Heaven's Light is Our Guide

Rajshahi University of Engineering & Technology

Course No: CSE 4204

Course Title: Sessional Based on CSE 4203

Experiment No. 4

Name of the Experiment:

a. Design and implementation of Kohonen Self-organizing Neural Networks algorithm.

b. Design and implementation of Hopfield Neural Networks algorithm.

Submitted by:
Ashraf-Ul-Alam
Roll: 1803070
Section: B
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

Submitted to:
Rizoan Toufiq
Assistant Professor
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology

Date of Submission: 19 December 2023

Contents

1 Introduction

This report explores the Kohonen Self-Organizing Map (SOM) and Hopfield Network algorithms in the context of the "Indian Liver Patient Records"[1] dataset. The Kohonen Network is designed for unsupervised learning, offering a unique approach to clustering and visualizing high-dimensional data. In contrast, the Hopfield Network is a recurrent neural network known for its applications in associative memory. This report examines the algorithms' performance in learning patterns and making predictions by implementing and testing on real-world medical data.

2 Dataset

2.1 Dataset Analysis

This analysis began by loading the dataset from the "indian_liver_patient.csv" [1] file. The dataset contains information on various attributes related to liver health, such as age, gender, bilirubin levels, liver enzymes, and more. The dataset has 583 entries and 11 columns.

- Age: An integer representing the patient's age.
- Gender: Categorical feature (0 for female, 1 for male).
- Total Bilirubin, Direct Bilirubin, Alkaline Phosphotase, Alamine Aminotransferase, Aspartate Aminotransferase: Numeric measurements.
- Total Proteins, Albumin, Albumin and Globulin Ratio: Numeric measurements.
- Dataset: The target variable (1 for liver disease, 2 for no liver disease).

2.2 Feature Processing

The original dataset contained the 'Gender' feature as a categorical variable with string values representing male and female. To make this feature compatible with the perceptron learning algorithm, which requires numerical input, the 'Gender' data was transformed into a numeric format using Scikit-Learn's Label Encoder, assigning '0' for female and '1' for male. Simultaneously, the 'Dataset' column is transformed to have 0 represent the presence of liver disease and 1 indicate the absence of liver disease.
This encoding step is crucial because it converts categorical data into a numerical format that neural networks can understand. It's essential for the algorithm to process this data accurately and seamlessly, ensuring it integrates well and performs effectively within the model.

2.3 Handling Missing Values

The missing values within the dataset were also checked and it was found that the "Albumin_and_Globulin_Ratio" column had four missing values. To address this, the missing values were filled with the mean of the column.

2.4 Correlation Analysis

For both the Kohonen Network Algorithm and the Hopfield Network Algorithm, a correlation analysis was conducted to assess feature suitability and visaully represented by using a heatmap [1]. The absence of significant correlations indicates that each feature independently influences the algorithms' processes. This independence is advantageous as it ensures accurate representation (Kohonen) and robust memory recall (Hopfield) without inter-feature dependencies, enhancing the reliability of both algorithms' outcomes.

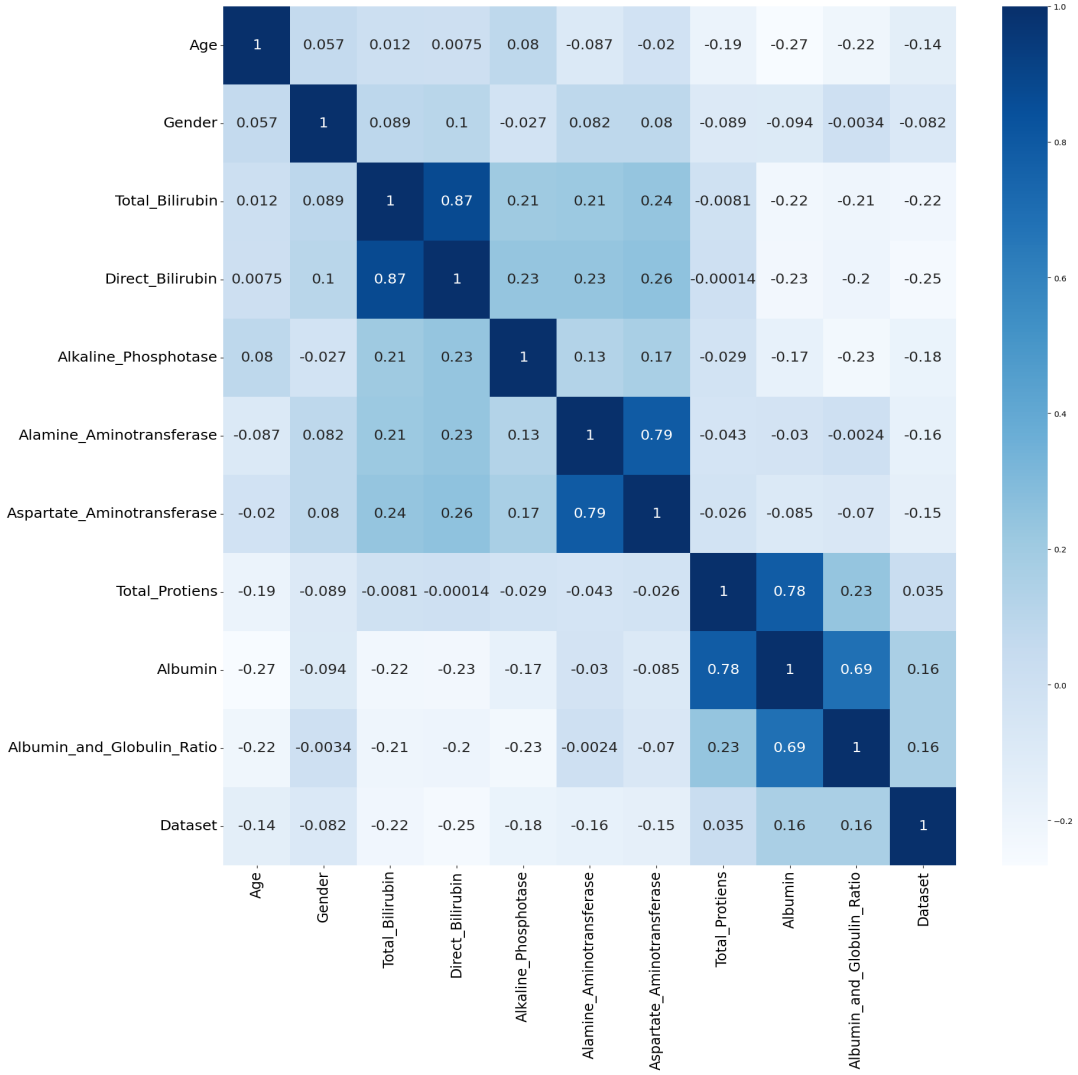

Figure 1: Correlation Heatmap

2.5 Data Type Conversion

To maintain consistency and avoid data loss, all the features were converted to the float data type.

2.6 Data Imbalance

A significant class imbalance can be observed in the target variable ("Dataset") with more instances of one class (0) compared to the other (1). To address this imbalance, Synthetic Minority Over-sampling Technique (SMOTE) was applied to oversample the minority class. The following table [1] illustrates the class distribution before and after applying SMOTE:

Table 1: Class Distribution Before and After SMOTE

Class	**Count Before SMOTE**	**Count After SMOTE**
0	416	416
1	167	416

In the original dataset, class 0 (indicating patients with liver disease) had 416 instances, while class 1 (indicating patients without liver disease) had only 167 instances. After applying SMOTE, both classes have an equal number of instances (416), effectively addressing the class imbalance issue.

Balancing the dataset through SMOTE ensures that the machine learning model is trained on a more representative dataset, preventing bias towards the majority class and improving its ability to make accurate predictions for both classes.

2.7 Z-score Normalization using StandardScaler

The dataset preprocessing involves employing 'StandardScaler' from 'sklearn.preprocessing' to perform Z-score normalization. This process standardizes each feature to possess a mean of 0 and a standard deviation of 1. By centering the data around 0 and scaling it using standard deviations, this method ensures that all features contribute equally to prevent any single feature from dominating the learning process. Z-score normalization is critical for enhancing model convergence rates and maintaining fair feature contributions, particularly in machine learning algorithms reliant on gradient descent or distance-based calculations.

2.8 Train-Test Split

The data was split into training and testing sets using train_test_split method from sklearn.model selection in Python.. The test_size parameter was set to 0.20 for spliting the dataset into 80% train data and 20% test data.

3 Methodology

3.1 Kohonen Self-Organizing Map (SOM) Algorithm

The Kohonen Self-Organizing Map (SOM) is an unsupervised learning algorithm used for clustering and visualization of high-dimensional data.

Algorithm 1 Kohonen Self-Organizing Map Training

Inputs:
$data$: Input dataset
$epochs$: Number of training epochs
$initial\_learning\_rate$: Initial learning rate
$input\_size$: Size of the input vectors
$num\_cluster$: Number of clusters (map units)
Initialize weights randomly: $weights \leftarrow$ random values within range($num\_cluster$, $input\_size$)
for $epoch$ **in** $epochs$ **do**
  for $input\_vector$ **in** $data$ **do**
    Compute distances between weights and input vector:
      $distances \leftarrow$ Compute squared Euclidean distances
    Find the index of the neuron with the minimum distance:
      $min\_distance\_index \leftarrow \text{argmin}(distances)$
    Calculate learning rate for current epoch:
      $learning\_rate \leftarrow initial\_learning\_rate \times e^{-epoch/epochs}$
    Update weights of the winning neuron and its neighbors:
      $weights[min\_distance\_index] \leftarrow weights[min\_distance\_index] + learning\_rate \times (input\_vector - weights[min\_distance\_index])$
  end for
end for
return Trained weights

By iteratively adjusting weights based on input vectors and their distances, the SOM organizes data into a topological map, preserving the input space's structure. The gradual reduction of the learning rate over epochs ensures convergence while maintaining the map's integrity. A topological map retains the relative relationships between data points. Neighboring regions in the input space that are similar or adjacent to each other should also be represented as neighboring regions in the map. Neurons in the SOM grid are arranged in a way that reflects the structural organization of the input data. Similar data points tend to cluster together on the map, making it easier to identify clusters, patterns, or groupings within the data. This ordered representation allows for intuitive visualization and interpretation.

3.2 Hopfield Network Algorithm

The Hopfield Network Algorithm involves two key steps: weight matrix calculation and pattern reconstruction. The weight matrix is computed based on training patterns, enabling the associative recall of stored patterns. By iteratively updating neuron states using the weight matrix and a hard-limiting nonlinearity, the network converges to stable states representing retrieved patterns from unknown test data. This capacity for associative memory makes it valuable for tasks involving pattern completion and error correction.

3.2.1 Hopfield Network Algorithm: Weight Matrix Calculation

The weight matrix W in a Hopfield Network is directly computed from the training patterns:

$$W_{ij} = \sum_k X_{ik} \times X_{jk} \quad \text{for} \quad i \neq j$$

This matrix captures associations among patterns, enabling associative memory recall without explicit learning mechanisms.

Algorithm 2 Hopfield Network: Weight Matrix Calculation

Inputs:
  X_train: Training patterns
Initialize connection weights based on training patterns:
  $W \leftarrow$ Zero matrix of size $(num\_nodes \times num\_nodes)$
for i **in** range(num_nodes) **do**
  for j **in** range(num_nodes) **do**
    if $i \neq j$ **then**
      Calculate the weight $W[i, j]$ based on training patterns:
        $W[i, j] \leftarrow \sum_k X\_train[k, i] \times X\_train[k, j]$
    end if
  end for
end for
return Weight matrix: W

3.2.2 Hopfield Network Algorithm: Reconstruct Pattern for Unknown Test Patterns

The reconstructed patterns in a Hopfield Network are derived using the previously calculated weight matrix W. This matrix guides the iterative update of neuron states, aiming to converge to stable states representing reconstructed patterns from unknown test inputs.

Algorithm 3 Hopfield Network: Reconstruct Pattern for Unknown Test Patterns

Inputs:
W: Weight matrix
X_test: Test patterns
$num\_iteration$: Maximum number of iterations
Create an empty list for reconstructed patterns: $reconstructed\_patterns \leftarrow []$
for μ **in** X_test **do**
  Reinitialize μ as the current test pattern
  for _ **in** range($num\_iteration$) **do**
    Copy the current pattern: $prev\_\mu \leftarrow \mu$.copy()
    for i **in** range(num_nodes) **do**
      Calculate total input for neuron i:
        $total\_input \leftarrow \sum_{j \neq i} W[i,j] \times \mu[j]$
      Apply hard-limiting nonlinearity to the total input:
        $\mu[i] \leftarrow$ hard_limiting_nonlinearity($total\_input$)
    end for
    if $\mu == prev\_\mu$ **then**
      Break if convergence is reached
      break
    end if
  end for
  Append the reconstructed pattern to the list:
    $reconstructed\_patterns$.append(μ)
end for
return Reconstructed patterns: $reconstructed\_patterns$

4 Results Analysis

4.1 Kohonen Network

To evaluate the performance of the Kohonen Network, we compute two fundamental metrics, the Quantization Error (QE) and Normalized Quantization Error (NQE), specifically on unseen data. These metrics serve as critical indicators of the network's accuracy in mapping input data onto its nodes.

The Quantization Error (QE) measures the average distance between input vectors and their corresponding winning neurons within the network grid. It is calculated using the following formula:

$$\text{QE} = \frac{1}{N} \sum_{i=1}^{N} ||\text{input vector}_i - \text{best matching weights}_i||^2$$

Here, N represents the number of input vectors.

The Normalized Quantization Error (NQE) provides a normalized assessment of the QE by considering the average distance between input vectors. It is calculated as the ratio of QE to the average distance between input vectors:

$$\text{NQE} = \frac{\text{QE}}{\text{Average distance between input vectors}}$$

For the Kohonen Network, two scenarios were evaluated: one using the original dataset (X_{train}) normalized by z-score normalization (as mentioned in Section 2.7) and another using the dataset reduced through Principal Component Analysis (PCA), denoted as $X_{\text{train\_pca}}$ with $n\_components = 2$.

4.1.1 Original Dataset (Z-Score Normalized)

When applying the Kohonen Network to the original dataset, the following results were obtained:

- QE: 2.287
- NQE: 1.28

4.1.2 PCA-Reduced Dataset

Upon applying PCA with $n\_components = 2$ and subsequently training the Kohonen Network on the reduced dataset ($X_{\text{train\_pca}}$), the results were as follows:

- QE: 1.45
- NQE: 0.60

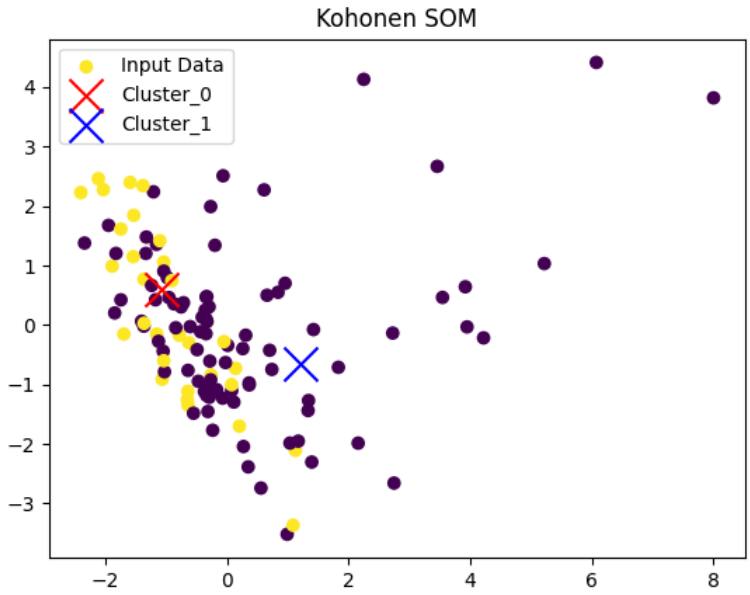

Figure 2: Input data from $X_{\text{train\_pca}}$ and Clusters of Kohonen SOM.

The Figure 2 represents the visualization of a Kohonen SOM applied to the input data ($X_{\text{test\_pca}}$) with corresponding cluster assignments (y_{test}.values). Each marker in the plot denotes a data point in the input space, distinguished by different colors, while the 'x' markers highlight the identified cluster centers by the Kohonen SOM. This visual representation effectively demonstrates how the SOM organizes the input data into clusters, facilitating a comprehensive analysis and interpretation of the algorithm's performance.

4.1.3 Analysis

Comparing the results between the original dataset and the PCA-reduced dataset, PCA demonstrates a notable improvement in reducing the quantization error. The reduced normalized quantization error suggests that training the network on PCA-transformed data achieves a more accurate mapping of input patterns to the network grid.

The effectiveness of PCA as a preprocessing step for dimensionality reduction in Kohonen Networks is evident from the reduced errors. Utilizing PCA for dimensionality reduction improves the network's performance, resulting in a more efficient representation of input patterns and improved quantization accuracy.

4.2 Hopfield Network

4.2.1 Data Transformation

The data were first normalized (as mentioned in Section 2.7) and then transformed into a binary format for compatibility with the Hopfield Network. The transformation involved assigning -1 to values below a threshold of 0 and +1 to values equal to or above the threshold:

$$X_{\text{binary}} = \text{np.where}(X_{\text{scaled}} \geq 0, 1, -1)$$

This conversion to binary states was essential for the Hopfield Network's operation, as it functions with bipolar binary states (-1 and +1). Then the X_{binary} was train test splitted as mention in Section 2.8.

4.2.2 Weight Matrix Calculation

The weight matrix (W) was computed using the transformed training data (X_{train}). This matrix encapsulates the learned associations from the training patterns.

4.2.3 Pattern Reconstruction and Accuracy

The network's performance was assessed by reconstructing patterns for unseen data (X_{test}). To evaluate the accuracy of pattern recalls, the reconstructed patterns were compared to the original patterns from the test set.

The accuracy was determined by counting the number of correctly recalled patterns and dividing it by the total number of test patterns. The recall accuracy was computed as:

$$\text{Recall Accuracy} = \frac{\text{Number of Correct Recalls}}{\text{Total Number of Test Patterns}}$$

Finally, the Pattern Recall Accuracy was expressed as a percentage:

$$\text{Pattern Recall Accuracy} = \text{Recall Accuracy} \times 100\%$$

The achieved Pattern Recall Accuracy of 100.00% indicates that the Hopfield Network successfully recalled all patterns from the unseen test data, showcasing its robust pattern reconstruction ability.

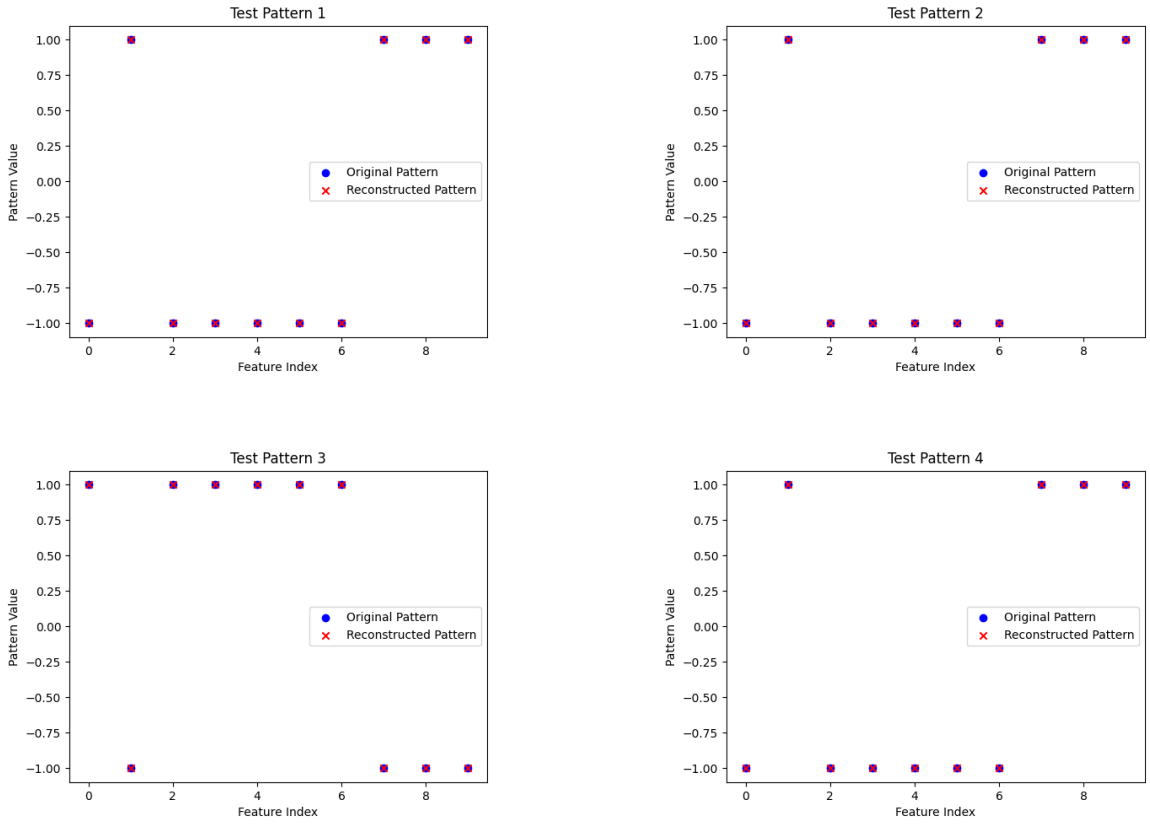

Figure 3: Original Pattern vs Reconstructed Pattern from Test Data

The Figure 3 depict the comparison between the Original Patterns and their corresponding Reconstructed Patterns from the test data, illustrating successful matches between each Reconstructed Pattern and its respective Original Pattern.

5 Conclusion

In conclusion, the study explored the Kohonen and Hopfield Networks' capabilities in pattern analysis and reconstruction. The Kohonen Network benefited from PCA-driven dimensionality reduction, exhibiting improved accuracy in capturing data structure. Meanwhile, the Hopfield Network, after data transformation, achieved a flawless Pattern Recall Accuracy of 100.00%, showcasing robust pattern reconstruction abilities.

The study highlights the efficacy of preprocessing techniques like PCA and data transformation, showcasing how they enhance neural network performance in distinct aspects of pattern analysis. The findings emphasize the suitability of Kohonen and Hopfield Networks for various pattern-related tasks in real-world data applications.

References

[1] UCI Machine Learning Repository. [Online]. Available: https://archive.ics.uci.edu/ [Accessed: November 2, 2023].