University of Technology - Iraq

Department of Computer Sciences – Multimedia Branch

Adaptive Systems

Third Class – Second Course

Lecturer: Asst. Prof. Dr. Hiba Basim Alwan

2024 – 2025

University of Technology

Department of Computer Science

Adaptive Systems | Multimedia Branch | Third Stage | 2023 - 2024

Adaptive Systems | Multimedia Branch | Third Stage | 2024 - 2025

Asst. Prof. Dr. Hiba Basim Alwan

Table of Contents

Experiment One: Back Propagation Neural Network

Introduction

Back Propagation (BP) Neural Network (NN) is the most widely used model for practical applications. It is a powerful mapping network that has been successfully applied to a wide variety ranging from credit application scoring to image compression.

BP NN is a systematic method for training multi-layer NN. It has a mathematical foundation that is strong if not highly practical. BP NN is usually layered with each layer fully connected to the layers before and after neurons are not connected to other neurons in the same layer. Typically, BP NN employs three or more layers of neurons including the input layer.

The Aim of the Experiment

This experiment aims to write a suitable code for the BP NN algorithm and show the student what will happen if the values of the input are changed.

The Algorithm

Step 1: Initialize weights to small random values

Step 2: While the stopping condition is false do steps 3 to 10

Step 3: for each training pair do steps 4 to 9

Feedforward

Step 4: each input unit (x_i, i = 1 to n) receives input signal xi and broadcasts this signal to all units in the layer above (the hidden layer)

Step 5: each hidden layer unit (z_j, j = 1 to p) sums its weighted input signals

$$z_{inj} = v_{0i} + \Sigma x_j v_{ij}$$

and applies its binary sigmoid activation function to compute its output signal and sends this signal to all units in the layer above (the output layer)

Step 6: each output unit (y_k, k = 1 to m) sums its weighted input signal

$$y_{ink} = w_{0k} + \Sigma z_j w_{jk}$$

and applies its binary sigmoid activation function to compute its output signal

Back propagate error

Step 7: each output unit (y_k, k = 1 to m) receives a target pattern corresponding to the input training patterns, computes its error information term and calculates its weights correction term used to update w_{jk} later

$$\delta_{2k} = y_k\,(1 - y_k)\,(T_k - y_k)$$

where T_k is the target pattern and k = 1 to m

Step 8: each hidden unit (z_j, j = 1 to p) computes its error information term and calculates its weights correction term used to update v_{ij} later

$$\delta_{1j} = z_j\,(1 - z_j)\,\Sigma\delta_{2k} w_{jk}$$

Update weights

Step 9: each output unit (y_k, k = 1 to m) updates its weights

$$new\ w_{jk} = \eta\delta_{2k} z_j + \alpha\ old\ w_{jk}\ j = 1\ to\ p$$

each hidden unit (z_j, j = 1 to p) updates its weights

$$new\ v_{ij} = \eta\delta_{1j} x_i + \alpha\ old\ v_{ij}\ i = 1\ to\ n$$

Step 10: test stopping condition

Experiment Two: Hopfield Neural Network

Introduction

The Hopfield network model is probably one of the most popular types of NN. There are several versions of Hopfield networks. They can be used as auto-associated memory.

The basic idea of the Hopfield network is that it can store a set of exemplar patterns as multiple stable states. Given a new input pattern, which may be partial or noisy, the network can converge to one of the exemplar patterns that is nearest to the input pattern. This is the basic concept of applying the Hopfield network as associative memory.

The Aim of the Experiment

This experiment aims to write a suitable code for the Hopfield NN algorithm and show the student what will happen if the values of the input are changed.

The Algorithm

Step 1: Initialize weights to store patterns

Step 2: While activations of the net are not converged do steps 3 to 9

Step 3: For each input vector x do steps 4 to 8

Step 4: Set initial activations of net equal to the external input vector

$$y_i = x_i \text{ for } i = 1 \text{ to } n$$

Step 5: Do steps 6 to 8 for each unit y_i

Step 6: Compute net input $y_{inj} = x_i + \Sigma y_j w_{ji}$

Step 7: Determine activation (output signal, **note:** the standard steps activation function)

Step 8: Broadcast the value of y_i to all other units

Step 9: Test for convergence

Experiment Three: Bidirectional Associative Memory Neural Network

Introduction

Bidirectional Associative Memory (BAM) is a type of recurrent NN. BAM is hetero associative memory, meaning given a pattern, it can return another pattern which is potentially of different size.

The topology of BAM contains two layers of neurons x and y. Layer x and layer are fully connected. Once the weights have been established, input into layer x, present the pattern in layer y and vice versa.

The Aim of the Experiment

This experiment aims to write a suitable code for the BAM NN algorithm and show the student what will happen if the values of the input are changed.

The Algorithm

Step 1: Consider the value of M, as BAM will be constructed with M pairs of patterns. Here the value of M is 4.

Step 2: Set A as input patterns

Step 3: Set B as the corresponding target patterns

Step 4: Assign the neurons in the input and output layers. Here, neurons in the input layer are 6 and the output layer is 3

Step 5: Compute the weight matrix using the BAM algorithm

Step 6: Test the BAM model for the input patterns which will return the corresponding target patterns as output. As well as for each of the target patterns, the BAM model will return the corresponding input patterns.

Experiment Four: Kohonen Neural Network

Introduction

The objective of a Kohonen network is to map input vectors (patterns) of arbitrary dimension N onto a discrete map with 1 or 2 dimensions. Patterns close to one another in the input space should be close to one another in the map: they should be topologically ordered. A Kohonen network is composed of a grid of output units and N input units. The input pattern is fed to each output unit. The input lines to each output unit are weighted. These weights are initialized to small random numbers.

The Aim of the Experiment

This experiment aims to write a suitable code for the Kohonen NN algorithm and show the student what will happen if the values of the input are changed.

The Algorithm

Step 0: Initialize weights w_{ij}

Set topological neighbourhood parameters

Set learning rate parameters

Step 1: While the stopping condition is false, do steps 2 – 8

Step 2: for each input vector x, do step 3 – 5

Step 3: for each j, compute the distance

$$D(j) = \Sigma(x_i - w_{ij})^2 \text{ Euclidean distances}$$

Step 4: Find index J such that D(J) is a minimum

Step 5: For all units j within a specified neighbourhood of J, and all i:

$$W_{ij}(new) = W_{ij}(old) + \propto (x_i + W_{ij}(old))$$

Step6: Update learning rate

Step7: Reduce the radius of the topological neighbourhood at specified times

Step8: Test stopping condition

Experiment Five: Create the Initial Population of the Genetic Algorithm

Introduction

Genetic Algorithm (GA) is a type of Evolutionary Algorithms (EA), a subset of machine learning, i.e., a search algorithm inspired by the Darwinian theory of biological evolution. In the 1970s, Holland designed the GA as a way of exploiting the potential of the natural evolution to employ computers. In GA, solutions are generated randomly and sent to an evaluator. The evaluator reports whether the solution posed is optimal. In GAs, this generation and test process is repeated iteratively over a set of solutions.

The Aim of the Experiment

This experiment aims to write a suitable code for generating a random population for GA and then evaluate each individual in this generation.

The Algorithm

For i = 1 to no. size of population

1. Randomly create an initial population of individual fixed-length (binary, integer, real, or character) strings.
2. Evaluate the randomly created individual by calculating the fitness of each individual in the population.

End

Experiment Six: Selecting Individuals to Create the Initial Population of the Genetic Algorithm

Introduction

Selection is the attribute of GA that determines which individuals will advance to the population of the next generation. The basic method of selection is to simply choose the highest-fitness individuals formed from the children of one generation and use them to form the next generation. Some of the selection methods are:

- Steady State Selection
- Elitist Selection
- Tournament Selection
- Roulette Wheel Selection

The Aim of the Experiment

This experiment aims to write a suitable code for the Roulette Wheel selection method to generate a new population. The basic idea is to determine the selection probability or survival probability for each chromosome proportional to the fitness value. Then a Roulette Wheel can be made to display these probabilities. The selection process is based on spinning the wheel several times equal to population size, each time selecting a single chromosome for the new population. The Roulette Wheel selection algorithm is illustrated in the following algorithm:

Begin

    P=0;

```
        J=0;
        Rand=Random × sum of fitness;
        Repeat
                J=J+1;
                P=P + fitness of J;
        Until ((P>Rand) OR (J=Max. population));
        Select =J;
End
```

Experiment Seven: Create the New Population of the Genetic Algorithm

Introduction

New population, i.e., reproduction occurs in two ways:

- The parent is simply copied.
- Apply crossover.

In GA, evaluation from generation to generation is simulated both by preserving the genetic information contained in the chromosome strings of fit individuals and by altering this information using random genetic changes. The goal of preserving the genetic information of fit individuals is achieved through crossover. Crossover creates child individuals by crossing over portions of two-parent individuals' chromosomes. One or both of the child individuals are retained in the new child population, and the child individuals are required to be unique concerning the other children and the parent population. The child population is unique but the crossover operator ensures that the genetic information of the parent population is preserved. Note that the crossover point, that point where the crossover takes place is randomly determined.

The Aim of the Experiment

This experiment aims to write a suitable code for generating a new population for GA and then evaluate each individual in this generation. This new population is generated through either simply copying the chromosome of the previous population to the new one or through applying one type of crossover which are:

- One point crossover
- K point crossover

The Algorithm

Create a new population of strings by applying at least the first two of the following three operations. The operations are applied to individual string(s) in the population chosen with a probability based on fitness.

Begin

Set P_c

For i = 1 P_c

1. Select first parent
2. Select second parent
3. Determine the type of crossover
4. Cross the selected parents to generate the new individuals based on the type of crossover

End

End

Experiment Eight: Other Operators of the Genetic Algorithm

Introduction

The goal of introducing change to the information in the chromosome string of individuals created by crossover is achieved with the mutation, addition, deletion and permutation operators.

The mutation operator introduces new information into the chromosome string of an individual by randomly altering one or more genes in that string. The mutation is used to prevent getting stuck in local maxima in a population. A randomly determined gene in the chromosome is changed to take on a new value from the alphabet.

The addition operator randomly adds a gene to the chromosome string. In the following example, a randomly determined gene from the alphabet is added at a random point in the chromosome. The randomly selected addition point is denoted by the symbol.

The deletion operator randomly deletes a gene from the chromosome string. In the following example, a randomly determined gene is removed from the chromosome.

The Aim of the Experiment

This experiment aims to write a suitable code for mutation, addition, and deletion.

The Algorithm

Begin

Set P_m, P_a, P_d

Select the type of random change

If a mutation is selected

For i = 1 to P_m

Select parent

Create a new individual by randomly altering one or more genes in that parent

End

If the addition is selected

For i = 1 to P_a

Select parent

Create a new individual by adding one or more genes to that parent

End

If deletion is selected

For i = 1 to P_d

Select parent

Create a new individual by deleting one or more genes in that parent

End

End

Experiment Nine: Implement Simple Genetic Algorithm

Introduction

GA is a search algorithm based on the mechanics of natural selection and natural genetics. From a mechanistic view, GA is a variation in the generated and test method. Solutions are generated and sent to an evaluator. The evaluator reports whether the solution posed is optimal. In GAs, this generation and test process is repeated iteratively over a set of solutions. The evaluator returns information to guide the selection of new solutions for the following iterations. In other words, we can say that GAs are solving methods requiring domain-specific knowledge that is often heuristic and they are a family of adaptive search procedures. GA derives their name from the fact that they are loosely based on models of genetic change in a population of individuals.

The Aim of the Experiment

This experiment aims to write a suitable code for implementing simple GA.

The Algorithm

a. Randomly create an initial population of individual fixed-length character strings.
b. Iteratively perform the following sub-steps on the population of strings until the termination criterion has been satisfied:
    i Evaluate the fitness of each individual in the population.
    ii Create a new population of strings by applying at least the first two of the following three operations. The operations are applied to individual string(s) in the population chosen with a probability based on fitness.

1. Copy existing individual strings to the new population.
2. Create two new strings by genetically recombining randomly chosen substrings from two existing strings.
3. Create a new string from an existing string by randomly mutating the character at one position in the string.

c. The best individual string that appeared in any generation (i.e., the best-so-far individual) is designated as the result of the GA for the run. The result may represent a solution (or an approximate solution) to the problem.