G H Raisoni College of Engineering and Management, Wagholi, Pune

Department of AI & AIML Engineering

LAB MANUAL

Subject: DEEP LEARNING (UCAMP303)

Class: TY (AIML A – 2020 Course)

Examination Scheme:
Practical (INT): 25 Marks
Total: 25 Marks

Prepared By: DR. VAISHALI YOGESH BAVISKAR

Institute Vision & Mission

Vision

To achieve excellent standards of quality education by keeping pace with rapidly changing technologies and to create technical manpower of global standards with capabilities of accepting new challenges

Mission

Our efforts are dedicated to impart quality and value based education to raise satisfaction level of all stakeholders. Our strength is directed to create competent professionals. Our endeavor is to provide all possible support to promote research and development activities.

Department Vision & Mission

VISION:

To develop globally competent professionals with a strong foundation in Artificial Intelligence and Machine Learning, empowering them to drive innovation and positively impact industry and society.

MISSION:

The department continuously strives to:

M1: Provide a comprehensive, cutting-edge curriculum that keeps pace with the rapid advancements in AI and ML technologies by collaborating with leading companies.

M2: Nurture a diverse, globally-minded student community by providing practical, experiential learning opportunities for students to apply their knowledge and skills to solve real-world problems.

M3: Create a culture that promotes continuous learning, research, and entrepreneurial skills in order to give our students a lifelong passion for staying abreast of new developments

LIST OF EXPERIMENTS

Sr. No.	Title of Experiment	CO
1	Tools and Libraries used for Deep Learning	CO1
2	Implementation of Artificial Neural Network for XOR Logic Gate with 2-bit Binary Input	CO2
3	Implementation of Artificial Neural Network for binary classification – Customer churn prediction	CO2
4	Handwritten digit recognition using Multilayer Perceptron model	CO2
5	Linear Regression by using Deep Neural Network	CO2
6	Binary classification using Deep Neural Network: Classify movie reviews into positive reviews and negative reviews based on the content of the reviews. Use IMDB dataset.	CO3
7	Write a program Logistic Regression with Neural Network mindset	CO3
8	Implement Planner data classification with one hidden layer	CO3
9	Implement Neural Network with one hidden layer	CO3
10	Build a deep neural network step by step	CO3
11	Implement the concept of regularization, gradient checking and optimization in convolutional model: step by step	CO4
	Virtual Lab Assignments	
12	Linear Perceptron Learning https://portal.coepvlab.ac.in/vlab/auth/home	CO4
13	Backpropogation algorithm to train a neural network https://vlab.spit.ac.in/ai/#/experiments/1	CO4
14	Handwritten Digit Recognition using CNN https://vlab.spit.ac.in/ai/#/experiments/2	CO5
15	Implement a car detection model using YOLO https://vlab.spit.ac.in/ai/#/experiments/6	CO5
16	Visualizing auto encoders https://vlab.spit.ac.in/ai/#/experiments/13	CO5
	Content Beyond Syllabus	
17	Implement a deep learning application using Generative AI	CO5

ASSIGNMENT NO. 1

TITLE

Tools and Libraries used for Deep Learning

OBJECTIVE

- To understand various tools used for deep learning implementation
- To understand and learn various libraries used for deep learning

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools: Python (with necessary libraries such as TensorFlow, Keras, PyTorch, MxNet, Theano, CNTK), Jupyter Notebook or any IDE for Python development.

THEORY

The various deep learning libraries are:

1. TensorFlow

TensorFlow is an open-source, cost-free software library for machine learning and one of the most popular deep learning frameworks. It is coded almost entirely using Python. Developed by Google, it is specifically optimized for the inference and training of deep learning models. Deep learning inference is the process by which trained deep neural network models are used to make predictions about previously untested data. TensorFlow allows developers to produce large-scale neural networks with many layers using data flow graphs. Hence, Tensorflow supports these large numerical computations by accepting data in the form of multidimensional arrays that host generalized vectors and matrices, called tensors.

2. Keras

Keras is another very popular open-source deep learning library. The framework provides a Python interface for developing artificial neural networks. Keras acts as an interface for the TensorFlow library. It has been accredited as an easy-to-use, simplistic interface. Keras is particularly useful because it can scale to large clusters of GPUs or entire TPU pods. Also, the functional API can handle models with non-linear topology, shared layers, and even multiple inputs or outputs. Keras was

developed to enable fast experimentation for bulky models while prioritizing the developer experience, which is why the platform is so intuitive.

3. PyTorch

PyTorch is a Python library that supports building deep learning projects such as computer vision and natural language processing. It has two main features: Tensor computing (such as NumPy) with strong acceleration via GPU and deep neural networks built on top of a tape-based automatic differentiation system, which numerically evaluates the derivative of a function specified by a computer program. PyTorch also includes the Optim and nn modules. The Optim module, *torch.optim*, uses different optimization algorithms used in neural networks. The nn module, or PyTorch autograd, lets you define computational graphs and also make gradients; this is useful because raw autograd can be low-level. Pytorch's advantages over other deep learning frameworks include a short learning curve and data parallelism, where computational work is distributed among multiple CPU or GPU cores.

4. MxNet

Apache MxNet is an open-source deep learning framework designed to train and deploy deep neural networks. A distinguishing feature of MxNet, when compared to other frameworks, is its scalability (the measure of a system's ability to increase or decrease in performance. MxNet is also especially known for its ability to be flexible with multiple languages, unlike frameworks like Keras that only support one language. The supported languages include C++, Python, Julia, Matlab, JavaScript, Go, R, and many more. However, it is not as widely used as other DL frameworks, which leaves it with a smaller open-source community.

5. Theano

Theano is a powerful deep learning tool that allows for efficient manipulation and evaluation of mathematical expressions, especially matrix-valued ones. It is an open-source project developed by the Montreal Institute for Learning Algorithms (MILA) at the Université de Montréal and is written in Python with a NumPy-like syntax. Theano can compile computations to run efficiently on either CPU or GPU architectures. In 2017, major development of Theano ceased due to competition from other strong industrial players, but it was taken over by the PyMC development team and renamed Aesara. Theano is used for various deep learning tasks, including image classification, natural language processing (NLP), and speech recognition. It supports a range of deep learning architectures, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Autoencoders. Theano also includes tools for visualization and debugging, making it easier for

users to design and validate their models. One of the main features of Theano is its ability to optimize the computation of mathematical expressions. Theano can automatically optimize the use of available hardware resources, including CPU and GPU, to speed up computations. Theano also includes features for controlling the use of memory, making it possible to handle large datasets and models.

6. CNTK

Microsoft Cognitive Toolkit (CNTK) is an open-source deep-learning toolkit for building, training, and evaluating neural networks. It allows for the combination of popular model types, such as feed-forward DNNs, CNNs, and RNNs/LSTMs, and uses SGD learning with automatic differentiation and parallelization across multiple GPUs and servers. It was released under an open-source license in April 2015. CNTK supports multiple languages, including C++, Python, and BrainScript, a custom language designed for building and describing neural networks. It is designed to scale efficiently in a multi-GPU and multi-machine environment, ideal for large-scale training and model deployment. The DL tool suite has been used in several industry applications, including speech recognition, image recognition, and natural language processing (NLP). It also offers support for several popular deep learning frameworks, including TensorFlow, Keras, and PyTorch, allowing for easy integration with other deep learning tools. In addition, CNTK includes some pre-trained models and tutorials to help users get started with their deep learning projects.

CONCLUSION

Thus, we have studied various tools and libraries required to build deep learning applications.

ASSIGNMENT NO. 2

TITLE

Implementation of Artificial Neural Network for XOR Logic Gate with 2-bit Binary Input

OBJECTIVE

- To understand the application of Artificial Neural Networks (ANNs) in solving non-linear problems.
- To implement a simple ANN with one hidden layer to model the XOR logic gate.
- To explore the forward propagation, back propagation, and gradient descent processes in neural networks.
- To analyze how an ANN can learn to approximate the XOR function through training.

THEORY

XOR logical function truth table for ***2-bit binary variables***, i.e, the input vector and the corresponding output y.

$x : (x_1, x_2)$

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

Challenges of XOR Problem:

The XOR function is not linearly separable, meaning it cannot be separated by a single linear decision boundary in the input space. This characteristic makes it impossible to solve using a simple linear classifier like a single-layer perceptron.

Artificial Neural Networks (ANN)

- An ANN is a computational model inspired by the way biological neural networks in the human brain process information. It consists of interconnected layers of nodes (neurons) where each connection is associated with a weight. The layers typically include an input layer, one or more hidden layers, and an output layer.

  Input Layer: Receives the input signals.

  Hidden Layer: Processes the inputs by applying weights, biases, and activation functions to produce outputs that are passed to the next layer.

  Output Layer: Produces the final output of the network.

  Activation Function: To solve the XOR problem, non-linear activation functions like the sigmoid function are used. The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- This function maps any real-valued number into the (0, 1) interval, enabling the network to capture non-linear relationships.

  Backpropagation Algorithm: Backpropagation is a supervised learning algorithm used for training ANNs. It involves two phases:

  Forward Propagation: The input data passes through the network layer by layer, producing an output.

  Backward Propagation: The error between the predicted and actual output is propagated back through the network, updating the weights and biases using gradient descent to minimize the loss.

  Training the Network: The network is trained by repeatedly feeding the input data through the network, calculating the error, and adjusting the weights and biases until the output closely matches the expected results.

  Solving the XOR Problem: By utilizing a hidden layer with non-linear activation functions, an ANN can learn to model the XOR function. During training, the network adjusts its weights to minimize the error in predicting the XOR outputs, even though the problem is non-linear and not linearly separable.

ALGORITHM

1. Import the required Python libraries
2. Define Activation Function : Sigmoid Function
3. Initialize neural network parameters (weights, bias).
4. Forward Propagation
5. Backward Propagation
6. Update weight and bias parameters
7. Train the learning model
8. Plot Loss value vs Epoch
9. Test the model performance

CONCLUSION

Here, the model predicted output for each of the test inputs are exactly matched with the XOR logic gate conventional output y according to the truth table and the cost function is also continuously converging.

Hence, it signifies that the Artificial Neural Network for the XOR logic gate is correctly implemented.

ASSIGNMENT NO. 3

TITLE

Implementation of Artificial Neural Network for binary classification – Customer churn prediction

OBJECTIVE

- To understand and implement an Artificial Neural Network (ANN) for solving a binary classification problem, specifically customer churn prediction.
- To gain practical experience in feature engineering, data preprocessing, and ANN model training using a real-world dataset.
- To evaluate the performance of the trained ANN model using metrics like accuracy and loss.
- To apply the trained ANN model for single-point prediction and analyze its output.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools:

- **Python:** A programming language used to implement and manage deep learning libraries.
- **Deep Learning Libraries:**
  - TensorFlow
  - Keras
  - PyTorch
  - MxNet
  - Theano (or Aesara, its successor)
  - CNTK
- **Development Environment:**
  - Jupyter Notebook (for interactive coding and visualization)

- Any Python Integrated Development Environment (IDE) such as PyCharm, VS Code, or Spyder.

THEORY

Artificial Neural Network (ANN)

An ANN is a computational model inspired by the human brain's network of neurons. It consists of interconnected nodes (neurons) arranged in layers: an input layer, one or more hidden layers, and an output layer. Each connection between neurons has a weight that adjusts during training to minimize the prediction error.

Binary Classification

In binary classification, the task is to categorize data points into one of two classes. For customer churn prediction, the model predicts whether a customer will leave the bank (churn) or stay.

Dataset Description

We are working with a dataset from the finance domain with 14 dimensions (features) and 100,000 records. The dataset includes features like CreditScore, Geography, Gender, Age, Tenure, Balance, NumOfProducts, HasCrCard, IsActiveMember, and EstimatedSalary, among others. The target variable Exited indicates whether a customer will exit the bank or not.

Feature Engineering and Preprocessing

Feature engineering involves preparing the data for model training. This includes encoding categorical variables (like Gender and Geography), splitting the data into training and testing sets, and performing feature scaling to normalize the data values.

Model Architecture:

- **Input Layer:** Consists of neurons equal to the number of features (after encoding and scaling).
- **Hidden Layers:** Two hidden layers with 6 neurons each, using the ReLU activation function.

- **Output Layer:** One neuron using the sigmoid activation function to output the probability of a customer churning.

Loss Function and Optimization:

- **Loss Function:** Binary Cross-Entropy is used to measure the difference between the predicted and actual labels.
- **Optimizer:** The Adam optimizer is used for adjusting the weights during training, ensuring the model converges efficiently.

Evaluation Metrics:

Accuracy is the primary metric used to evaluate the performance of the model. The final model's accuracy and loss are used to determine its effectiveness.

ALGORITHM

1. **Data Preprocessing**
    - Load the dataset and extract features (X) and the target variable (Y).
    - Encode categorical variables using label encoding and one-hot encoding.
    - Split the dataset into training and testing sets.
    - Apply feature scaling to normalize the input data.
2. **Model Initialization**
    - Initialize the ANN model using the Sequential class in Keras.
3. **Building the ANN**
    - Add the first hidden layer with a specified number of neurons and ReLU activation.
    - Add additional hidden layers as needed.
    - Add the output layer with a single neuron and sigmoid activation for binary classification.
4. **Compiling the Model**
    - Compile the ANN using an optimizer (e.g., Adam), loss function (binary cross-entropy), and evaluation metric (accuracy).
5. **Training the Model**

- Train the ANN on the training dataset using the fit method, specifying batch size and the number of epochs.

6. **Evaluating the Model**
    - Evaluate the model's performance on the test dataset.
    - Use the trained model to predict the outcome for a new input instance.

CONCLUSION

Thus, we have implemented ANN model which is an effective tool for binary classification tasks such as customer churn prediction, where it learns complex patterns from the data to make accurate predictions.

ASSIGNMENT NO. 4

TITLE

Handwritten digit recognition using Multilayer Perceptron model

OBJECTIVE

- Gain familiarity with the MNIST dataset, including its structure, dimensions, and usage in deep learning.
- Learn to preprocess the MNIST dataset, including reshaping images, normalizing pixel values, and converting labels to one-hot encoded format
- Implement a Multilayer Perceptron (MLP) using TensorFlow and Keras to classify handwritten digits.
- Use dropout to prevent overfitting and ensure the model generalizes well to new data.
- Assess the performance of the trained model using accuracy metrics on the test dataset.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools

Python: A programming language used for implementing and managing the deep learning model.

Deep Learning Libraries:

- TensorFlow
- Keras (integrated with TensorFlow)

Development Environment:

- Jupyter Notebook (for interactive coding and visualization)
- Any Python Integrated Development Environment (IDE) such as PyCharm, VS Code, or Spyder

THEORY

MNIST Dataset Overview

The MNIST dataset is a benchmark dataset in the field of machine learning and deep learning. It contains 70,000 grayscale images of handwritten digits (0-9), each of size 28x28 pixels. The dataset is divided into 60,000 training images and 10,000 test images. The goal is to build a model that can accurately classify these images into the correct digit class.

Multilayer Perceptron (MLP)

An MLP is a type of artificial neural network that consists of multiple layers of neurons. Each neuron in one layer is connected to every neuron in the next layer. The MLP used in this experiment consists of:

- **Input Layer:** Takes the flattened 28x28 pixel images as input.
- **Hidden Layers:** Two fully connected (Dense) layers with ReLU activation and dropout for regularization.
- **Output Layer:** A fully connected layer with a softmax activation function to output probabilities for each digit class.

- **Activation Functions:** ReLU (Rectified Linear Unit) introduces non-linearity into the model, allowing it to learn complex patterns. Softmax is used in the output layer to convert the logits into a probability distribution across 10 classes.
- **Dropout:** A regularization technique where a fraction of neurons is randomly dropped during training to prevent overfitting and improve model generalization.
- **Loss Function:** Categorical cross-entropy is used for training, which measures the difference between the true labels and the predicted probabilities.
- **Optimizer:** Adam is used as the optimization algorithm, which adjusts the learning rate during training to efficiently minimize the loss.

ALGORITHM

1. **Data Loading**

   - Load the MNIST dataset using TensorFlow's mnist.load_data() function.

2. **Data Preprocessing**

- Reshape the training and test images from 28x28 pixels to a 1D vector of 784 pixels.
- Normalize the pixel values to a range of [0, 1] by dividing by 255.
- Convert the labels to one-hot encoded vectors.

3. **Model Building**

- Initialize a Sequential model.
- Add the first Dense layer with 256 neurons, ReLU activation, and Dropout (0.45).
- Add a second Dense layer with 256 neurons, ReLU activation, and Dropout (0.45).
- Add the output Dense layer with 10 neurons and softmax activation.

4. **Model Compilation**

- Compile the model using the Adam optimizer, categorical cross-entropy as the loss function, and accuracy as the evaluation metric.

5. **Model Training**

- Train the model on the training data for 20 epochs with a batch size of 128.

6. **Model Evaluation**

- Evaluate the model on the test dataset to determine its accuracy.

7. **Single-Point Prediction**

- Optionally, make a prediction on a single test image to verify the model's output.

CONCLUSION

The experiment successfully demonstrated the implementation of a Multilayer Perceptron (MLP) for handwritten digit classification using the MNIST dataset. The model was trained using two hidden layers with ReLU activation and Dropout for regularization. The final model achieved a high

accuracy. on the test dataset, confirming that it learned to generalize well to unseen data. This experiment highlighted the importance of data preprocessing, model architecture design, and regularization techniques in building effective deep learning models.

ASSIGNMENT NO. 5

TITLE

Linear regression by using Deep Neural network

OBJECTIVE

- To predict the median value of owner-occupied homes in Boston using a combination of Linear Regression and Deep Neural Networks (DNN).
- To explore and understand the relationship between various features (such as crime rate, average number of rooms, property tax rate) and the housing prices.
- To implement both Linear Regression and a Deep Neural Network model to compare their effectiveness in predicting housing prices.
- To optimize the models for better prediction accuracy using techniques like feature scaling, normalization, and hyperparameter tuning.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools:

Python: A programming language used for implementing and managing the models.

Deep Learning Libraries:

- TensorFlow
- Keras (integrated with TensorFlow)

Development Environment:

- Jupyter Notebook (for interactive coding and visualization)
- Any Python Integrated Development Environment (IDE) such as PyCharm, VS Code, or Spyder

Additional Tools:

- Scikit-learn (for handling the dataset and performing linear regression)

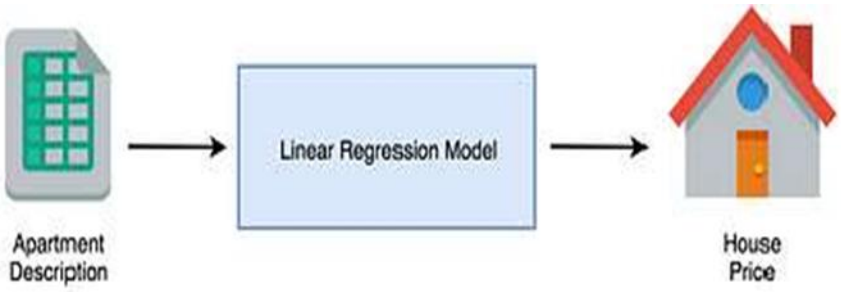
- Pandas (for data manipulation)
- NumPy (for numerical operations)

THEORY

Linear Regression is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used. There are many names for a regression's dependent variable. It may be called an outcome variable, criterion variable, endogenous variable, or regressand. The independent variables can be called exogenous variables, predictor variables, or regressors. Linear regression is used in many different fields, including finance, economics, and psychology, to understand and predict the behavior of a particular variable. For example, in finance, linear regression might be used to understand the relationship between a company's stock price and its earnings, or to predict the future value of a currency based on its past performance.

It's a **Supervised Learning algorithm** which goal is to **predict continuous, numerical values based on given data input**. From the geometrical perspective, each data sample is a point. Linear Regression tries to **find parameters of the linear function**, so the **distance between the all the points and the line is as small as possible**. Algorithm used for parameters update is called **Gradient Descent**.

For example, if we have a dataset consisting of apartments properties and their prices in some specific area, Linear Regression algorithm can be used to find a mathematical function which will try to estimate the value of different apartment (outside of the dataset), based on its attributes.



Deep Neural Network

A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real wo ld problems like classification. We restrict ourselves to feed forward neural networks. Neural networks are widely used in supervised learning and reinforcement learning problems. These networks are based on a set of layers connected to each other. In deep learning, the number of hidden layers, mostly non-linear, can be large; say about 1000 layers. DL models produce much better results than normal ML networks. A deep neural network is beneficial when you need to replace human labor with autonomous work without compromising its efficiency. The deep neural network usage can find various applications in real life. The American company Pony.ai is another example of how you can use DNN. They developed a system for AI cars that can work without a driver. It requires more than just a simple algorithm of actions, but a much deeper learning system, which should be able to recognize people, road signs and other markings like trees, and other important objects.

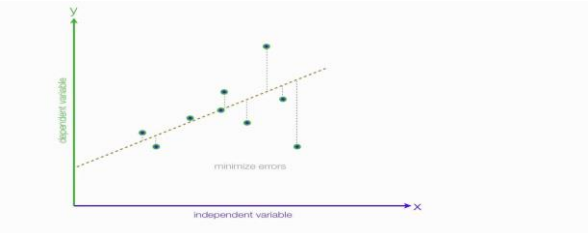
The famous company UbiTech creates AI robots. One of their creations is the Alpha 2 robot that can live in a family, speak with its members, search for information, write messages, and execute voice commands The neural network needs to learn all the time to solve tasks in a more qualified manner or even to use various methods to provide a better result. When it gets new information in the system, it learns how to act accordingly to a new situation. Learning becomes deeper when tasks you solve get harder. Deep neural network represents the type of machine learning when the system uses many layers of nodes to derive high-level functions from input information. It means transforming the data into a more creative and abstract component.

Visualizing data

It is very important to always understand the structure of data. The more features there are, the harder it is. In this case, scatter plot is used to **display the relationship between target and training features**. Depending on what is necessary to show, some other types of visualization (e.g. box plot) and techniques could be useful (e.g. clustering). Here, a **linear dependency between features can be observed** — with the increase of values on axis x, values on the y-axis are linearly increasing or decreasing accordingly. It's great because if that was not the case (e.g. relationship would be exponential), then it would be hard to fit a line through all the points and different algorithm should be considered.

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.

The objective of Linear Regression is to find a line that minimizes the prediction error of all the data points.

- The Mean absolute error represents the average of the absolute difference between the actual and predicted values in the dataset. It measures the average of the residuals in the dataset.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

Where,

$\hat{y}$ — *predicted value of y*

$\bar{y}$ — *mean value of y*

Mean Squared Error represents the average of the squared difference between the original and predicted values in the data set. It measures the variance of the residuals

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

Root Mean Squared Error is the square root of Mean Squared error. It measures the standard deviation of residuals.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

The coefficient of determination or R-squared represents the proportion of the variance in the dependent variable which is xplained by the linear regression model. It is a scale-free score i.e. irrespective of the values being small or large, the value of R square will be less than one.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Boston Dataset :

The Boston housing dataset contains information about various houses in boston thorugh different parameters. Each record in the database describes a Boston town. The data was drawn from the

Boston Standard Metropolitan Statistical Area (SMSA) in 1970. The data was originally a part of UCI Machine Learning Repository . There are 506 samples and 13 feature variables in this dataset.

ALGORITHM

1. Data Preprocessing

- Load the Boston Housing dataset.
- Handle missing values and outliers.
- Normalize the features to have a mean of 0 and a standard deviation of 1, which helps in improving the convergence speed of the models.
- Split the dataset into training and testing sets (e.g., 80% training, 20% testing).

2. Linear Regression Model

- Initialize the Linear Regression model.
- Train the model using the training data by finding the optimal weights and bias that minimize the error.
- Evaluate the model on the testing set using metrics like Mean Squared Error (MSE) and R-squared.
- Use the trained model to make predictions on new data.

3. Deep Neural Network Model

- Define the architecture of the DNN with an input layer (number of features), multiple hidden layers (with ReLU activation), and an output layer (single neuron for regression).
- Compile the model with a suitable loss function (e.g., MSE) and an optimizer (e.g., Adam).
- Train the DNN on the training data, adjusting weights and biases to minimize the loss function.
- Evaluate the DNN on the testing set using the same metrics as Linear Regression.
- Use the trained DNN to predict new housing prices.

4. Model Comparison

- Compare the performance of the Linear Regression model and the DNN model based on their evaluation metrics.
- Analyze which model provides better prediction accuracy and generalizes well to unseen data.

CONCUSION

In this way we learn to implement Boston housing price prediction problem by Linear regression using Deep Neural network.

ASSIGNMENT NO. 6

TITLE

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

OBJECTIVE

- Explore the structure and content of the IMDB dataset, including how movie reviews are represented as sequences of integers.
- Learn how to preprocess text data by vectorizing sequences and preparing labels for input into a neural network.
- Implement a simple neural network model using Keras to classify movie reviews as positive or negative.
- Analyze the performance of the trained model using accuracy metrics on both the validation and test datasets.
- Understand the concept of overfitting and apply techniques to reduce it, such as adjusting the number of training epochs.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools:

- **Python:** A programming language used for implementing and managing the models.
- **Deep Learning Libraries:**
  - TensorFlow (with Keras)
- **Development Environment:**
  - Jupyter Notebook (for interactive coding and visualization)
  - Any Python Integrated Development Environment (IDE) such as PyCharm, VS Code, or Spyder
- **Additional Tools:**

- Scikit-learn (for data handling and preprocessing)
- Pandas (for data manipulation)
- NumPy (for numerical operations)

THEORY

IMDB Dataset Overview

The IMDB dataset is a widely used benchmark for sentiment analysis, containing 50,000 highly polarized movie reviews. The dataset is equally divided into 25,000 reviews for training and 25,000 for testing, with each set containing an equal number of positive and negative reviews. Each review is preprocessed into a sequence of integers, where each integer represents a specific word in the dictionary.

Neural Networks for Sentiment Analysis

Neural networks, particularly fully connected (Dense) layers, are well-suited for sentiment analysis tasks. In this experiment, we will use a simple neural network with two hidden layers, each containing 16 neurons. The input to the network will be a 10,000-dimensional vector representing the presence or absence of the top 10,000 most frequent words in the reviews.

- **One-Hot Encoding:** Convert sequences of integers into 10,000-dimensional vectors, where each index represents a word, and the value is 1 if the word is present in the review.

- **Activation Functions:** ReLU (Rectified Linear Unit) is used in the hidden layers to introduce non-linearity, while Sigmoid is used in the output layer to produce a probability score between 0 and 1, indicating the likelihood of a review being positive.
- **Loss Function:** Binary cross-entropy is used to measure the difference between the true labels and the predicted probabilities.
- **Optimizer:** RMSprop is used to efficiently minimize the loss during training.
- **Overfitting:** A common issue where a model performs well on training data but poorly on new, unseen data. It is identified by a divergence in training and validation performance metrics over time.

ALGORITHM

1. **Data Loading**

- Load the IMDB dataset from Keras, keeping only the top 10,000 most frequently occurring words.

2. Data Preprocessing

- Vectorize the review sequences into 10,000-dimensional vectors.
- Convert the labels to float32 format.

3. Model Building

- Initialize a Sequential model.
- Add the first Dense layer with 16 neurons, ReLU activation, and an input shape of 10,000.
- Add a second Dense layer with 16 neurons and ReLU activation.
- Add the output Dense layer with 1 neuron and Sigmoid activation for binary classification.

4. Model Compilation

- Compile the model using the RMSprop optimizer, binary cross-entropy loss function, and accuracy as the evaluation metric.

5. Validation Setup

- Set aside 10,000 samples from the training data for validation.

6. Model Training

- Train the model for 20 epochs using mini-batches of 512 samples, while monitoring performance on the validation set.

7. Evaluation and Overfitting Identification

- Plot training and validation loss and accuracy to identify overfitting.
- Reduce the number of epochs to 3 to prevent overfitting.

8. Model Retraining

- Retrain the model for 3 epochs based on the previous observations.

9. Model Evaluation

- Evaluate the final model on the test dataset and calculate the mean absolute error (MAE) to assess prediction accuracy.

CONCLUSION

The experiment successfully demonstrated the process of building, training, and evaluating a neural network for sentiment analysis of movie reviews using the IMDB dataset. The model was initially trained for 20 epochs, but signs of overfitting were observed after 3-5 epochs. To address this, the model was retrained for 3 epochs, achieving a training accuracy of 99% and a validation accuracy of 86%. The experiment highlighted the importance of monitoring validation performance to prevent overfitting and showed how a simple neural network could effectively classify the sentiment of movie reviews.

ASSIGNMENT NO. 7

TITLE

Write a program Logistic Regression with Neural Network mindset

OBJECTIVE

- Develop an intuition for logistic regression and its application in binary classification problems.
- Learn how to minimize the cost function to optimize the model.
- Understand how to update model parameters using gradient descent based on the derivatives of the cost function.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools:

Python: A programming language used for implementing the logistic regression model.

Deep Learning Libraries:

- TensorFlow (for implementing logistic regression and gradient descent)

Development Environment:

- Jupyter Notebook (for interactive coding and visualization)
- Any Python Integrated Development Environment (IDE) such as PyCharm, VS Code, or Spyder

Additional Tools:

- NumPy (for numerical operations and gradient calculations)
- Matplotlib (for visualizing results)
- Scikit-learn (for additional utilities such as data splitting and performance metrics)

THEORY

Logistic Regression as a Simple Neural Network

Logistic regression is a type of binary classifier that can be viewed as a very simple neural network with no hidden layers. It works by mapping input features to a weighted sum (plus a bias) and then applying a sigmoid function to squash the result between 0 and 1, representing the probability of the input belonging to one of the two classes.

For an input feature vector x(i) and corresponding label y(i):

Linear Combination:

$z(i) = w^T x(i) + b$

where w is the weight vector and b is the bias term.

Sigmoid Activation:

$\hat{y}(i) = a(i) = \text{sigmoid}(z(i))$

The sigmoid function maps z(i) to a probability value between 0 and 1.

Cost Function:

$L(a(i), y(i)) = -y(i) \log(a(i)) - (1 - y(i)) \log(1 - a(i))$

This loss function measures how well the predicted probability a(i) matches the true label y(i).

Overall Cost:

$J = (1/m) * \Sigma L(a(i), y(i))$

The total cost is the average of the loss over all training examples.

Gradient Descent for Optimization

Gradient descent is used to minimize the cost function J. The gradients of J with respect to the model parameters w and b are computed, and the parameters are updated iteratively:

$w := w - \alpha \partial J/\partial w$

$b := b - \alpha \partial J/\partial b$

where α is the learning rate, a hyperparameter that controls the step size in the parameter space.

ALGORITHM

The process of building the cat classifier involves the following steps:

1. **Initialize Parameters**: Initialize the weight vector www and bias bbb to zeros or small random values.
2. **Forward Propagation**: Calculate the predicted output y^\hat{y}y^ using the current parameters, compute the cost function JJJ, and derive the gradients.
3. **Backward Propagation**: Compute the gradients of the cost function with respect to the parameters www and bbb.
4. **Optimization**: Update the parameters www and bbb using gradient descent.
5. **Prediction**: Use the optimized parameters to make predictions on the training and test sets.
6. **Evaluation**: Compare the predicted labels with the actual labels to evaluate the model's performance.

CONCLUSION

In this experiment, developed a logistic regression-based cat classifier capable of achieving at least 70% accuracy. learned how to:

- Preprocess the dataset to prepare it for model training.
- Implement the core functions required for building a logistic regression model.
- Train the model using gradient descent to minimize the cost function.
- Evaluate the model's performance on unseen test data.

ASSIGNMENT NO. 8

TITLE

Implement Planner data classification with one hidden layer

OBJECTIVE

- To build a Neural Network model with a single hidden layer to classify data from the flower dataset.
- The dataset is a 2-class problem where the task is to classify data points into two distinct categories.
- The objective is to achieve a higher accuracy than what was possible with logistic regression, specifically aiming for 91% accuracy using a simple Neural Network.

PLATFORM REQUIRED

Operating System: Windows or Linux

Software/Tools:

Python: A programming language for implementing the neural network model.

Deep Learning Libraries:

- TensorFlow (for building and training the neural network model)
- Keras (for easier neural network construction and management)

Development Environment:

- Jupyter Notebook (for interactive coding and visualization)
- Any Python Integrated Development Environment (IDE) such as PyCharm, VS Code, or Spyder

Additional Tools:

- NumPy (for numerical operations)
- Matplotlib (for visualizing results and model performance)

- Scikit-learn (for utilities such as data splitting and performance metrics)

THEORY

Neural Networks are a series of algorithms that attempt to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. A Neural Network typically consists of an input layer, one or more hidden layers, and an output layer.

Input Layer (n_x): The layer that takes in the features of the input data. For instance, if the input data is a set of points in a 2D space, then the input layer would have 2 neurons.

Hidden Layer (n_h): This layer processes the inputs from the input layer. Each neuron in the hidden layer applies a weight to its inputs and passes the result through a non-linear activation function, typically the sigmoid or tanh function. This layer captures complex patterns in the data.

Output Layer (n_y): This layer generates the final predictions. In a binary classification problem, the output layer typically has a single neuron with a sigmoid activation function that outputs a probability value between 0 and 1.

The steps in building a Neural Network model include:

1. Initialization: Setting up initial weights and biases.
2. Forward Propagation: Calculating the output of the model using the current parameters.
3. Loss Calculation: Measuring how well the predictions match the actual results using a cost function.
4. Backward Propagation: Updating the weights and biases to minimize the loss.
5. Optimization: Iteratively adjusting the parameters using a method like Gradient Descent to improve the model's performance.

ALGORITHM

1. Define Neural Network Structure

- Set the number of input features (n_x), the size of the hidden layer (n_h), and the size of the output layer (n_y).

2. Initialize Parameters

- Initialize weights and biases for each layer with small random values (for weights) and zeros (for biases).

3. Forward Propagation

- Compute the following for the hidden layer:

$Z^{[1]} = W^{[1]}X + b^{[1]}$

$A^{[1]} = \tanh(Z^{[1]})$

- Compute the following for the output layer:

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

$A^{[2]} = \text{sigmoid}(Z^{[2]})$

- The output $A^{[2]}$ represents the predictions.

4. Compute Cost

- Use the cost function to calculate the error between the predicted and actual labels.

$J = -(1/m) * \text{sum}(y*\log(a^{[2]}) + (1-y)*\log(1-a^{[2]}))$

5. Backward Propagation

- Calculate the gradients with respect to each parameter.
- Update the parameters using Gradient Descent.

6. Prediction

- Use the trained model to predict the class labels of new data points.

7. Model Accuracy

- Compare the predicted labels to the actual labels to determine the model's accuracy.

CONCLUSION

The neural network model with a single hidden layer successfully trained to classify the planar dataset with an accuracy of 91%. This exercise helped in understanding the fundamental steps involved in building and training a simple neural network, including forward propagation, backpropagation, and parameter optimization.

ASSIGNMENT NO. 9

TITLE

Implement Neural Network with one hidden layer

OBJECTIVE

1. Define the neural network structure (# of input units, # of hidden units, etc).
2. Initialize the model's parameters.

PLATFORM REQUIRED

Operating System:

- **Windows**, **macOS**, or **Linux**

Software/Tools:

Python:

- A versatile programming language used for implementing machine learning and neural network models.

Deep Learning Libraries:

- **TensorFlow**: For building and training neural networks.
- **Keras**: A high-level API integrated with TensorFlow, simplifying neural network creation and training.

Development Environment:

- **Jupyter Notebook**: Ideal for interactive development, testing, and visualization.
- **Python IDE**: Such as **PyCharm**, **VS Code**, or **Spyder** for writing and managing code.

Additional Libraries:

- **NumPy**: For numerical operations and matrix manipulations.
- **Scikit-learn**: For data preprocessing, splitting, and additional utilities.
- **Matplotlib**: For plotting and visualizing data and model performance.

THEORY

Activation Functions in Neural Networks

The main objective of the activation function is to apply a mapping of a weighted sum to the output. Common activation functions include tanh, ReLU, and sigmoid, each serving different purposes.

Linear Activation Function

- The output is not restricted within a range and can vary from negative to positive infinity.
- The linear nature means that outputs are directly proportional to inputs, which limits the complexity the network can model.

Non-Linear Activation Function

- Helps the model generalize across various data inputs, enabling it to capture more complex patterns.
- Non-linear functions introduce derivatives that facilitate backpropagation.
- Common types include:

Sigmoid or Logistic Activation Function

- Output ranges from 0 to 1, making it ideal for binary classification tasks.
- Helps in normalizing neuron outputs and provides a smooth gradient, avoiding sharp changes.

Tanh or Hyperbolic Tangent Activation Function

- An improvement over the sigmoid function with an output range from -1 to 1.
- It is commonly used in hidden layers, enhancing learning efficiency.

ReLU (Rectified Linear Unit) Activation Function

- Widely used due to its range from 0 to infinity, helping in backpropagation.
- Allows sparse activation, improving computation efficiency.

Softmax Function

- A variant of the sigmoid function used in classification tasks with multiple classes.
- Normalizes the output to a probability distribution across multiple classes.

Packages Used

- numpy for fundamental scientific computing.
- sklearn for data mining and analysis tools.
- matplotlib for data visualization.
- planar_utils for various utility functions relevant to this assignment.

ALGORITHM

1. **Define Neural Network Structure**
   - Set the number of input units (`n_x`), hidden units (`n_h`), and output units (`n_y`).
2. **Initialize Parameters**
   - Randomly initialize weights and set biases to zero for the neural network.
3. **Forward Propagation**
   - Implement the forward propagation to calculate `Z[1]`, `A[1]`, `Z[2]`, and `A[2]`, which are the linear and activation values for the layers.
4. **Cost Function**
   - Compute the cost to evaluate how well the model is performing.
5. **Backward Propagation**
   - Calculate gradients and update parameters using gradient descent.
6. **Prediction**
   - Use the trained model to make predictions on new data

CONCLUSION

By studying and implementing the neural network with one hidden layer, we gained insights into how activation functions and network architecture impact learning. Using the tanh activation function, we can capture complex patterns in the data, leading to improved model accuracy.

ASSIGNMENT NO. 10

TITLE

To Build Deep Neural Network Step by Step.

OBJECTIVE

- Build a neural network with multiple hidden layers and understand the role of each layer.
- Use ReLU (Rectified Linear Unit) to introduce non-linearity into the network and enhance its ability to capture complex patterns.
- Compute the output of the network by passing input data through each layer, applying linear transformations and activation functions.
- Update network parameters (weights and biases) using gradients computed from the loss function to minimize the prediction error.
- Measure the difference between predicted and actual outputs using binary cross-entropy and adjust network parameters to minimize this loss.
- Assess the trained network's accuracy and generalization capability using a validation dataset.
- Understand the practical application of neural networks, activation functions, and optimization techniques in a real-world context.

PLATFORM REQUIRED

Operating System:

- Windows, macOS, or Linux

Software/Tools:

- Python:

A powerful programming language widely used for machine learning and neural network implementation.

Deep Learning Libraries:

- TensorFlow: A comprehensive library for building and training deep learning models.
- Keras: A high-level API for TensorFlow that simplifies the process of creating and training neural networks.

Development Environment:

- Jupyter Notebook: Ideal for interactive development, visualization, and experimentation.
- Python IDE: Such as PyCharm, VS Code, or Spyder for writing and managing code.

Additional Libraries:

- NumPy: For numerical operations and matrix manipulations.
- Matplotlib: For plotting and visualizing data and model performance.
- Scikit-learn: For additional utilities, such as data preprocessing and metrics.

THEORY

Neural Networks and Deep Learning

- **Neural Networks:** A neural network is a computational model inspired by the human brain, consisting of layers of interconnected nodes or "neurons." Each connection has an associated weight, and each neuron applies an activation function to its input.
- **Deep Neural Networks (DNNs):** DNNs are neural networks with multiple hidden layers between the input and output layers. They can capture complex patterns and representations in data through their hierarchical structure.

Non-Linear Activation Functions

- **ReLU (Rectified Linear Unit):** ReLU is a non-linear activation function defined as f(x)=max⁡(0,x)f(x) = \max(0, x)f(x)=max(0,x). It introduces non-linearity into the network, allowing it to model complex relationships.

Forward and Backward Propagation

- **Forward Propagation:** This process involves passing the input data through the network layer by layer to compute the output. Each layer performs a linear transformation followed by an activation function.
- **Backward Propagation:** This process involves updating the weights of the network using gradient descent. The gradients of the loss function with respect to the weights are computed, and weights are adjusted to minimize the loss.

Loss Function

- The loss function measures the difference between the predicted output and the true output. For binary classification, common loss functions include binary cross-entropy.

ALGORITHM

1. **Initialization**

- Initialize parameters (weights and biases) of the network randomly or using a specific initialization technique (e.g., He initialization for ReLU).

2. **Forward Propagation**
   - For each layer, compute the linear transformation Z=W·A+bZ = W \cdot A + bZ=W·A+b.
   - Apply the activation function A=ReLU(Z)A = \text{ReLU}(Z)A=ReLU(Z) (for hidden layers) or A=sigmoid(Z)A = \text{sigmoid}(Z)A=sigmoid(Z) (for output layer).
   - Compute the final output of the network.
3. **Compute Loss**
   - Calculate the loss using the chosen loss function (e.g., binary cross-entropy).
4. **Backward Propagation**
   - Compute gradients of the loss with respect to weights and biases using the chain rule.
   - Update the parameters using gradient descent or other optimization algorithms (e.g., Adam).
5. **Iteration**
   - Repeat the forward and backward propagation steps for a specified number of epochs or until convergence.
6. **Evaluation**
   - Evaluate the performance of the model on a validation dataset to check its accuracy and generalization ability.

CONCLUSION

In this experiment, we successfully built a deep neural network (DNN) from scratch, focusing on understanding essential concepts such as activation functions, forward and backward propagation, and loss computation. By implementing a DNN with multiple hidden layers and using the ReLU activation function, we effectively modeled complex patterns and achieved better performance in binary classification tasks. This process highlighted the importance of careful parameter initialization, accurate gradient computation, and iterative training in optimizing the network. Through this hands-on approach, we gained valuable insights into the workings of deep learning algorithms and their practical applications, providing a solid foundation for further exploration in the field of neural networks.

ASSIGNMENT NO. 11

TITLE

Implement the concept of regularization, gradient checking and optimization in convolutional model: step by step

OBJECTIVE

- Understand the flow and interaction between different layers in a CNN, including input, convolutional, activation, pooling, and fully connected layers.
- Write and structure functions for forward propagation, backward propagation, logistic loss, regularization, gradient checking, and optimization to facilitate the construction and debugging of a neural network.
- Set up and adjust parameters according to the desired CNN structure, ensuring the network can learn effectively from the data.
- Use filters to extract features from the input image at every position, producing an output volume (feature map) of different sizes.

PLATFORM REQUIRED

Operating System:

Windows, macOS, or Linux

Software/Tools:

Python:

- A versatile and widely used programming language for machine learning and deep learning projects.

Deep Learning Libraries:

- **TensorFlow** or **PyTorch**: These libraries provide comprehensive tools to build and train convolutional neural networks (CNNs). Both libraries offer support for implementing custom regularization techniques, gradient checking, and optimization algorithms.
- **Keras**: If using TensorFlow, Keras can simplify the creation and training of deep learning models.

Development Environment:

- **Jupyter Notebook**: For interactive development, code testing, and visualization.
- **Python IDE**: Like **PyCharm**, **VS Code**, or **Spyder** for managing larger projects or scripts.

Additional Libraries:

- **NumPy**: For efficient numerical operations, particularly when dealing with arrays and matrices.
- **Matplotlib**: For visualizing images, feature maps, and training progress.
- **OpenCV** or **Pillow**: For advanced image processing tasks.

THEORY

Input Layer

- The input layer is where the raw input is fed into the network. In the context of CNNs, this is typically an image or a sequence of images.
- The input image generally has dimensions such as width, height, and depth (e.g., 32x32x3 for a color image).

Convolutional Layer:

- Extracts features from the input data using learnable filters (kernels).
- Each filter slides over the input data and performs a dot product between the filter's weights and the corresponding region in the input image, producing a feature map.

Activation Layer:

- Introduces non-linearity into the network, allowing it to model complex patterns.
- ReLU, Tanh, and Leaky ReLU are typical activation functions used to process the feature maps.

Pooling Layer:

- Reduces the spatial size of the feature maps, speeding up computation, reducing memory usage, and helping prevent overfitting.

- **Types:** Common pooling operations include max pooling and average pooling.

Flattening Layer:

- **Purpose:** Converts the 2D feature maps into a 1D vector, which can be fed into the fully connected layers for classification or regression.

Fully Connected Layer:

- **Purpose:** Performs the final classification or regression task by taking the input from the previous layers and applying weights to compute the final output.

Output Layer:

- **Purpose:** Outputs the final prediction using activation functions like Sigmoid or Softmax to convert logits into probability scores.

ALGORITHM

1. **Import Necessary Libraries:**
    - Import numpy for numerical operations and matplotlib for plotting images.
2. **Set Parameters:**
    - Define the size of the input image, number of filters, kernel size, strides, and padding for the convolutional layer.
    - Initialize other parameters such as learning rate, regularization strength, and number of epochs for training.
3. **Define Kernel:**
    - Initialize the filter (kernel) for the convolutional layer, typically with small random values.
4. **Load and Plot Image:**
    - Load the input image and display it using matplotlib to visualize the raw data.
5. **Reformat the Image:**
    - Preprocess the image by resizing or normalizing it as required for input into the CNN.
6. **Apply Convolution Layer Operation:**
    - Perform convolution by sliding the filter over the image, computing the dot product at each position, and generating the feature map.
7. **Apply Activation Layer Operation:**

- Apply an activation function (e.g., ReLU) to the feature map to introduce non-linearity.

8. **Apply Pooling Layer Operation:**
   - Perform pooling (e.g., max pooling) to reduce the spatial dimensions of the feature map.
9. **Flatten and Connect Layers:**
   - Flatten the pooled feature maps and pass them through fully connected layers for final classification or regression.
10. **Implement Regularization, Gradient Checking, and Optimization:**
    - Apply regularization techniques like L2 regularization to prevent overfitting.
    - Use gradient checking to verify the correctness of your backward propagation implementation.
    - Implement an optimization algorithm (e.g., stochastic gradient descent, Adam) to update the network's parameters.

CONCLUSION

In this experiment, we explored the steps required to implement a Convolutional Neural Network (CNN) from scratch. By understanding and applying the concepts of convolution, activation, pooling, and fully connected layers, we were able to construct a basic CNN architecture. Additionally, we integrated essential techniques like regularization, gradient checking, and optimization to enhance the network's performance and ensure its robustness. This hands-on approach provided a solid foundation for building and understanding more complex deep learning models, demonstrating the practical applications and effectiveness of CNNs in tasks like image classification.

ASSIGNMENT NO. 17

TITLE

Implement a Deep Learning Application using Generative AI

OBJECTIVE

- Develop a deep learning application using Generative AI to create new data samples that resemble a given dataset.
- Explore the capabilities of Generative Adversarial Networks (GANs) or other generative models in synthesizing realistic data, such as images, text, or audio.
- Understand and implement the key components of Generative AI, including the generator and discriminator networks (in the case of GANs).

PLATFORM REQUIRED

Operating System:

- Windows, macOS, or Linux

Software/Tools:

Python:

- The primary programming language for deep learning and Generative AI projects.

Deep Learning Libraries:

- **TensorFlow** with **Keras** or **PyTorch**: Both frameworks provide robust support for building and training GANs and other generative models.
- **TensorFlow-GAN**: A library for training and evaluating Generative Adversarial Networks.

Development Environment:

- **Jupyter Notebook**: Ideal for interactive experimentation, visualization, and documentation.

- **Python IDE**: Options like **PyCharm**, **VS Code**, or **Spyder** are suitable for larger projects.

Additional Libraries:

- **NumPy**: For numerical operations and handling data arrays.
- **Matplotlib** or **Seaborn**: For visualizing generated data, loss curves, and training progress.
- **OpenCV** or **Pillow**: For advanced image processing tasks, especially when dealing with image datasets.
- **Scikit-learn**: For additional data preprocessing and evaluation metrics.

THEORY

Generative AI is a branch of artificial intelligence focused on creating models that can generate new data points from an existing dataset. The primary models used in Generative AI include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and autoregressive models.

Generative Adversarial Networks (GANs)

- GANs consist of two neural networks, a generator and a discriminator, that are trained together. The generator's goal is to produce data that is indistinguishable from the real data, while the discriminator's goal is to differentiate between real and generated data. Over time, the generator becomes better at creating realistic data, and the discriminator becomes better at distinguishing between real and fake data.

Generator: This network takes random noise as input and generates data samples that resemble the training data.

Discriminator: This network takes both real data and generated data as input and tries to classify them as real or fake.

- The training process involves optimizing the generator to fool the discriminator while simultaneously training the discriminator to correctly classify real and generated data. This adversarial process leads to the generation of high-quality data samples.

ALGORITHM

1. **Data Preparation**

   - Collect and preprocess the dataset to be used for training the generative model. This may involve normalizing data, resizing images, or tokenizing text.

2. **Model Architecture**

   - Define the architecture for the generator and discriminator networks.
   - The generator network usually consists of layers that upsample the input noise to produce data samples.
   - The discriminator network consists of layers that downsample the input data and output a probability score indicating whether the data is real or fake.

3. **Training Loop**

- Initialize the weights of both networks.
- For each training iteration:
- Generate fake data samples using the generator.
- Obtain real data samples from the dataset.
- Train the discriminator on both real and fake data, with corresponding labels.
- Train the generator by updating its weights to maximize the likelihood of the discriminator classifying its outputs as real.

4. **Evaluation**

   - After training, evaluate the quality of the generated data by visual inspection, quantitative metrics, or deploying the model in a practical application.

5. **Fine-Tuning**

   - Fine-tune the model by adjusting hyperparameters, adding regularization techniques, or altering the network architecture to improve the quality of the generated data.

CONCLUSION

In this experiment , we successfully implemented a deep learning application using Generative AI. By following the steps outlined, we explored the capabilities of Generative Adversarial Networks (GANs) to generate realistic data samples that closely resemble the original dataset The experiment demonstrated the critical components of a GAN, including the generator and discriminator networks, and how these networks interact through an adversarial training process. The generator learned to create synthetic data, while the discriminator improved its ability to differentiate between real and generated data. Over time, the GAN achieved a balance where the generated data became increasingly realistic. It highlights the practical applications of Generative AI, including data augmentation, creative content generation, and synthetic data creation for various industries. The skills and knowledge gained through this lab provide a solid foundation for further exploration and application of Generative AI techniques in real-world scenarios.