EXPERIMENT NO. 6

AIM: Implement Back Propagation Neural Network Algorithm for the given dataset.

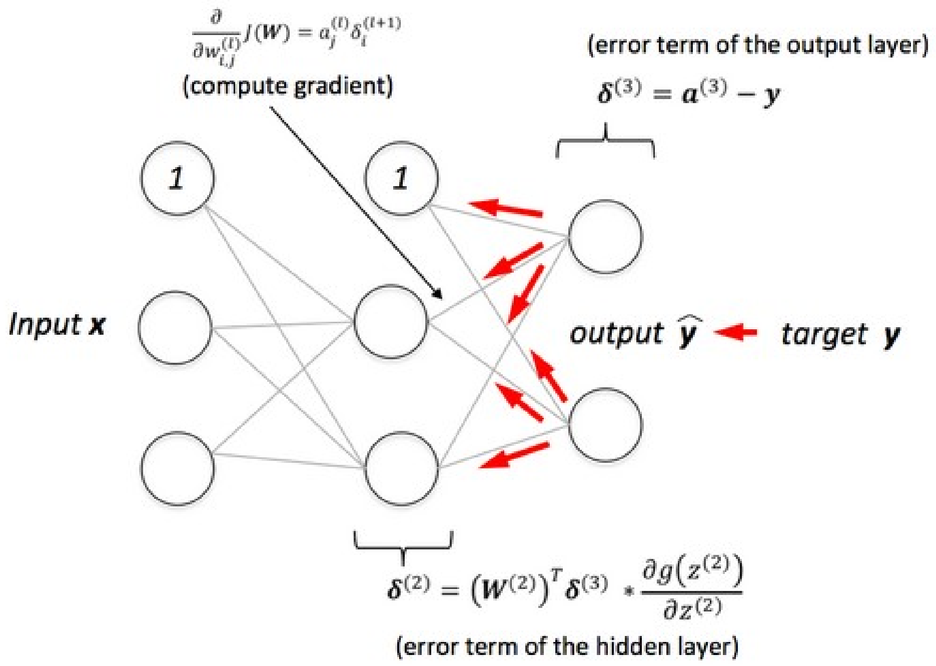
SOFTWARE: Python

LABORATORY OUTCOMES:

- Students will be able to implement Back Propagation Network algorithm for the given dataset.
- Students will be able to plot the **mean square error** and **accuracy** of Back Propagation Network

THEORY:

Backpropagation neural network is used to improve the accuracy of neural network and make them capable of self-learning. Backpropagation means "backward propagation of errors". Here error is spread into the reverse direction in order to achieve better performance. Backpropagation is an algorithm for supervised learning of artificial neural networks that uses the gradient descent method to minimize the cost function. It searches for optimal weights that optimize the mean-squared distance between the predicted and actual labels.
BPN was discovered by Rumelhart, Williams & Honton in 1986. The core concept of BPN is to back propagate or spread the error from units of output layer to internal hidden layers in order to tune the weights to ensure lower error rates. It is considered a practice of fine-tuning the weights of neural networks in each iteration. Proper tuning of the weights will make a sure minimum loss and this will make a more robust, and generalizable trained neural network.


$\frac{\partial}{\partial w_{i,j}^{(l)}} J(W) = a_j^{(l)} \delta_i^{(l+1)}$
(compute gradient)
(error term of the output layer)
$\delta^{(3)} = a^{(3)} - y$
1
1
Input x
output $\hat{y}$
target y
$\delta^{(2)} = (W^{(2)})^T \delta^{(3)} * \frac{\partial g(z^{(2)})}{\partial z^{(2)}}$
(error term of the hidden layer)

How BPN Works?

BPN learns in an iterative manner. In each iteration, it compares training examples with the actual target label. Target label can be a class label or continuous value. The backpropagation algorithm works in the following three steps:

Initialize Network: BPN randomly initializes the weights.
Forward Propagate: After initialization, we will propagate into the forward direction. In this phase, we will compute the output and calculate the error from the target output.

Back Propagate Error: For each observation, weights are modified in order to reduce the error in a technique called the delta rule or gradient descent. It modifies weights in a "backward" direction to all the hidden layers.

Implementation Steps of BPN using Python:

Step 1: Import Libraries: Lets import the required modules and libraries such as numpy, pandas, scikit-learn, and matplotlib.
Step 2: Load Dataset: Let's first load the Iris dataset using load_iris() function of scikit-learn library and seprate them in features and target labels. This data set has three classes Iris-setosa, Iris-versicolor, and Iris-virginica.
Step 3: Prepare Dataset: Create dummy variables for class labels using get_dummies() function.
Step 4: Split Train and Test set: To understand model performance, dividing the dataset into a training set and a test set is a good strategy. Let's split dataset by using function train_test_split(). you need to pass basically 3 parameters features, target, and test_set size. Additionally, you can use random_state in order to get the same kind of train and test set.
Step 5: Initialise Hyper parameters and Weights: Lets initialize the hyperparameters such as learning rate, iterations, input size, number of hidden layers, and number of output layers.
Lets initialize the weights for hidden and output layers with random values.
Step 6: Helper Function: Lets create helper functions such as sigmoid, mean_square_error, and accuracy. We will create a for loop for given number of iterations that execute the three steps (feedforward propagation, error calculation and backpropagation phase) and update the weights in each iteration.
Step 7: Plot MSE and Accuracy: Lets plot mean squared error in each iteration using pandas plot() funciton.
Step 8: Predict for Test Data and Evaluate the Performance: Let's make prediction for the test data and assess the performance of Backpropagation neural network.

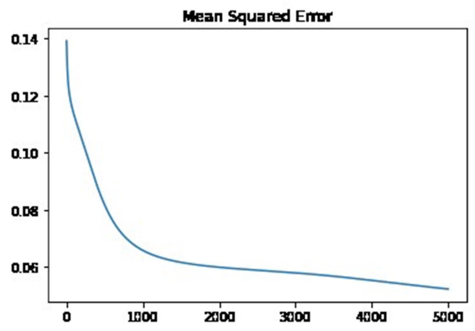
PROGRAM:

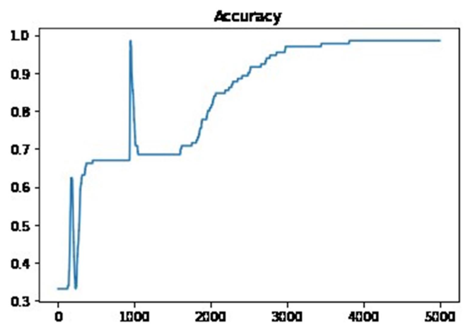
```
#Import libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Loading dataset
data = load_iris()
X = data.data
y = data.target

# Split dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)

# Hyperparameters
learning_rate = 0.1
```

```
iterations = 5000
N = y_train.size
input_size = 4
hidden_size = 2
output_size = 3

np.random.seed(10)
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))
W2 = np.random.normal(scale=0.5, size=(hidden_size, output_size))

# Helper functions
 def sigmoid(x):
 return 1 / (1 + np.exp(-x))
 def mean_squared_error(y_pred, y_true):
 # One-hot encode y_true (i.e., convert [0, 1, 2] into [[1, 0, 0], [0, 1, 0], [0, 0, 1]])
 y_true_one_hot = np.eye(output_size)[y_true]
# Reshape y_true_one_hot to match y_pred shape
y_true_reshaped = y_true_one_hot.reshape(y_pred.shape)
# Compute the mean squared error between y_pred and y_true_reshaped
error = ((y_pred - y_true_reshaped)**2).sum() / (2*y_pred.size)
return error
def accuracy(y_pred, y_true):
acc = y_pred.argmax(axis=1) ==  y_true.argmax(axis=1)
return acc.mean()
results = pd.DataFrame(columns=["mse", "accuracy"])

# Training loop
 for itr in range(iterations):
   # Feedforward propagation
   Z1 = np.dot(X_train, W1)
   A1 = sigmoid(Z1)
   Z2 = np.dot(A1, W2)
   A2 = sigmoid(Z2)

   # Calculate error
   mse = mean_squared_error(A2, y_train)
   acc = accuracy(np.eye(output_size)[y_train], A2)
   new_row = pd.DataFrame({"mse": [mse], "accuracy": [acc]})
   results = pd.concat([results, new_row], ignore_index=True)

   # Backpropagation
   E1 = A2 - np.eye(output_size)[y_train]
   dW1 = E1 * A2 * (1 - A2)
   E2 = np.dot(dW1, W2.T)
   dW2 = E2 * A1 * (1 - A1)

   # Update weights
```

```
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N
    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update

# Visualizing the results

results.mse.plot(title="Mean Squared Error")
plt.show()
results.accuracy.plot(title="Accuracy")
plt.show()

# Test the model

Z1 = np.dot(X_test, W1)
A1 = sigmoid(Z1)
Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)
test_acc = accuracy(np.eye(output_size)[y_test], A2)
print("Test accuracy: {}".format(test_acc))
```

OUTPUTS:





```
Test accuracy: 0.95
```

CONCLUSION:

Implementation of BPN for Irish dataset is carried out here. BPN is a method to optimize neural networks by propagating the error or loss into a backward direction. It finds loss for each node and updates its weights accordingly in order to minimize the loss using gradient descent.

TEXT/REFERENCE BOOKS:

- “Neural Network a Comprehensive Foundation” By Simon Haykin
- “Introduction to Soft Computing” By Dr. S. N. Shivnandam, Mrs. S. N. Deepa
- “Neural Network: A classroom Approach” By Satish Kumar
- “Neural Network, Fuzzy Logic and Genetic Algorithms” By Rajshekharan S, Vijayalakshmi Pai
- “Neural Network Design” by Hagan Demuth, Beale“Neural Network for Pattern Recognition”, Christopher M. Bishop

WEB ADDRESS (URLS):

- https://machinelearninggeek.com/backpropagation-neural-network-using-python/
- https://www.askpython.com/python/examples/backpropagation-in-python
- https://www.sebastian-mantey.com/theory-blog/basics-of-deep-learning-p13-implementing-the-backpropagation-algorithm-with-numpy

-------------------------------x------------------------------x--------------------------------