Group No: 02

Sensors/ Actuators & Intelligent Systems: ME2281

Semester: 4

Artificial Neural Network Assignment

By

Index No.	Name	Marks
220638J	Thejasvenan T	

Date of submission	04/05/2025
Due date of submission	04/05/2025

Department of Mechanical Engineering
University of Moratuwa
Sri Lanka

Table of Contents

1 Introduction

Artificial Neural Networks (ANNs) are powerful tools in pattern recognition and machine learning. This assignment focuses on training an ANN to recognize the letters **C, T, and H** using a **3×3-pixel input**. The model is trained through **genetic algorithms**, which iteratively refine weights and thresholds to optimize accuracy.

By leveraging **mutation, crossover, and fitness evaluation**, the ANN evolves over **1000 generations**, improving its ability to identify letter patterns. The objective is not necessarily to achieve perfect results but to accurately implement the **training process** and understand key concepts behind ANN and genetic algorithms.

This report documents the **training methodology**, computational steps, and analysis of results, while reflecting on learning outcomes gained through the assignment.

2 Problem Definition

The goal of this assignment is to train an **Artificial Neural Network (ANN)** to recognize the letters **C, T, and H** based on a **3×3-pixel input grid**. Using **genetic algorithms**, the ANN will refine its weights and thresholds to improve accuracy through evolutionary training.

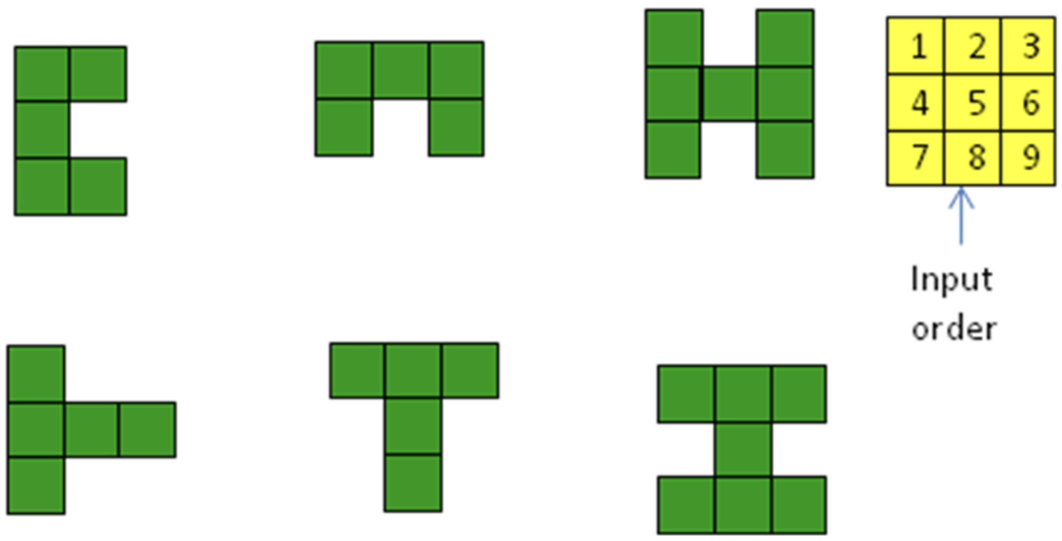


Fig 1: Different Orientations

Each letter may appear in different orientations, as shown in **Figure 1**, requiring the ANN to generalize its learning. The **input pixels are encoded as 1 (image present) or 0 (image absent)**, forming a **nine-element input vector**. If the letter does not fully occupy a **3×3 grid**, it is left-justified or top-aligned for consistency.

Expected Output:

Each letter is mapped to a unique **two-bit output**:

- **Letter C:** Output (1, 0)
- **Letter T:** Output (0, 1)
- **Letter H:** Output (1, 1)
- **None of the above:** Output (0, 0)

3 Calculation Process

For this assignment, **8 hidden nodes** have been selected for the ANN structure.

- The step processing function is applied for both hidden and output nodes.
- This means the neuron output will be either 0 or 1, based on a threshold.

The weight connections are calculated as follows:

- Between input and hidden layers:

$$\text{Total Weights} = \text{Number of Inputs} \times \text{Number of Hidden Nodes}$$

$$9 \times 8 = 72$$

- Between hidden and output layers:

$$\text{Total Weights} = \text{Number of Hidden Nodes} \times \text{Number of Output Nodes}$$

$$8 \times 2 = 16$$

Total weights = 72 + 16 = 88

Total thresholds (one per hidden and output node) = 8 + 2 = 10

Thus, the final chromosome length (total number of parameters) = 98.

To begin ANN training, the initial values for weights and thresholds are randomly assigned within the range:

- Weight Range: [-1, 1]
- Threshold Range: [0, 1]

Each value is encoded using **12 bits per variable** to support binary representation.

The total bit length per individual is calculated as:

$$\text{Total Bits} = 98 \times 12 = 1176 \text{ bits}$$

This ensures proper genetic encoding of ANN parameters.

The order of parameters will be like this,

1. Weights Between Input and Hidden Layer:

   w_11, w_12, ..., w_19, w_21, ..., w_29, ..., w_81, ..., w_89
2. Weights Between Hidden and Output Layer:

   w'_11, w'_12, ..., w'_18, w'_21, ..., w'_28
3. Thresholds:

   θ_1, θ_2, ..., θ_8, θ'_1, θ'_2

The training follows this sequential approach:

1. Start with Letter C.
2. Compute the ANN output.
3. Measure error based on expected vs. predicted values.
4. Adjust the weights based on error feedback.
5. Repeat for Letter T, H, and None, each using updated weight configurations.

Once the binary-coded weights and thresholds are converted to real values, the ANN performs the feedforward process:

1. Compute the weighted sum for each hidden neuron.
2. Apply the step activation function to determine binary hidden node outputs.
3. Compute the weighted sum for output neurons using hidden layer outputs.
4. Apply the step function again to finalize the ANN decision.

For each input combination, the ANN's output is compared with the desired output. The error is calculated using **sum of squared errors** (SSE):

$$\text{Error} = \sum (E_i - O_i)^2$$

Where:
- (E_i) is the expected output.
- (O_i) is the ANN's actual output.

The total error across all training samples is then used to compute the **fitness function**:

$$\text{Fitness} = \frac{1}{1 + \text{Total Error}}$$

Higher fitness values indicate better-performing ANN configurations.

4 Genetic Algorithm Implementation

Genotype-to-Phenotype Coding and Fitness Function Selection

Genotype-to-Phenotype Coding

- Each **individual** in the genetic algorithm represents a candidate ANN configuration.
- The **genotype** (chromosome) is encoded as a **binary string**, where each gene represents either:
    - **Weights** (between input-hidden and hidden-output layers).
    - **Thresholds** (for hidden and output neurons).
- Each variable (weight or threshold) is encoded using **12 bits** and later decoded to real values.

Decoding Process

To convert binary genes into real values:

$$\text{Real Value} = \text{Min Value} + \left(\frac{\text{Binary Value} \times (\text{Max} - \text{Min})}{2^{12} - 1}\right)$$

Where:

- **Min Value & Max Value** correspond to the range of weight or threshold.
- The binary value is obtained by converting the **12-bit string** to an integer.

Fitness Function Selection

The fitness function evaluates how well an ANN configuration performs in **recognizing letters** based on training data.

Fitness Calculation will be as follows:

1. Compute **total error**:

$$\text{Total Error} = \sum (\text{Expected Output} - \text{Actual Output})^2$$

2. Convert error into fitness:

$$\text{Fitness} = \frac{1}{1 + \text{Total Error}}$$

3. **Higher fitness values** correspond to ANN configurations with **lower error**, ensuring the GA selects and evolves better solutions over generations.

Parent Selection, Crossover, Mutation, and Ranking Process

Parent Selection (Tournament Selection)

Since the assignment specifies using the **best two individuals out of five** (for Group 02):

1. Randomly select **5 individuals** from the population.
2. Rank them based on **fitness** (higher fitness = better individual).
3. Choose the **best 2 individuals** as parents.

Crossover (1-Point Crossover)

Crossover Rate = 0.8, meaning 80% of selected pairs will perform crossover.

- A **single crossover point** is randomly selected in the binary chromosome.
- Genes beyond the crossover point are swapped between parents to create **2 offspring**.
- This operation ensures **genetic diversity** while retaining high-performing traits.

Mutation

Mutation Rate = 0.01, meaning each gene has **1% probability of flipping**.

- If a mutation occurs, **0 becomes 1**, and **1 becomes 0**.
- Mutation introduces **variability**, preventing convergence to suboptimal solutions.

Ranking and Selection of the Next Generation

After crossover and mutation:

1. **26 offspring** are produced from **13 pairs of parents**.
2. The **best 4 elite individuals** (top fitness from previous generation) are **retained**.
3. The new population consists of **26 offspring + 4 elites** = **30 individuals**.
4. The next generation is ranked, and the cycle repeats for **1000 generations**.

Initial Population Generation and Error Analysis

Initial Population Generation

1. Create **30 individuals**, each with a chromosome containing **98 parameters (encoded with 12 bits per gene)**.
2. Each weight and threshold are randomly initialized:
    - **Weights range:** [-1, 1]
    - **Thresholds range:** [0, 1]
3. Convert the **binary representation** into **real phenotype values** for ANN processing.

Error Analysis

1. Run feedforward propagation on training inputs **(Letters C, T, H, None)**.
2. Compare **ANN predicted outputs** vs. **expected outputs**.
3. Calculate **total error** using the squared error formula.
4. Track error progression over **multiple generations**, ensuring improvement.
5. After training, validate ANN performance on **unseen test cases**.

ANN Weight Adjustment Method

- **Assignment Method:** Adjust weights after training each letter individually.
- **Implemented Method:** Train ANN using all letters per generation, then update weights via genetic operations.
- **Comparison & Justification:** Using a genetic algorithm allows **parallel optimization across all training samples**, leading to better convergence.

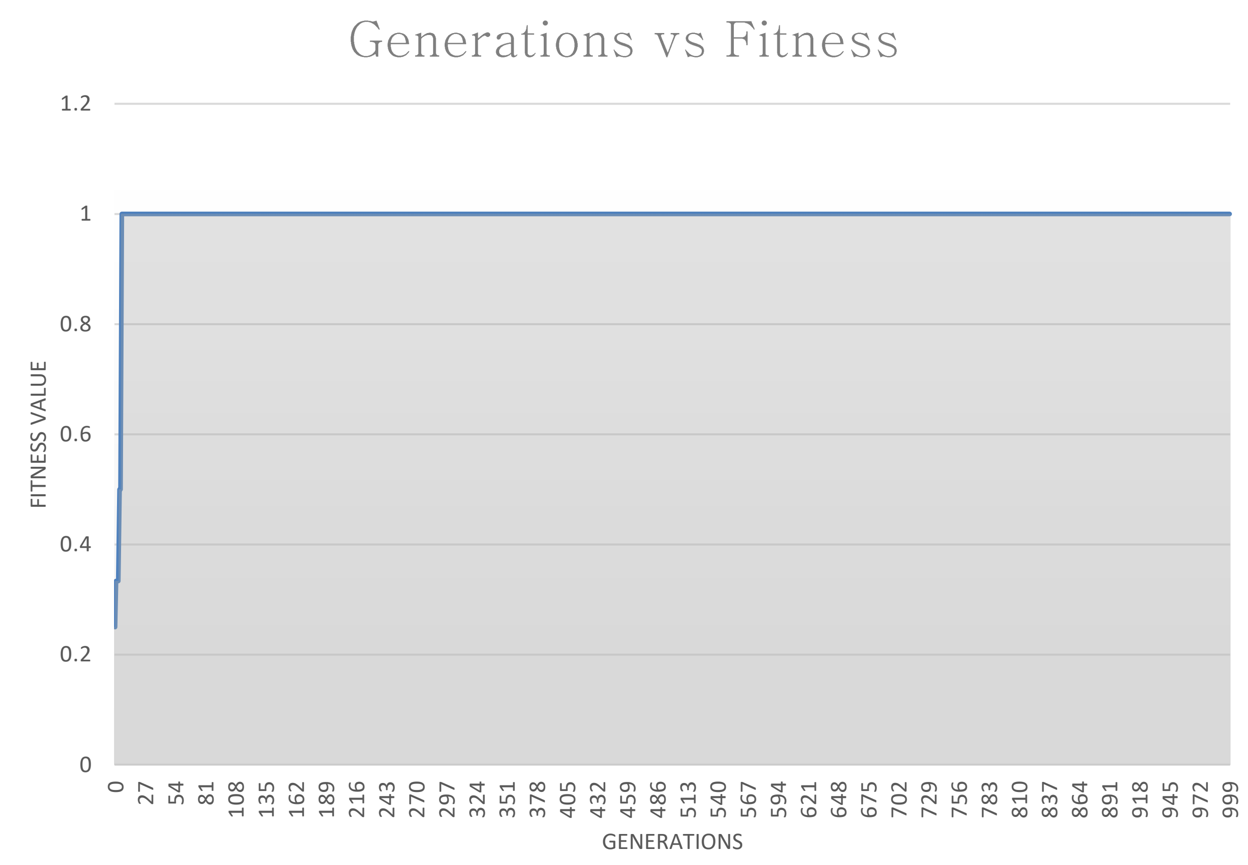
5 Training Results & Analysis

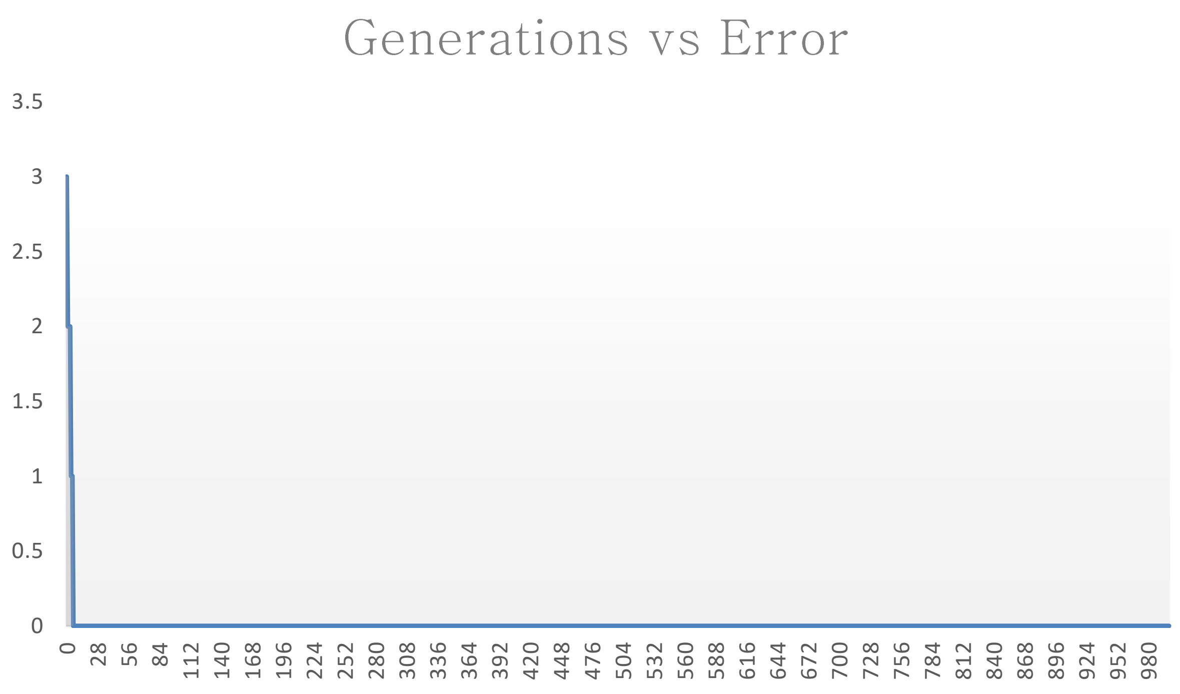
During the ANN training process, weightages and thresholds evolved across **1000 generations** through **genetic optimization**. The training was conducted using a **population size of 30 individuals**, with tournament selection, crossover, and mutation applied to refine the neural network’s ability to recognize letters **C, T, H, and None** in various orientations.

The results from selected generations (e.g., 1, 200, 400, 800, and 1000) were analyzed to track improvements in **fitness values, error reduction, and ANN accuracy**.

A detailed log of trained **weightages and threshold values** for these five generations has been **attached as an annex** in the report. This annex provides a tabulated format showcasing how the genetic algorithm refined ANN parameters over multiple generations.

Additionally, a **fitness vs. generations graph** and **error analysis comparison** have been included to visualize ANN performance trends.


Generations vs Fitness
FITNESS VALUE
GENERATIONS
1.2
1
0.8
0.6
0.4
0.2
0
0
27
54
81
108
...
972
999


Generations vs Error
3.5
3
2.5
2
1.5
1
0.5
0
0
28
56
84
...
952
980

6 Validation & Testing

Once the ANN training process was completed, it was necessary to test its performance using **new input models** that were not included during training. This step aimed to evaluate the generalization capability of the ANN by applying trained weights and thresholds to unseen patterns.

The input models used for testing were obtained from **Annex A** of the assignment document and included variations of letters **C, T, H, and None** in different orientations. No additional training was conducted during this phase.

The results of the ANN predictions were recorded in a **tabulated format**, comparing expected outputs with computed outputs and evaluating the **validation errors**.

A detailed tabulated form of **new input models and their results** has been **attached as an annex** in this report, providing a structured view of ANN behavior during testing.

Pattern	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	*Output Node 1*	*Output Node 2*	*Expected Outcome 1*	*Expected Outcome 2*	*Total Error*
C	1	1	0	0	1	0	1	1	0	1	0	1	0	0
	0	0	0	1	1	1	1	0	1	1	0	1	0	0
T	0	1	0	0	1	0	1	1	1	0	1	0	1	0
	0	0	1	1	1	1	0	0	1	0	1	0	1	0
H	1	0	1	1	1	1	1	0	1	1	1	1	1	0
	1	0	1	1	1	1	1	0	1	1	1	1	1	0
None	1	0	0	1	0	0	1	0	0	0	0	0	0	0
	1	1	1	0	1	0	0	1	1	0	1	0	0	1

7 Discussion & Learning Outcomes

A major challenge during the implementation was ensuring correct **manual validation of feedforward outputs** before automating training. Organizing weights, thresholds, and verifying each step using logical functions in Excel was tedious but essential to avoid early errors.

Building a fully functional ANN in Excel came with limitations, especially in **tracking hidden and output neuron activations**. Debugging formula logic was time-consuming, but it deepened my understanding of neural flow and activation behavior.

Despite these hurdles, the **Genetic Algorithm successfully improved ANN performance** over 1000 generations. Weights and thresholds were refined, reducing training errors and improving recognition for **C, T, and H**. However, the letter **"None" was consistently misclassified as "T"**, indicating partial success.

This misclassification suggests **overfitting**, where the ANN learned to expect structural features—even for blank patterns. With "None" having no active pixels, it may have been interpreted as a weak form of "T".

The lectures provided a strong **theoretical foundation** for this project, especially in understanding:

- How **artificial neurons** work through weighted summations and activation functions.
- The **structure of a feedforward neural network**, from input to hidden to output layers.
- The **working of Genetic Algorithms**, including fitness evaluation, selection, crossover, and mutation processes.
- How to design a basic **fitness function** based on prediction accuracy.

Beyond lectures, this assignment forced me to dive deeper into **manual ANN implementation and debugging**, which gave me insights that a pure code-based approach would have hidden. Specifically:

- I learned how to implement a complete **feedforward pass manually in Excel**, using SUMPRODUCT and IF logic to mimic neuron behavior.
- I gained experience in **tracking individual node behavior**, allowing me to validate where errors occur in hidden or output layers.
- I now appreciate the importance of **data representation**, especially how the None class might need better encoding or separation from active letter patterns.
- I understood how different **weight combinations and thresholds affect classification**, and how small changes can drastically influence output.
- Managing and interpreting results across **1000 generations of evolution** helped me understand convergence, stagnation, and the need for diversity in GA populations.

Through this task, I deepened my understanding of **how ANN nodes interact**, how **feedforward works numerically**, and how a Genetic Algorithm searches for better solutions in a single assignment.

Overall, this assignment was a perfect balance of theory, logic, and manual precision-proving that even without automation, a well-structured neural system can evolve and learn.