+1(bias)=3 Size for Width and Height with 64 Depth or filtter or channel
   # 128*3*3=1152+1=1153*64 + 64(bias)= 73856
   # To classify the images, we still need a Dense and Softmax layer.
   # We need to flatten the 3x3x128 feature map to a vector of size 1152
    keras.layers.Flatten(),
   keras.layers.Dense(128, activation='relu'),
   # 128 Size of Node in Dense Layer
   # 1152*128 = 147584
   keras.layers.Dropout(0.25),
   keras.layers.Dense(10, activation='softmax')
   # 10 Size of Node another Dense Layer
   # 128*10+10 bias= 1290
])
model.summary()
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 5, 5, 64)          0
```

```
conv2d_2 (Conv2D)        (None, 3, 3, 128)       73856

flatten (Flatten)        (None, 1152)            0

dense (Dense)            (None, 128)             147584

dropout_2 (Dropout)      (None, 128)             0

dense_1 (Dense)          (None, 10)              1290

=================================================================
Total params: 241,546
Trainable params: 241,546
Non-trainable params: 0
# Compile and Train the Model
# After defining the model, we will compile it and train it on the training data.
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))
# 1875 is a number of batches. By default batches contain 32 samles.60000 / 32 = 1875
# Finally, we will evaluate the performance of the model on the test data.
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
313/313 [==============================] - 3s 10ms/step - loss: 0.2606 - accuracy: 0.9031
Test accuracy: 0.9031000137329102
```

Conclusion- In this way we can Classify fashion clothing into categories using CNN.

Assignment Question

1. What is Binary Classification?
2. What is binary Cross Entropy?
3. What is Validation Split?
4. What is the Epoch Cycle?
5. What is Adam Optimizer?

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group A

Assignment No: 1(A)

Title of the Assignment: Design and implement Parallel Breadth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for BFS

Objective of the Assignment: Students should be able to perform Parallel Breadth First Search based on existing algorithms using OpenMP

Prerequisite:

1. Basic of programming language
2. Concept of BFS
3. Concept of Parallelism

Contents for Theory:

1. **What is BFS?**
2. **Example of BFS**
3. **Concept of OpenMP**
4. **How Parallel BFS Work**
5. **Code Explanation with Output**

What is BFS?

BFS stands for Breadth-First Search. It is a graph traversal algorithm used to explore all the nodes of a graph or tree systematically, starting from the root node or a specified starting point, and visiting all the neighboring nodes at the current depth level before moving on to the next depth level.

The algorithm uses a queue data structure to keep track of the nodes that need to be visited, and marks each visited node to avoid processing it again. The basic idea of the BFS algorithm is to visit all the nodes at a given level before moving on to the next level, which ensures that all the nodes are visited in breadth-first order.

BFS is commonly used in many applications, such as finding the shortest path between two nodes, solving puzzles, and searching through a tree or graph.

Example of BFS

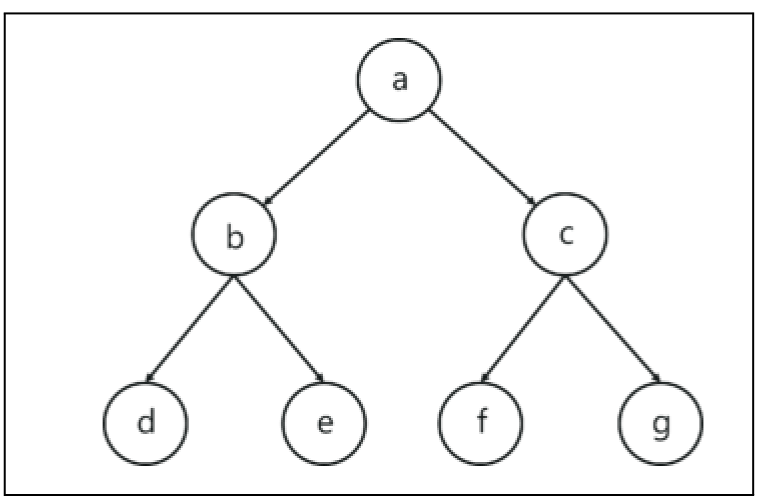
Now let's take a look at the steps involved in traversing a graph by using Breadth-First Search:

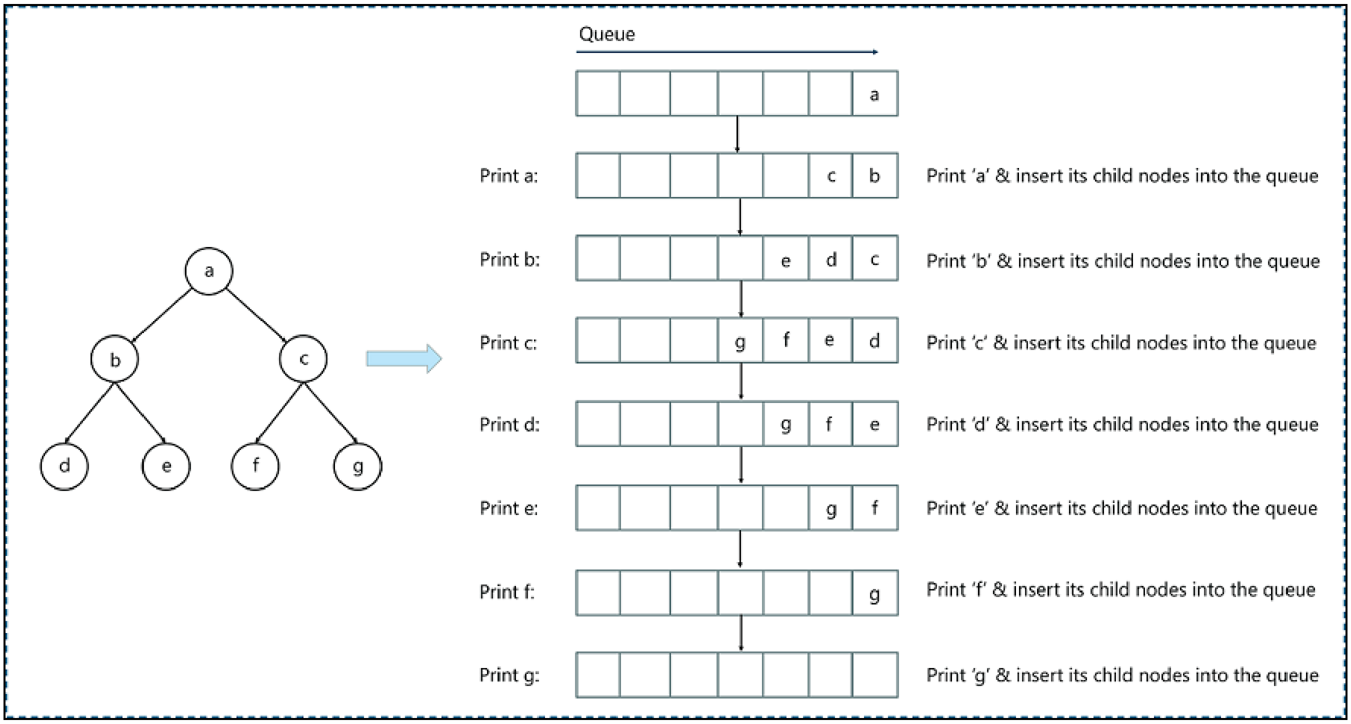
Step 1: Take an Empty Queue.

Step 2: Select a starting node (visiting a node) and insert it into the Queue.

Step 3: Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes (exploring a node) into the Queue.

Step 4: Print the extracted node.





Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.
- OpenMP is widely used in scientific computing, engineering, and other fields that require high-performance computing. It is supported by most modern compilers and is available on a wide range of platforms, including desktops, servers, and supercomputers.

How Parallel BFS Work

- Parallel BFS (Breadth-First Search) is an algorithm used to explore all the nodes of a graph or tree systematically in parallel. It is a popular parallel algorithm used for graph traversal in distributed computing, shared-memory systems, and parallel clusters.

- The parallel BFS algorithm starts by selecting a root node or a specified starting point, and then assigning it to a thread or processor in the system. Each thread maintains a local queue of nodes to be visited and marks each visited node to avoid processing it again.
- The algorithm then proceeds in levels, where each level represents a set of nodes that are at a certain distance from the root node. Each thread processes the nodes in its local queue at the current level, and then exchanges the nodes that are adjacent to the current level with other threads or processors. This is done to ensure that the nodes at the next level are visited by the next iteration of the algorithm.
- The parallel BFS algorithm uses two phases: the computation phase and the communication phase. In the computation phase, each thread processes the nodes in its local queue, while in the communication phase, the threads exchange the nodes that are adjacent to the current level with other threads or processors.
- The parallel BFS algorithm terminates when all nodes have been visited or when a specified node has been found. The result of the algorithm is the set of visited nodes or the shortest path from the root node to the target node.
- Parallel BFS can be implemented using different parallel programming models, such as OpenMP, MPI, CUDA, and others. The performance of the algorithm depends on the number of threads or processors used, the size of the graph, and the communication overhead between the threads or processors.

Conclusion- In this way we can achieve parallelism while implementing BFS

Assignment Question

1. **What if BFS?**
2. **What is OpenMP?What is its significance in parallel programming?**
3. **Write down applications of Parallel BFS**
4. **How can BFS be parallelized using OpenMP? Describe the parallel BFS algorithm using OpenMP.**
5. **Write Down Commands used in OpenMP?**

Reference link

- https://www.edureka.co/blog/breadth-first-search-algorithm/

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group A

Assignment No: 1(B)

Title of the Assignment: Design and implement Parallel Depth First Search based on existing algorithms using OpenMP. Use a Tree or an undirected graph for DFS

Objective of the Assignment: Students should be able to perform Parallel Depth First Search based on existing algorithms using OpenMP

Prerequisite:

1. Basic of programming language
2. Concept of DFS
3. Concept of Parallelism

Contents for Theory:

1. **What is DFS?**
2. **Example of DFS**
3. **Concept of OpenMP**
4. **How Parallel DFS Work**

What is DFS?

DFS stands for Depth-First Search. It is a popular graph traversal algorithm that explores as far as possible along each branch before backtracking. This algorithm can be used to find the shortest path between two vertices or to traverse a graph in a systematic way. The algorithm starts at the root node and explores as far as possible along each branch before backtracking. The backtracking is done to explore the next branch that has not been explored yet.

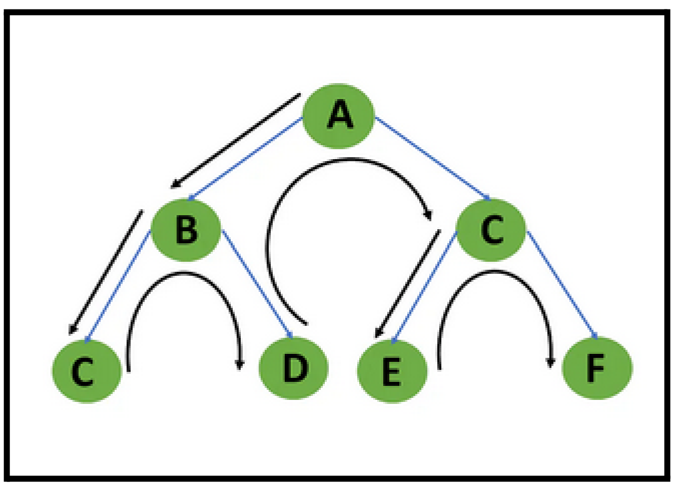
DFS can be implemented using either a recursive or an iterative approach. The recursive approach is simpler to implement but can lead to a stack overflow error for very large graphs. The iterative approach uses a stack to keep track of nodes to be explored and is preferred for larger graphs.

DFS can also be used to detect cycles in a graph. If a cycle exists in a graph, the DFS algorithm will eventually reach a node that has already been visited, indicating that a cycle exists.

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.



Example of DFS:

To implement DFS traversal, you need to take the following stages.

Step 1: Create a stack with the total number of vertices in the graph as the size.

Step 2: Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.

Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.

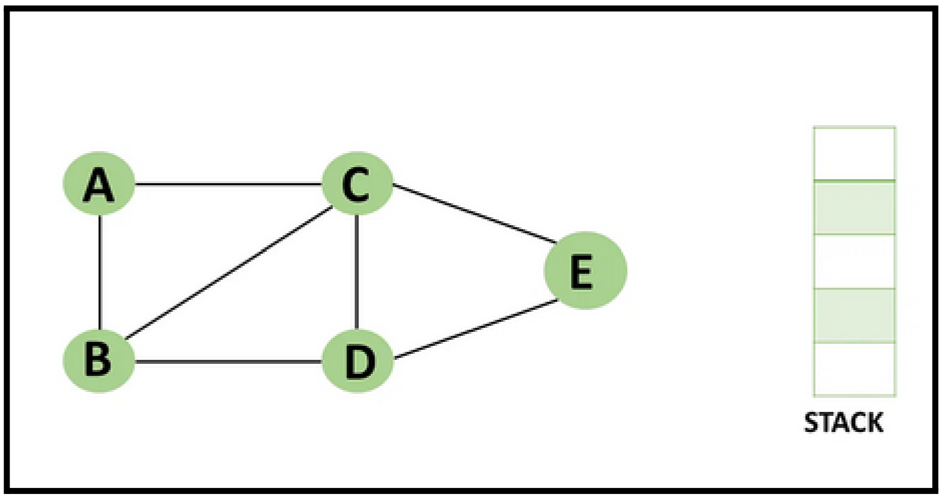
Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.

Step 5 - If there are no new vertices to visit, go back and pop one from the stack using backtracking.

Step 6 - Continue using steps 3, 4, and 5 until the stack is empty.

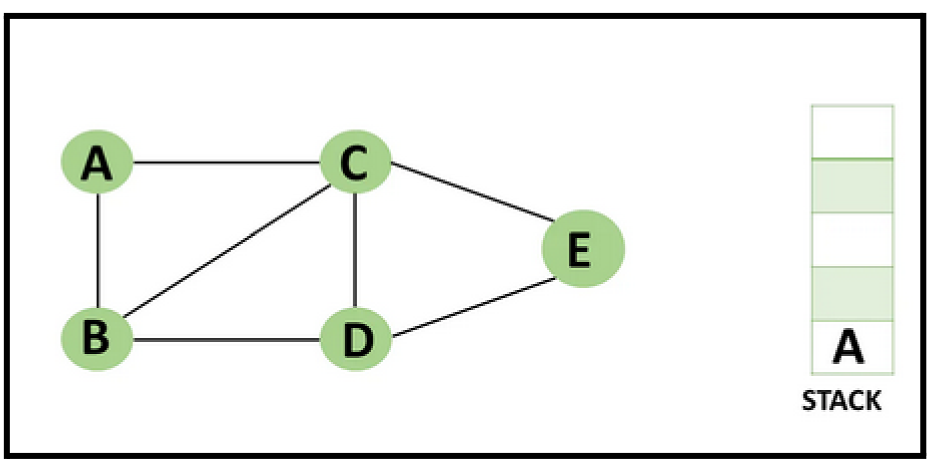
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

Consider the following graph as an example of how to use the dfs algorithm.



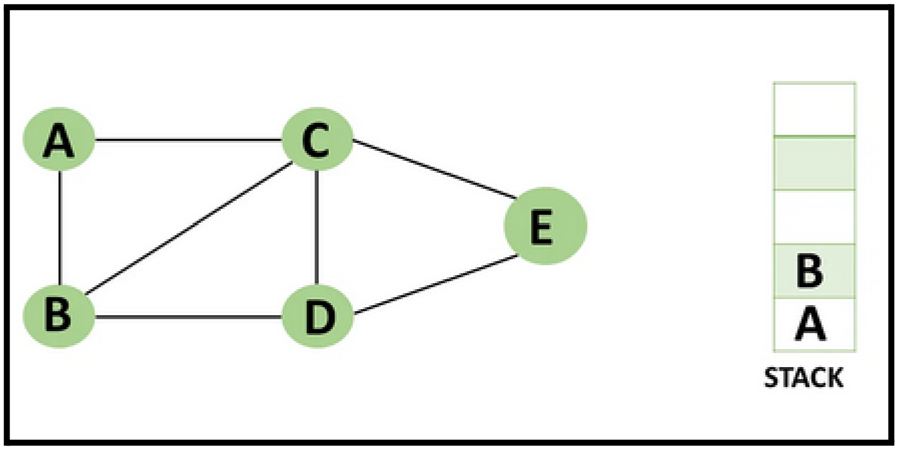
Step 1: Mark vertex A as a visited source node by selecting it as a source node.

- You should push vertex A to the top of the stack.



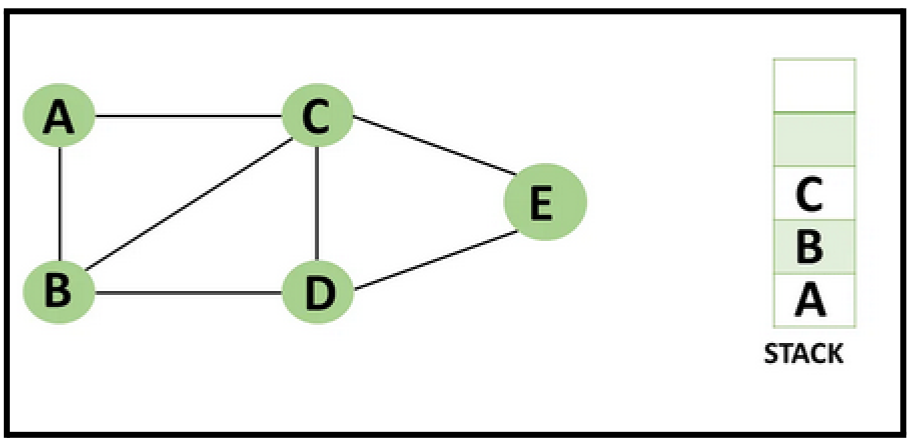
Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

You should push vertex B to the top of the stack



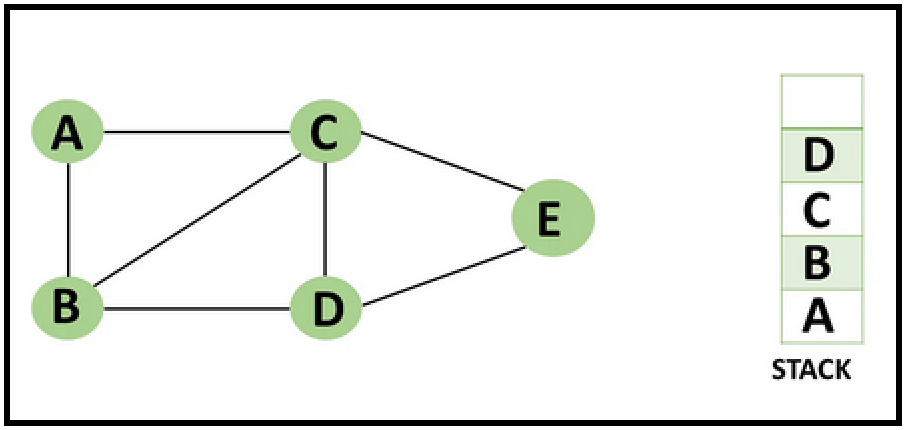
Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

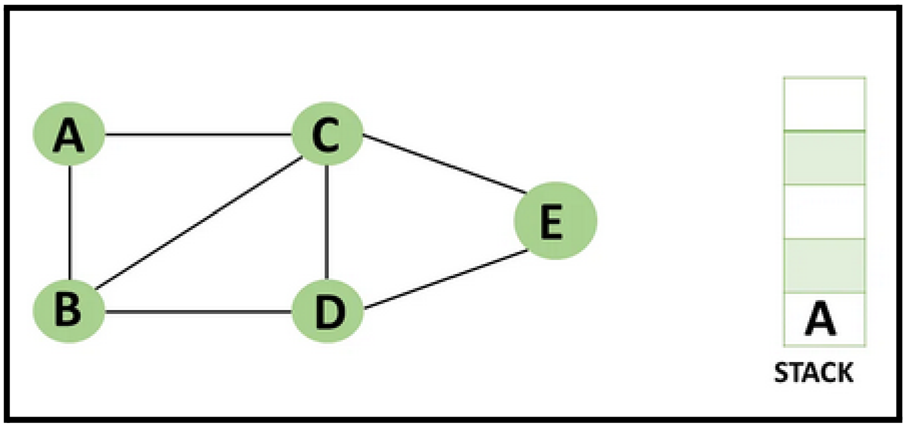
- Vertex C is pushed to the top of the stack.



Step 4: You can visit any nearby unvisited vertices of vertex C, you need to select vertex D and designate it as a visited vertex.

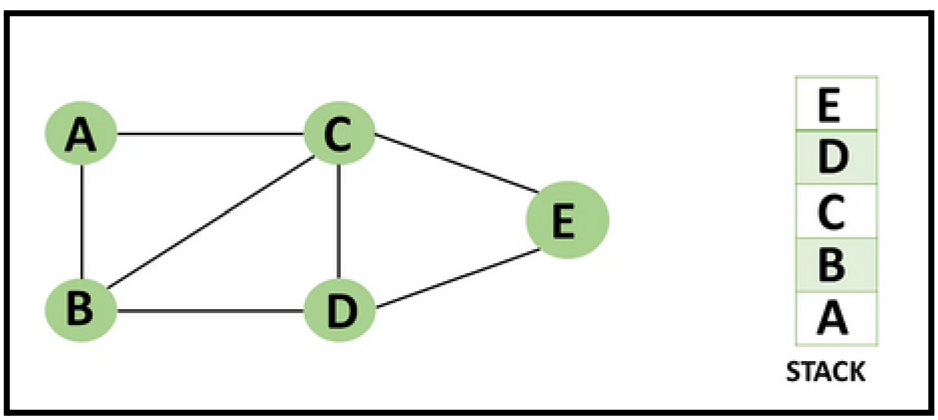
- Vertex D is pushed to the top of the stack.



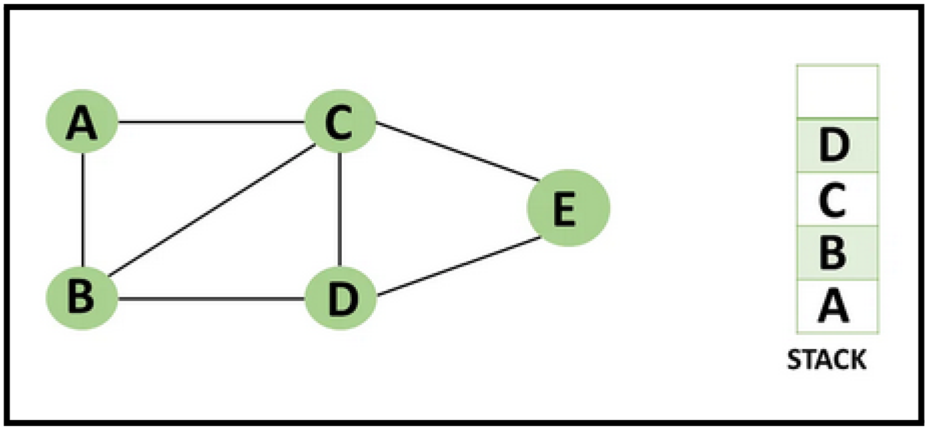


Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

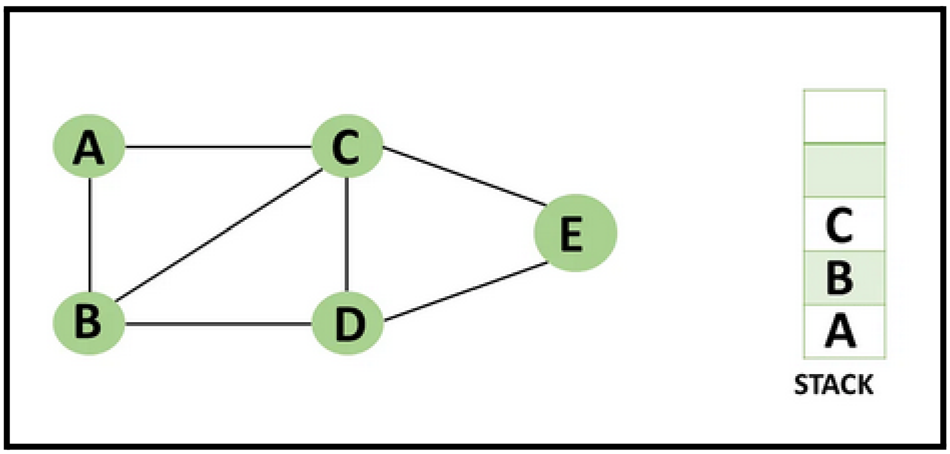
- Vertex E should be pushed to the top of the stack.



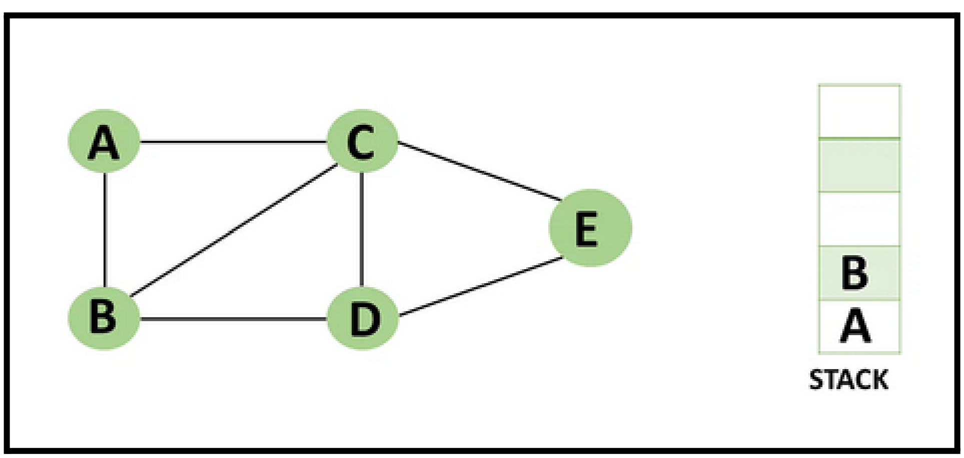
Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.



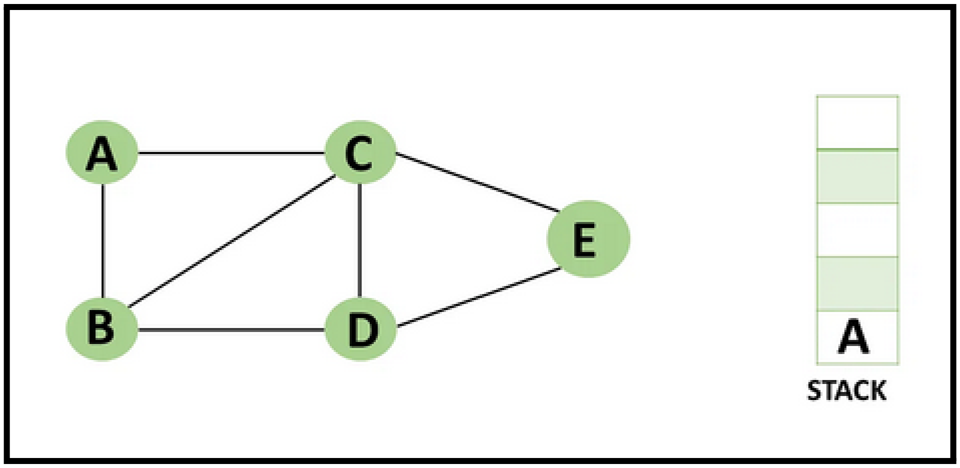
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.



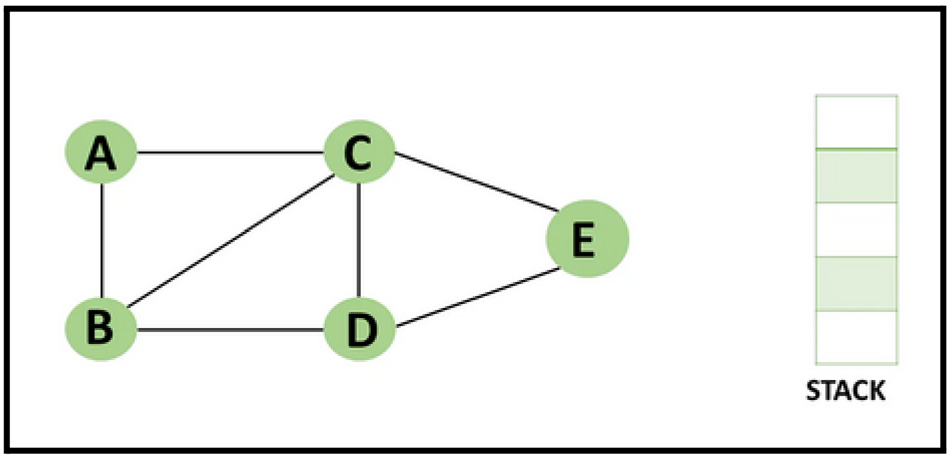
Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.



Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.



Step 10: All of the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.



Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to

execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

How Parallel DFS Work

- Parallel Depth-First Search (DFS) is an algorithm that explores the depth of a graph structure to search for nodes. In contrast to a serial DFS algorithm that explores nodes in a sequential manner, parallel DFS algorithms explore nodes in a parallel manner, providing a significant speedup in large graphs.

- Parallel DFS works by dividing the graph into smaller subgraphs that are explored simultaneously. Each processor or thread is assigned a subgraph to explore, and they work independently to explore the subgraph using the standard DFS algorithm. During the exploration process, the nodes are marked as visited to avoid revisiting them.

- To explore the subgraph, the processors maintain a stack data structure that stores the nodes in the order of exploration. The top node is picked and explored, and its adjacent nodes are pushed onto the stack for further exploration. The stack is updated concurrently by the processors as they explore their subgraphs.

- Parallel DFS can be implemented using several parallel programming models such as OpenMP, MPI, and CUDA. In OpenMP, the #pragma omp parallel for directive is used to distribute the work among multiple threads. By using this directive, each thread operates on a different part of the graph, which increases the performance of the DFS algorithm.

Conclusion- In this way we can achieve parallelism while implementing DFS

Assignment Question

1. **What if DFS?**
2. **Write a parallel Depth First Search (DFS) algorithm using OpenMP**
3. **What is the advantage of using parallel programming in DFS?**
4. **How can you parallelize a DFS algorithm using OpenMP?**
5. **What is a race condition in parallel programming, and how can it be avoided in OpenMP?**

Reference link

- https://www.programiz.com/dsa/graph-dfs
- https://www.simplilearn.com/tutorials/data-structure-tutorial/dfs-algorithm

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group A

Assignment No: 2(B)

Title of the Assignment: Write a program to implement Parallel Bubble Sort. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Objective of the Assignment: Students should be able to Write a program to implement Parallel Bubble Sort and can measure the performance of sequential and parallel algorithms.

Prerequisite:

1. Basic of programming language
2. Concept of Bubble Sort
3. Concept of Parallelism

Contents for Theory:

1. **What is Bubble Sort? Use of Bubble Sort**
2. **Example of Bubble sort?**
3. **Concept of OpenMP**
4. **How Parallel Bubble Sort Work**
5. **How to measure the performance of sequential and parallel algorithms?**

What is Bubble Sort?

Bubble Sort is a simple sorting algorithm that works by repeatedly swapping adjacent elements if they are in the wrong order. It is called "bubble" sort because the algorithm moves the larger elements towards the end of the array in a manner that resembles the rising of bubbles in a liquid.

The basic algorithm of Bubble Sort is as follows:

1. Start at the beginning of the array.
2. Compare the first two elements. If the first element is greater than the second element, swap them.
3. Move to the next pair of elements and repeat step 2.
4. Continue the process until the end of the array is reached.
5. If any swaps were made in step 2-4, repeat the process from step 1.

The time complexity of Bubble Sort is O(n^2), which makes it inefficient for large lists. However, it has the advantage of being easy to understand and implement, and it is useful for educational purposes and for sorting small datasets.

Bubble Sort has limited practical use in modern software development due to its inefficient time complexity of O(n^2) which makes it unsuitable for sorting large datasets. However, Bubble Sort has some advantages and use cases that make it a valuable algorithm to understand, such as:

1. Simplicity: Bubble Sort is one of the simplest sorting algorithms, and it is easy to understand and implement. It can be used to introduce the concept of sorting to beginners and as a basis for more complex sorting algorithms.
2. Educational purposes: Bubble Sort is often used in academic settings to teach the principles of sorting algorithms and to help students understand how algorithms work.
3. Small datasets: For very small datasets, Bubble Sort can be an efficient sorting algorithm, as its overhead is relatively low.
4. Partially sorted datasets: If a dataset is already partially sorted, Bubble Sort can be very efficient. Since Bubble Sort only swaps adjacent elements that are in the wrong order, it has a low number of operations for a partially sorted dataset.
5. Performance optimization: Although Bubble Sort itself is not suitable for sorting large datasets, some of its techniques can be used in combination with other sorting algorithms to optimize their performance. For example, Bubble Sort can be used to optimize the performance of Insertion Sort by reducing the number of comparisons needed.

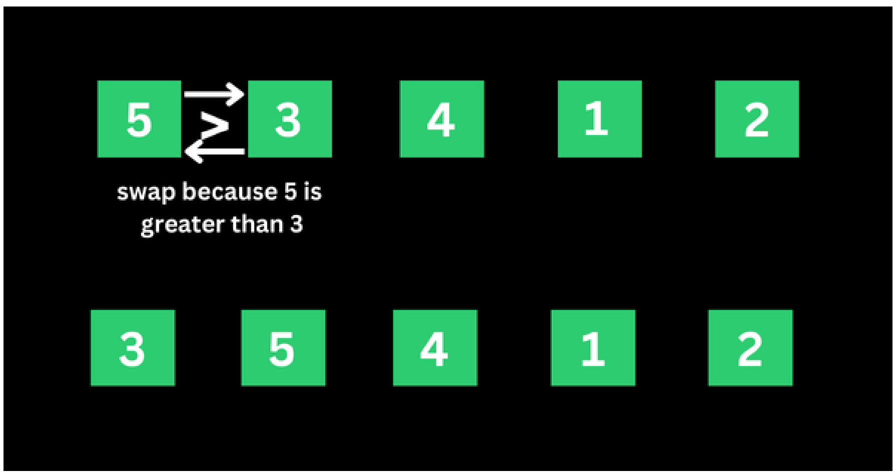
Example of Bubble sort

Let's say we want to sort a series of numbers 5, 3, 4, 1, and 2 so that they are arranged in ascending order…

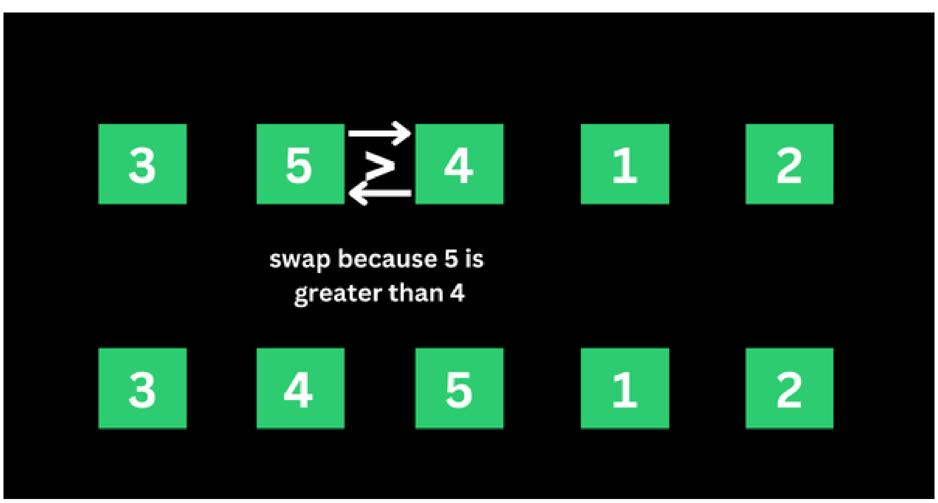
The sorting begins the first iteration by comparing the first two values. If the first value is greater than the second, the algorithm pushes the first value to the index of the second value.

First Iteration of the Sorting

Step 1: In the case of 5, 3, 4, 1, and 2, 5 is greater than 3. So 5 takes the position of 3 and the numbers become 3, 5, 4, 1, and 2.

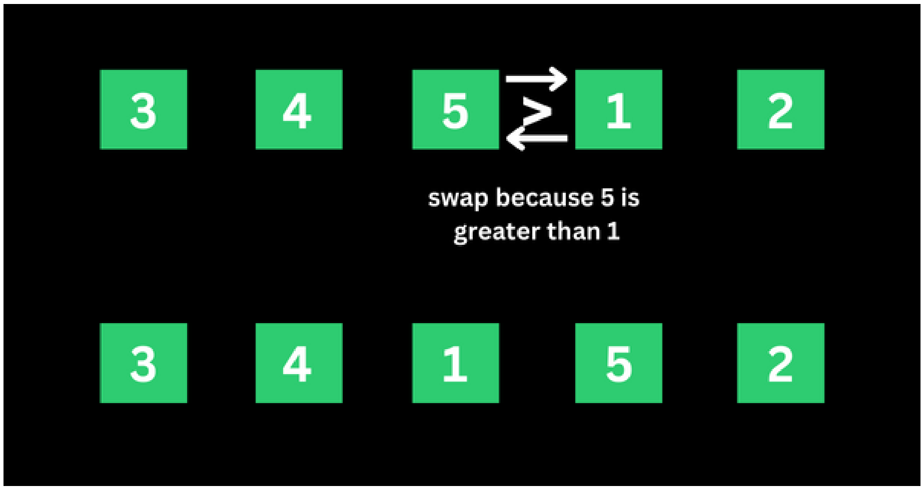


Step 2: The algorithm now has 3, 5, 4, 1, and 2 to compare, this time around, it compares the next two values, which are 5 and 4. 5 is greater than 4, so 5 takes the index of 4 and the values now become 3, 4, 5, 1, and 2.

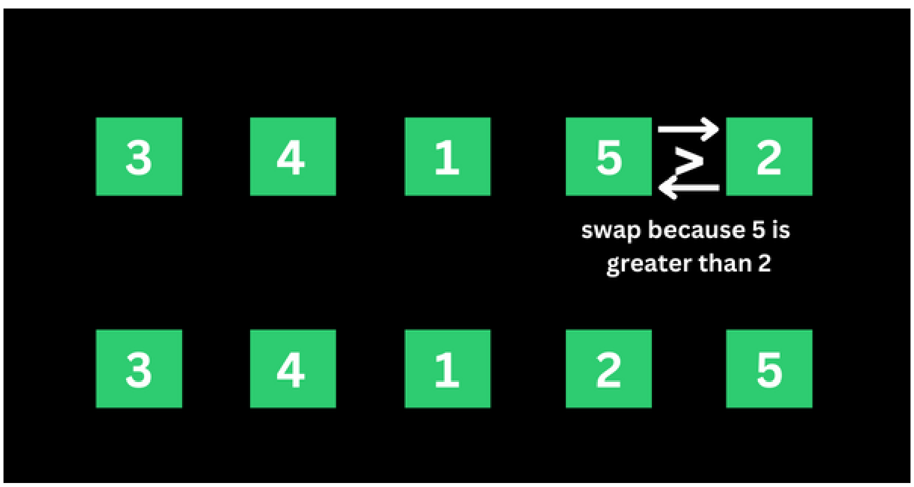


Step 3: The algorithm now has 3, 4, 5, 1, and 2 to compare. It compares the next two values, which are 5

and 1. 5 is greater than 1, so 5 takes the index of 1 and the numbers become 3, 4, 1, 5, and 2.



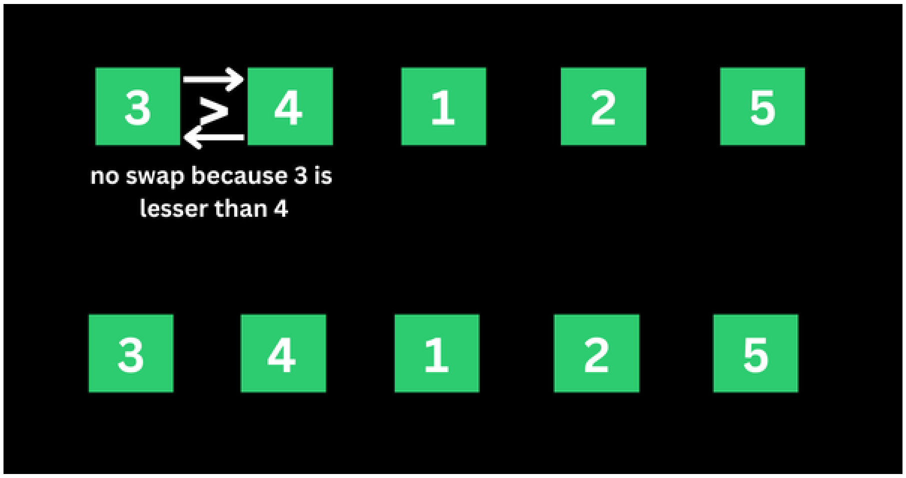
Step 4: The algorithm now has 3, 4, 1, 5, and 2 to compare. It compares the next two values, which are 5 and 2. 5 is greater than 2, so 5 takes the index of 2 and the numbers become 3, 4, 1, 2, and 5.



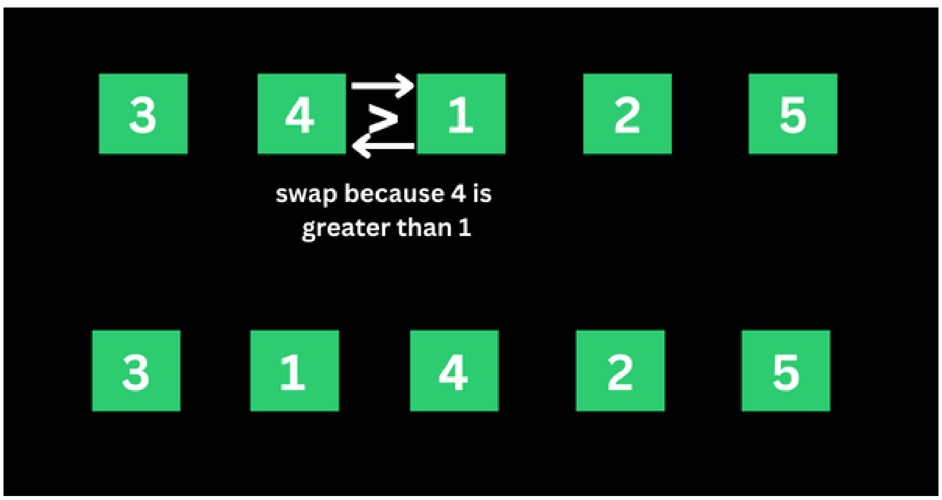
That's the first iteration. And the numbers are now arranged as 3, 4, 1, 2, and 5 – from the initial 5, 3, 4, 1, and 2. As you might realize, 5 should be the last number if the numbers are sorted in ascending order. This means the first iteration is really completed.

Second Iteration of the Sorting and the Rest

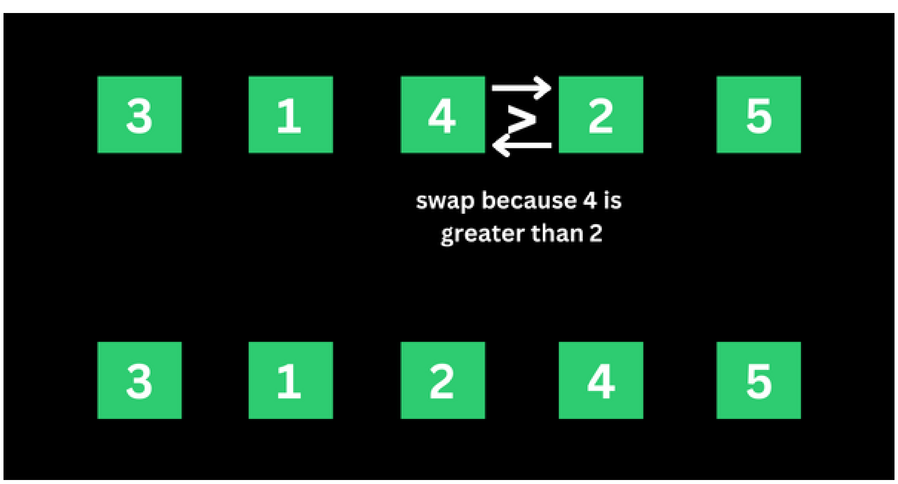
The algorithm starts the second iteration with the last result of 3, 4, 1, 2, and 5. This time around, 3 is smaller than 4, so no swapping happens. This means the numbers will remain the same.



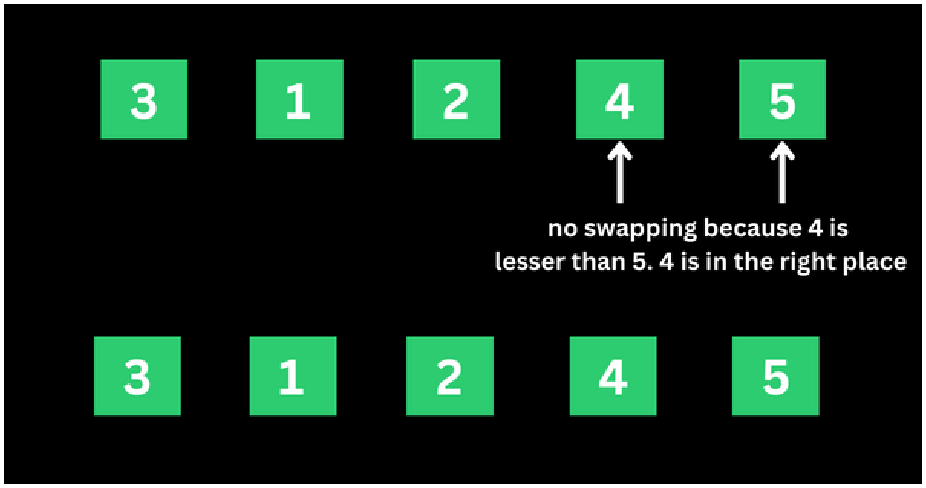
The algorithm proceeds to compare 4 and 1. 4 is greater than 1, so 4 is swapped for 1 and the numbers become 3, 1, 4, 2, and 5.



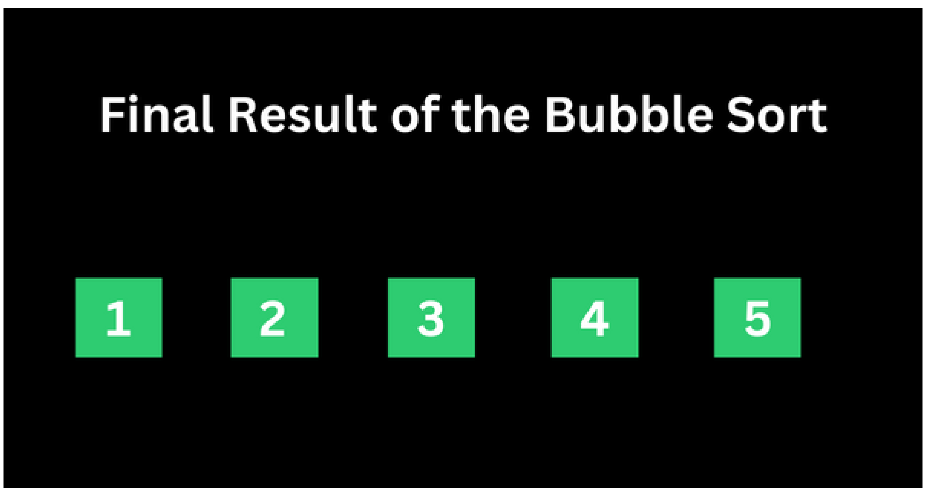
The algorithm now proceeds to compare 4 and 2. 4 is greater than 2, so 4 is swapped for 2 and the numbers become 3, 1, 2, 4, and 5.



4 is now in the right place, so no swapping occurs between 4 and 5 because 4 is smaller than 5.



That's how the algorithm continues to compare the numbers until they are arranged in ascending order of 1, 2, 3, 4, and 5.



Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern

processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

How Parallel Bubble Sort Work

- Parallel Bubble Sort is a modification of the classic Bubble Sort algorithm that takes advantage of parallel processing to speed up the sorting process.

- In parallel Bubble Sort, the list of elements is divided into multiple sublists that are sorted concurrently by multiple threads. Each thread sorts its sublist using the regular Bubble Sort algorithm. When all sublists have been sorted, they are merged together to form the final sorted list.

- The parallelization of the algorithm is achieved using OpenMP, a programming API that supports parallel processing in C++, Fortran, and other programming languages. OpenMP provides a set of compiler directives that allow developers to specify which parts of the code can be executed in parallel.

- In the parallel Bubble Sort algorithm, the main loop that iterates over the list of elements is divided into multiple iterations that are executed concurrently by multiple threads. Each thread sorts a subset of the list, and the threads synchronize their work at the end of each iteration to ensure that the elements are properly ordered.

- Parallel Bubble Sort can provide a significant speedup over the regular Bubble Sort algorithm, especially when sorting large datasets on multi-core processors. However, the speedup is limited by the overhead of thread creation and synchronization, and it may not be worth the effort for small datasets or when using a single-core processor.

How to measure the performance of sequential and parallel algorithms?

To measure the performance of sequential Bubble sort and parallel Bubble sort algorithms, you can follow these steps:

1. Implement both the sequential and parallel Bubble sort algorithms.
2. Choose a range of test cases, such as arrays of different sizes and different degrees of sortedness, to test the performance of both algorithms.

3. Use a reliable timer to measure the execution time of each algorithm on each test case.
4. Record the execution times and analyze the results.

When measuring the performance of the parallel Bubble sort algorithm, you will need to specify the number of threads to use. You can experiment with different numbers of threads to find the optimal value for your system.

Here are some additional tips for measuring performance:

- Run each algorithm multiple times on each test case and take the average execution time to reduce the impact of variations in system load and other factors.
- Monitor system resource usage during execution, such as CPU utilization and memory consumption, to detect any performance bottlenecks.
- Visualize the results using charts or graphs to make it easier to compare the performance of the two algorithms.

How to check CPU utilization and memory consumption in ubuntu

In Ubuntu, you can use a variety of tools to check CPU utilization and memory consumption. Here are some common tools:

1. **top:** The top command provides a real-time view of system resource usage, including CPU utilization and memory consumption. To use it, open a terminal window and type top. The output will display a list of processes sorted by resource usage, with the most resource-intensive processes at the top.
2. **htop**: htop is a more advanced version of top that provides additional features, such as interactive process filtering and a color-coded display. To use it, open a terminal window and type htop.
3. **ps**: The ps command provides a snapshot of system resource usage at a particular moment in time. To use it, open a terminal window and type ps aux. This will display a list of all running processes and their resource usage.
4. **free:** The free command provides information about system memory usage, including total, used, and free memory. To use it, open a terminal window and type free -h.
5. **vmstat:** The vmstat command provides a variety of system statistics, including CPU utilization, memory usage, and disk activity. To use it, open a terminal window and type vmstat.

Conclusion- In this way we can implement Bubble Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

Assignment Question

1. **What is parallel Bubble Sort?**
2. **How does Parallel Bubble Sort work?**
3. **How do you implement Parallel Bubble Sort using OpenMP?**
4. **What are the advantages of Parallel Bubble Sort?**
5. **Difference between serial bubble sort and parallel bubble sort**

Reference link

- https://www.freecodecamp.org/news/bubble-sort-algorithm-in-java-cpp-python-with-example-code/

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group A

Assignment No: 2(B)

Title of the Assignment: Write a program to implement Parallel Merge Sort. Use existing algorithms and measure the performance of sequential and parallel algorithms.

Objective of the Assignment: Students should be able to Write a program to implement Parallel Merge Sort and can measure the performance of sequential and parallel algorithms.

Prerequisite:

1. Basic of programming language
2. Concept of Merge Sort
3. Concept of Parallelism

Contents for Theory:

1. **What is Merge? Use of Merge Sort**
2. **Example of Merge sort?**
3. **Concept of OpenMP**
4. **How Parallel Merge Sort Work**
5. **How to measure the performance of sequential and parallel algorithms?**

What is Merge Sort?

Merge sort is a sorting algorithm that uses a divide-and-conquer approach to sort an array or a list of elements. The algorithm works by recursively dividing the input array into two halves, sorting each half, and then merging the sorted halves to produce a sorted output.

The merge sort algorithm can be broken down into the following steps:

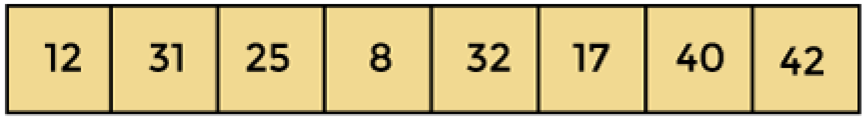
1. Divide the input array into two halves.
2. Recursively sort the left half of the array.
3. Recursively sort the right half of the array.
4. Merge the two sorted halves into a single sorted output array.

- The merging step is where the bulk of the work happens in merge sort. The algorithm compares the first elements of each sorted half, selects the smaller element, and appends it to the output array. This process continues until all elements from both halves have been appended to the output array.
- The time complexity of merge sort is O(n log n), which makes it an efficient sorting algorithm for large input arrays. However, merge sort also requires additional memory to store the output array, which can make it less suitable for use with limited memory resources.
- In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.
- One thing that you might wonder is what is the specialty of this algorithm. We already have a number of sorting algorithms then why do we need this algorithm? One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.
- Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine.

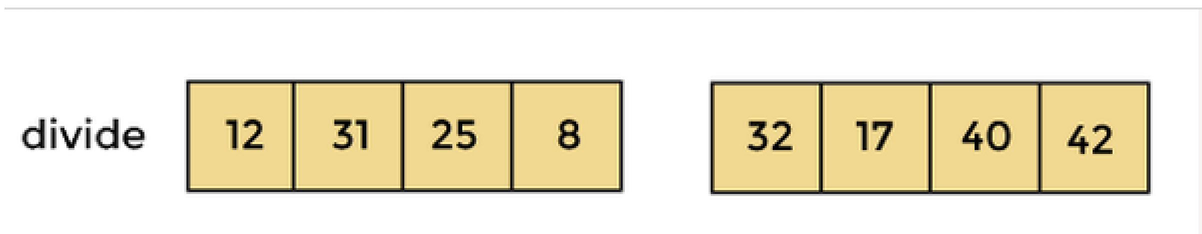
Example of Merge sort

Now, let's see the working of merge sort Algorithm. To understand the working of the merge sort algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example.

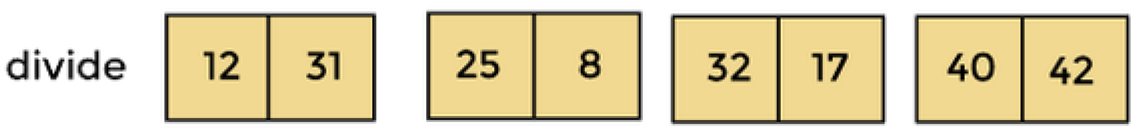
Let the elements of array are -



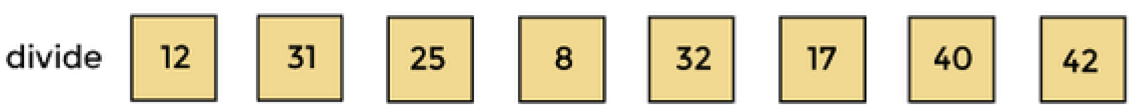
- According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.
- As there are eight elements in the given array, so it is divided into two arrays of size 4.



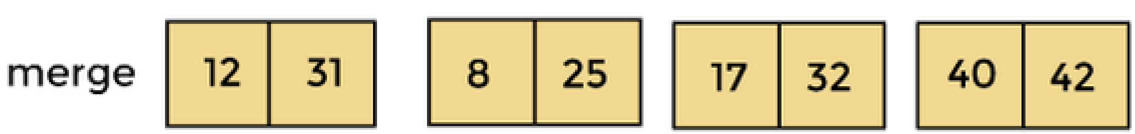
- Now, again divide these two arrays into halves. As they are of size 4, divide them into new arrays of size 2.



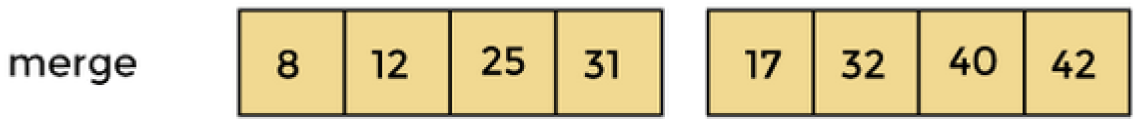
- Now, again divide these arrays to get the atomic value that cannot be further divided.



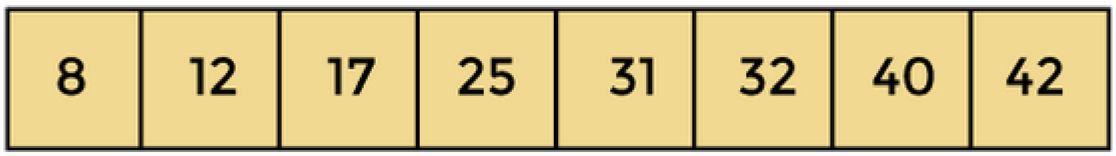
- Now, combine them in the same manner they were broken.
- In combining, first compare the element of each array and then combine them into another array in sorted order.
- So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.



- In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.



- Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -



Concept of OpenMP

- OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared-memory parallel programming in C, C++, and Fortran. It is used to write parallel programs that can run on multicore processors, multiprocessor systems, and parallel computing clusters.
- OpenMP provides a set of directives and functions that can be inserted into the source code of a program to parallelize its execution. These directives are simple and easy to use, and they can be applied to loops, sections, functions, and other program constructs. The compiler then generates parallel code that can run on multiple processors concurrently.
- OpenMP programs are designed to take advantage of the shared-memory architecture of modern processors, where multiple processor cores can access the same memory. OpenMP uses a fork-join model of parallel execution, where a master thread forks multiple worker threads to execute a parallel region of the code, and then waits for all threads to complete before continuing with the sequential part of the code.

How Parallel Merge Sort Work

- Parallel merge sort is a parallelized version of the merge sort algorithm that takes advantage of multiple processors or cores to improve its performance. In parallel merge sort, the input array is divided into smaller subarrays, which are sorted in parallel using multiple processors or cores. The sorted subarrays are then merged together in parallel to produce the final sorted output.
- The parallel merge sort algorithm can be broken down into the following steps:

- Divide the input array into smaller subarrays.
- Assign each subarray to a separate processor or core for sorting.
- Sort each subarray in parallel using the merge sort algorithm.
- Merge the sorted subarrays together in parallel to produce the final sorted output.
- The merging step in parallel merge sort is performed in a similar way to the merging step in the sequential merge sort algorithm. However, because the subarrays are sorted in parallel, the merging step can also be performed in parallel using multiple processors or cores. This can significantly reduce the time required to merge the sorted subarrays and produce the final output.
- Parallel merge sort can provide significant performance benefits for large input arrays with many elements, especially when running on hardware with multiple processors or cores. However, it also requires additional overhead to manage the parallelization, and may not always provide performance improvements for smaller input sizes or when run on hardware with limited parallel processing capabilities.

How to measure the performance of sequential and parallel algorithms?

There are several metrics that can be used to measure the performance of sequential and parallel merge sort algorithms:

1. **Execution time:** Execution time is the amount of time it takes for the algorithm to complete its sorting operation. This metric can be used to compare the speed of sequential and parallel merge sort algorithms.
2. **Speedup**: Speedup is the ratio of the execution time of the sequential merge sort algorithm to the execution time of the parallel merge sort algorithm. A speedup of greater than 1 indicates that the parallel algorithm is faster than the sequential algorithm.
3. **Efficiency:** Efficiency is the ratio of the speedup to the number of processors or cores used in the parallel algorithm. This metric can be used to determine how well the parallel algorithm is utilizing the available resources.
4. **Scalability**: Scalability is the ability of the algorithm to maintain its performance as the input size and number of processors or cores increase. A scalable algorithm will maintain a consistent speedup and efficiency as more resources are added.

To measure the performance of sequential and parallel merge sort algorithms, you can perform experiments on different input sizes and numbers of processors or cores. By measuring the execution time, speedup, efficiency, and scalability of the algorithms under different conditions, you can determine which algorithm is more efficient for different input sizes and hardware configurations. Additionally, you can use profiling tools to analyze the performance of the algorithms and identify areas for optimization

Conclusion- In this way we can implement Merge Sort in parallel way using OpenMP also come to know how to how to measure performance of serial and parallel algorithm

Assignment Question

1. **What is parallel Merge Sort?**
2. **How does Parallel Merge Sort work?**
3. **How do you implement Parallel MergeSort using OpenMP?**
4. **What are the advantages of Parallel MergeSort?**
5. **Difference between serial Mergesort and parallel Mergesort**

Reference link

- https://www.geeksforgeeks.org/merge-sort/
- https://www.javatpoint.com/merge-sort

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :......................   Date of Completion:.......................

Group A

Assignment No: 3

Title of the Assignment: Implement Min, Max, Sum and Average operations using Parallel Reduction.

Objective of the Assignment: To understand the concept of parallel reduction and how it can be used to perform basic mathematical operations on given data sets.

Prerequisite:

1. Parallel computing architectures
2. Parallel programming models
3. Proficiency in programming languages

Contents for Theory:

1. What is parallel reduction and its usefulness for mathematical operations on large data?
2. Concept of OpenMP
3. How do parallel reduction algorithms for Min, Max, Sum, and Average work, and what are their advantages and limitations?

Parallel Reduction.

Here's a **function-wise manual** on how to understand and run the sample C++ program that demonstrates how to implement Min, Max, Sum, and Average operations using parallel reduction.

1. **Min_Reduction function**
    - The function takes in a vector of integers as input and finds the minimum value in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "min" operator to find the minimum value across all threads.
    - The minimum value found by each thread is reduced to the overall minimum value of the entire array.
    - The final minimum value is printed to the console.

2. **Max_Reduction function**
    - The function takes in a vector of integers as input and finds the maximum value in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "max" operator to find the maximum value across all threads.
    - The maximum value found by each thread is reduced to the overall maximum value of the entire array.
    - The final maximum value is printed to the console.

3. **Sum_Reduction function**
    - The function takes in a vector of integers as input and finds the sum of all the values in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.
    - The sum found by each thread is reduced to the overall sum of the entire array.
    - The final sum is printed to the console.

4. **Average_Reduction function**
    - The function takes in a vector of integers as input and finds the average of all the values in the vector using parallel reduction.
    - The OpenMP reduction clause is used with the "+" operator to find the sum across all threads.

- The sum found by each thread is reduced to the overall sum of the entire array.
- The final sum is divided by the size of the array to find the average.
- The final average value is printed to the console.

5. **Main Function**
   - The function initializes a vector of integers with some values.
   - The function calls the min_reduction, max_reduction, sum_reduction, and average_reduction functions on the input vector to find the corresponding values.
   - The final minimum, maximum, sum, and average values are printed to the console.

6. **Compiling and running the program**

Compile the program: You need to use a C++ compiler that supports OpenMP, such as g++ or clang. Open a terminal and navigate to the directory where your program is saved. Then, compile the program using the following command:

```
$ g++ -fopenmp program.cpp -o program
```

This command compiles your program and creates an executable file named "program". The "-fopenmp" flag tells the compiler to enable OpenMP.

Run the program: To run the program, simply type the name of the executable file in the terminal and press Enter:

```
$ ./program
```

Conclusion: We have implemented the Min, Max, Sum, and Average operations using parallel reduction in C++ with OpenMP. Parallel reduction is a powerful technique that allows us to perform these operations on large arrays more efficiently by dividing the work among multiple threads running in parallel. We presented a code example that demonstrates the implementation of these operations using parallel reduction in C++ with OpenMP. We also provided a manual for running OpenMP programs on the Ubuntu platform.

Assignment Question

1. What are the benefits of using parallel reduction for basic operations on large arrays?
2. How does OpenMP's "reduction" clause work in parallel reduction?
3. How do you set up a C++ program for parallel computation with OpenMP?
4. What are the performance characteristics of parallel reduction, and how do they vary based on input size?
5. How can you modify the provided code example for more complex operations using parallel reduction?

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :...................... Date of Completion:.......................

Group A

Assignment 4(A)

Title of the Assignment: Write a CUDA Program for Addition of two large vectors

Objective of the Assignment: Students should be able to perform CUDA Program for Addition of two large vectors

Prerequisite:

1. CUDA Concept
2. Vector Addition
3. How to execute Program in CUDA Environment

Contents for Theory:

1. **What is CUDA**
2. **Addition of two large Vector**
3. **Execution of CUDA Environment**

What is CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use the power of NVIDIA graphics processing units (GPUs) to accelerate computation tasks in various applications, including scientific computing, machine learning, and computer vision.CUDA provides a set of programming APIs, libraries, and tools that enable developers to write and execute parallel code on NVIDIA GPUs. It supports popular programming languages like C, C++, and Python, and provides a simple programming model that abstracts away much of the low-level details of GPU architecture.

Using CUDA, developers can exploit the massive parallelism and high computational power of GPUs to accelerate computationally intensive tasks, such as matrix operations, image processing, and deep learning. CUDA has become an important tool for scientific research and is widely used in fields like physics, chemistry, biology, and engineering.

Steps for Addition of two large vectors using CUDA

1. Define the size of the vectors: In this step, you need to define the size of the vectors that you want to add. This will determine the number of threads and blocks you will need to use to parallelize the addition operation.
2. Allocate memory on the host: In this step, you need to allocate memory on the host for the two vectors that you want to add and for the result vector. You can use the C malloc function to allocate memory.
3. Initialize the vectors: In this step, you need to initialize the two vectors that you want to add on the host. You can use a loop to fill the vectors with data.
4. Allocate memory on the device: In this step, you need to allocate memory on the device for the two vectors that you want to add and for the result vector. You can use the CUDA function cudaMalloc to allocate memory.
5. Copy the input vectors from host to device: In this step, you need to copy the two input vectors from the host to the device memory. You can use the CUDA function cudaMemcpy to copy the vectors.
6. Launch the kernel: In this step, you need to launch the CUDA kernel that will perform the addition operation. The kernel will be executed by multiple threads in parallel. You can use the <<<...>>> syntax to specify the number of blocks and threads to use.
7. Copy the result vector from device to host: In this step, you need to copy the result vector from the device memory to the host memory. You can use the CUDA function cudaMemcpy to copy the result vector.

8. Free memory on the device: In this step, you need to free the memory that was allocated on the device. You can use the CUDA function cudaFree to free the memory.
9. Free memory on the host: In this step, you need to free the memory that was allocated on the host. You can use the C free function to free the memory.

Execution of Program over CUDA Environment

Here are the steps to run a CUDA program for adding two large vectors:

1. Install CUDA Toolkit: First, you need to install the CUDA Toolkit on your system. You can download the CUDA Toolkit from the NVIDIA website and follow the installation instructions provided.
2. Set up CUDA environment: Once the CUDA Toolkit is installed, you need to set up the CUDA environment on your system. This involves setting the PATH and LD_LIBRARY_PATH environment variables to the appropriate directories.
3. Write the CUDA program: You need to write a CUDA program that performs the addition of two large vectors. You can use a text editor to write the program and save it with a .cu extension.
4. Compile the CUDA program: You need to compile the CUDA program using the nvcc compiler that comes with the CUDA Toolkit. The command to compile the program is:

```
nvcc -o program_name program_name.cu
```

5. This will generate an executable program named program_name.

Run the CUDA program: Finally, you can run the CUDA program by executing the executable file generated in the previous step. The command to run the program is:

```
./program_name
```

This will execute the program and perform the addition of two large vectors.

Questions:

1. What is the purpose of using CUDA to perform addition of two large vectors?
2. How do you allocate memory for the vectors on the device using CUDA?
3. How do you launch the CUDA kernel to perform the addition of two large vectors?
4. How can you optimize the performance of the CUDA program for adding two large vectors

Answers	Coding Efficiency	Viva	Timely Completion	Total	Dated Sign of Subject Teacher
5	5	5	5	20	

Start Date :..................... Date of Completion:......................

Group A

Assignment 4(B)

Title of the Assignment: Write a Program for Matrix Multiplication using CUDA C

Objective of the Assignment: Students should be able to performProgram for Matrix Multiplication using CUDA C

Prerequisite:

1. CUDA Concept
2. Matrix Multiplication
3. How to execute Program in CUDA Environment

Contents for Theory:

1. **What is CUDA**
2. **Matrix Multiplication**
3. **Execution of CUDA Environment**

What is CUDA

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It allows developers to use the power of NVIDIA graphics processing units (GPUs) to accelerate computation tasks in various applications, including scientific computing, machine learning, and computer vision.CUDA provides a set of programming APIs, libraries, and tools that enable developers to write and execute parallel code on NVIDIA GPUs. It supports popular programming languages like C, C++, and Python, and provides a simple programming model that abstracts away much of the low-level details of GPU architecture.

Using CUDA, developers can exploit the massive parallelism and high computational power of GPUs to accelerate computationally intensive tasks, such as matrix operations, image processing, and deep learning. CUDA has become an important tool for scientific research and is widely used in fields like physics, chemistry, biology, and engineering.

Steps for Matrix Multiplication using CUDA

Here are the steps for implementing matrix multiplication using CUDA C:

1. Matrix Initialization: The first step is to initialize the matrices that you want to multiply. You can use standard C or CUDA functions to allocate memory for the matrices and initialize their values. The matrices are usually represented as 2D arrays.
2. Memory Allocation: The next step is to allocate memory on the host and the device for the matrices. You can use the standard C malloc function to allocate memory on the host and the CUDA function cudaMalloc() to allocate memory on the device.
3. Data Transfer: The third step is to transfer data between the host and the device. You can use the CUDA function cudaMemcpy() to transfer data from the host to the device or vice versa.
4. Kernel Launch: The fourth step is to launch the CUDA kernel that will perform the matrix multiplication on the device. You can use the <<<...>>> syntax to specify the number of blocks and threads to use. Each thread in the kernel will compute one element of the output matrix.
5. Device Synchronization: The fifth step is to synchronize the device to ensure that all kernel executions have completed before proceeding. You can use the CUDA function cudaDeviceSynchronize() to synchronize the device.
6. Data Retrieval: The sixth step is to retrieve the result of the computation from the device to the host. You can use the CUDA function cudaMemcpy() to transfer data from the device to the host.
7. Memory Deallocation: The final step is to deallocate the memory that was allocated on the host and the device. You can use the C free function to deallocate memory on the host and the CUDA function

cudaFree() to deallocate memory on the device.

Execution of Program over CUDA Environment

1. Install CUDA Toolkit: First, you need to install the CUDA Toolkit on your system. You can download the CUDA Toolkit from the NVIDIA website and follow the installation instructions provided.
2. Set up CUDA environment: Once the CUDA Toolkit is installed, you need to set up the CUDA environment on your system. This involves setting the PATH and LD_LIBRARY_PATH environment variables to the appropriate directories.
3. Write the CUDA program: You need to write a CUDA program that performs the addition of two large vectors. You can use a text editor to write the program and save it with a .cu extension.
4. Compile the CUDA program: You need to compile the CUDA program using the nvcc compiler that comes with the CUDA Toolkit. The command to compile the program is:

```
nvcc -o program_name program_name.cu
```

5. This will generate an executable program named program_name.

Run the CUDA program: Finally, you can run the CUDA program by executing the executable file generated in the previous step. The command to run the program is:

```
./program_name
```

Questions:

1. What are the advantages of using CUDA to perform matrix multiplication compared to using a CPU?
2. How do you handle matrices that are too large to fit in GPU memory in CUDA matrix multiplication?
3. How do you optimize the performance of the CUDA program for matrix multiplication?
4. How do you ensure correctness of the CUDA program for matrix multiplication and verify the results?