Experiment No : 01

Program Title : Implementing K Nearest Neighbor Algorithm

Objective :

1. Take a dataset and divide it into training and testing dataset
2. Implement KNN algorithm on testing dataset
3. Calculate accuracy
4. Do the same process for 60-40 , 70-30 and 80-20 training testing dataset and compare the accuracy

Methodology :

1. Select the value of k by taking the square root of total data points.
2. Take an unknown data point or testing data point as input.
3. Calculate Euclidean distance from testing data points to all training points
4. Sort the distance in ascending order
5. Pick the first 'K' points from the sorted list
6. Calculate the number of classes
7. The class with the highest number of occurrence will be the nearest neighbor or the class for that unknown or testing data points

Implementation in Code :

The implementation of KNN algorithm on a particular dataset is given next page.

Performance Analysis :

Training Percentage	Testing Percentage	Accuracy Percentage
60	40	99.0833
70	30	99.3333
80	20	99.3333

Conclusion and Observation :

1. **Instance-based learning:** KNN is an instance based learning , meaning that it doesn't exactly build a model for training dataset , instead it uses training dataset to for test dataset.
2. **Choice of K:** The accuracy mostly depends on the value of k. So it's wiser to check the accuracy for different values of k
3. **Curse of dimensionality :** The more the number of dimensions increases , the time taken for knn will also be higher.
4. **Feature Scaling :** For distance based algorithm like KNN , it's necessary to keep all the data in same scaling. If data are in different scaling , then it will result in low accuracy.

Experiment No : 02

Program Title : Implementing Single Layer Perceptron

Objective :

1. Create a dataset of n bit where values will be 0 and 1 only.
2. Output of First half row will be 0 and the rest half will be 1.
3. Divide it into train and test dataset.
4. Apply Single Layer Perceptron algorithm on test dataset
5. Calculate accuracy
6. Do the same process for 60-40 , 70-30 and 80-20 training testing dataset and compare the accuracy

Methodology :

For Training Phase :

1. Generate 'n' number of random weights where 'n' is the number of feature in dataset
2. Get a fixed value for threshold value and learning rate
3. Divide the dataset into train and test part.
4. Take the first row in training data as input and multiply all input data with corresponding weights and take the summation of it
5. If summation >= Threshold value , then the predicted output will be 1
6. If summation < Threshold value , then the predicted output will be 0.
7. If predicted output for that input and actual output for that input are same , then no weight updation is required and go to 10 no step.
8. If predicted output and actual output for that input are not same , then update the weight using the formula :

W_New = W_Old + learning_rate * (Actual Output – Predicted Output) * input

9. Break the loop and go back to 4 no step with the updated weights
10. Go to next input row.
11. Repeat the process from 4 to 10 until all the training input gets 100% classified with a certain set of weights.

For Testing Phase :

1. Take the first row in testing data as input and multiply all input data with corresponding weights and take the summation of it
2. If summation >= Threshold value , then the predicted output will be 1
3. If summation < Threshold value , then the predicted output will be 0.
4. Take the first row in training data as input and multiply all input data with corresponding weights and take the summation of it
5. Calculate the accuracy after checking how many of them are classified correctly.

Implementation in Code :

The implementation of Single Layer Perceptron algorithm on a particular dataset is given next page.

Performance Analysis :

Training Percentage	Testing Percentage	Accuracy Percentage
60	40	92
70	30	100
80	20	100

Conclusion and Observation :

1. The more the testing percentage increases , the higher the accuracy becomes.
2. If learning rate is higher then , it will reach to local minima quickly but when it's small , it reaches to local minima slowly.
3. This algorithm is time consuming for large dataset , since it is learning step by step.

Implementation in Code :

This implementation of Single layer perceptron has been done for groupwise learning.

Performance Analysis :

Training Percentage	Testing Percentage	Accuracy Percentage
60	40	92.1568
70	30	94.7882
80	20	96.0880

Conclusion and Observation :

1. The more the testing percentage increases , the higher the accuracy becomes.
2. If learning rate is higher then , it will reach to local minima quickly but when it’s small , it reaches to local minima slowly.
3. This algorithm is time consuming for large dataset , since it is learning groupwise

Experiment No : 03

Program Title : Implementing Backpropagation Algorithm

Objective :

1. Create a dataset of n bit where values will be 0 and 1 only.
2. Output of First half row will be 0 and the rest half will be 1.
3. Divide it into train and test dataset.
4. Apply backpropagation algorithm on test dataset
5. Calculate accuracy
6. Do the same process for 60-40 , 70-30 and 80-20 training testing dataset and compare the accuracy

Methodology :

For Training Phase :

1. Take 'n' as no of nodes in input layer which is equal to no of feature. Here for the convenience of code , no of hides layers is also considered to be 'n'. The number of output layer is m = 1 , since it's a binary classification.
2. Initialize Wij , bj , Wjk and bk with random values.
   Here ,
   Wij = n X n order of weight matrix between input and hidden layer
   bj = Bias to 'n' no nodes of hidden layer
   Wjk = (n X m = n X 1) order of weight matrix between hidden and output layer
   bk = Bias to m = 1 'no' of nodes of output layer.
3. Initialize Oi[n] , netj[n] , activj[n], Oj[n] and netk[m],activk[m], Ok[m] with 0.
4. Take first input and start Forward Propagation
5. If error <= 0.01 , then go to 8.

6. If error > 0.01 , then backpropagation starts.
7. Break the loop and go back to 4 no step with the updated weights
8. Go to next input row.
9. Repeat the process from 4 to 8 until all the training input gets 100% classified with a certain set of weights.

For Testing Phase :

1. Take first input and start Forward Propagation
2. If error <= 0.01 , then right = right + 1
3. If error > 0.01 , then wrong = wrong + 1.
4. Calculate the accuracy after checking how many of them are classified correctly.

Implementation in code :

The implementation of Multi layer perceptron or backpropagation has been done in the next page.

Performance Analysis :

Training Percentage	Testing Percentage	Accuracy Percentage
60	40	0.0
70	30	100.0
80	20	100.0

Conclusion and Observation :

In conclusion, backpropagation is a powerful and widely used algorithm for training neural networks. It addresses the challenge of optimizing the network's weights to minimize the error between predicted and target outputs. Backpropagation, combined with gradient descent optimization, enables deep neural networks to learn complex patterns and make accurate predictions in various domains. Here are observations :

1. The more the testing percentage increases , the higher the accuracy becomes.
2. If learning rate is higher then , it will reach to local minima quickly but when it's small , it reaches to local minima slowly.
3. This algorithm is time consuming for large dataset.

Experiment No : 04

Program Title : Implementing kohonen neural network

Objective :

1. Create a dataset of n bit where values will be 0 and 1 only.
2. Divide it into train and test dataset.
3. Apply kohonen neural network on test dataset
4. Determine which test data clusters with which output node

Methodology :

1. Take 2 layer , one of which is output layer where the number of nodes will be as same as the number of training data and other one is input layer where number of node is as same as number of feature.
2. Check if neighbor != 0 , If true then proceed and if not , go to 10.
3. Take first input and calculate distance from that input to every output node
4. Sort the distance in ascending order
5. Select the output node with minimum distance from that input
6. Update the node with minimum distance and it's neighbors
7. Continue 3 to 6 until the all training input are used
8. Update no of neighbor using this formula :

   Neighbor_new = Neighbor_old – learning_rate * Neighbor_old
9. Go to 2 again
10. End

Implemention in Code : The implementation of Kohonen neural network is at next page

Performance Analysis :

For Training Data :

Training Input No.	Output node
0	7
1	13
2	8
3	11
4	4
5	4
6	5
7	18
8	1
9	19
10	3
11	9
12	15
13	10
14	2
15	14
16	12
17	6
18	12
19	17

For Testing Data :

Testing Input No.	Output node
20	12
21	4
22	0
23	18
24	1
25	19
26	8
27	3
28	15
29	10
30	2
31	14

Conclusion and Observations : In conclusion, the Kohonen neural network, also known as the Self-Organizing Map (SOM), is a powerful unsupervised learning algorithm that can discover and represent the underlying structure of complex data. It offers a unique approach to clustering and visualizing high-dimensional data in a lower-dimensional space.

Experiment No : 05

Program Title : Implementing hopfield network algorithm

Objective :

1. Create a dataset of n bit where values will be 1 and -1 only.
2. Divide it into train and test dataset.
3. Apply hopfield network algorithm on test dataset
4. Determine which test data clusters with which training input.

Methodology :

1. Divide the dataset into training and testing part.
2. Take m = number of features
3. Number of neurons will also be m = number of features
4. Order of weight matrix will be (m X m)
5. If i == j , Wij = 0
6. else , Wij = Summation(feature_i * feature_j)
7. Take the first test input as new pattern
8. Take this new pattern at the first neuron
9. Summation = new pattern * W[neuron][:]
10. If summation > 0 , then new_pattern[neuron] = +1
11. elif summation < 0 , then new_pattern[neuron] = -1
12. else , no change
13. Take the updated_pattern to second neuron and then next neuron and keep updating until convergence happens

Implementation in Code : The implementation of hopfield network algorithm is at next page

Performance Analysis :

```
New Pattern : [1, -1, 1, -1]
At Neuron  0 : [1, -1, 1, -1]
At Neuron  1 : [1, -1, 1, -1]
At Neuron  2 : [1, -1, -1, -1]
At Neuron  3 : [1, -1, -1, -1]
At Neuron  0 : [1, -1, -1, -1]
At Neuron  1 : [1, -1, -1, -1]
At Neuron  2 : [1, -1, -1, -1]
Converged pattern of the test pattern : [1, -1, -1, -1]
Cluster with 8
----------------------------------------------------
New Pattern : [1, -1, 1, 1]
At Neuron  0 : [1, -1, 1, 1]
At Neuron  1 : [1, -1, 1, 1]
At Neuron  2 : [1, -1, -1, 1]
At Neuron  3 : [1, -1, -1, 1]
At Neuron  0 : [1, -1, -1, 1]
At Neuron  1 : [1, -1, -1, 1]
At Neuron  2 : [1, -1, -1, 1]
Converged pattern of the test pattern : [1, -1, -1, 1]
Cluster with 9
----------------------------------------------------
New Pattern : [1, 1, -1, -1]
At Neuron  0 : [1, 1, -1, -1]
At Neuron  1 : [1, -1, -1, -1]
At Neuron  2 : [1, -1, -1, -1]

At Neuron  3 : [1, -1, -1, -1]

At Neuron  0 : [1, -1, -1, -1]

At Neuron  1 : [1, -1, -1, -1]

Converged pattern of the test pattern : [1, -1, -1, -1]

Cluster with 8

----------------------------------------------------

New Pattern : [1, 1, -1, 1]

At Neuron  0 : [1, 1, -1, 1]

At Neuron  1 : [1, -1, -1, 1]

At Neuron  2 : [1, -1, -1, 1]

At Neuron  3 : [1, -1, -1, 1]

At Neuron  0 : [1, -1, -1, 1]

At Neuron  1 : [1, -1, -1, 1]

Converged pattern of the test pattern : [1, -1, -1, 1]

Cluster with 9

----------------------------------------------------

New Pattern : [1, 1, 1, -1]

At Neuron  0 : [-1, 1, 1, -1]

At Neuron  1 : [-1, 1, 1, -1]

At Neuron  2 : [-1, 1, 1, -1]

At Neuron  3 : [-1, 1, 1, -1]

At Neuron  0 : [-1, 1, 1, -1]

Converged pattern of the test pattern : [-1, 1, 1, -1]

Cluster with 6

----------------------------------------------------
```

```
New Pattern : [1, 1, 1, 1]

At Neuron  0 : [-1, 1, 1, 1]

At Neuron  1 : [-1, 1, 1, 1]

At Neuron  2 : [-1, 1, 1, 1]

At Neuron  3 : [-1, 1, 1, 1]

At Neuron  0 : [-1, 1, 1, 1]

Converged pattern of the test pattern : [-1, 1, 1, 1]

Cluster with 7

------------------------------------------------------
```

Conclusion and Observation :

he Hopfield neural network is a type of recurrent neural network (RNN) that is capable of storing and retrieving patterns or memories. It provides a unique approach to associative memory and pattern recognition tasks. The Hopfield network is designed to converge to stable states or attractors. Each stored pattern in the network corresponds to an attractor state, and the network dynamics converge to one of these states during recall. This convergence property allows the network to recognize and complete partial or noisy input patterns.