Assignment Report ANNs

Fundamentals of Intelligent System

Name : I Wayan Firdaus Winarta Putra
NRP : 5023231064

DEPARTEMEN TEKNIK BIOMEDIK
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS
INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA
2024

1. Theoretical Background

Artificial Neural Networks (ANNs), also referred to as parallel distributed processing systems or connectionist models, are computational models inspired by the architecture and functional principles of biological neurons in the human brain. The primary goal of ANN models is to mimic the brain's cognitive processing capabilities in order to enhance a system's ability to solve complex problems that are difficult to address using conventional algorithmic approaches. The human nervous system comprises an intricate network of interconnected neurons, with the brain serving as the central processing unit. Communication between neurons occurs through electrical impulses known as action potentials, and information is transmitted at synapses, where neurotransmitters bridge the gap between presynaptic and postsynaptic neurons.

One of the earliest and most influential models in the development of artificial neural networks is the McCulloch-Pitts neuron model, proposed by Warren McCulloch and Walter Pitts in 1943. This model introduces a simplified representation of a neuron that receives multiple binary inputs (0 or 1), computes a weighted sum of those inputs, and generates a binary output based on a fixed threshold value. The structure of the network is assumed to be fixed, and the signal transmission includes a delay component. The binary output behavior of this model allows it to simulate basic logical functions such as AND, OR, and NOT gates, forming the foundation for more complex neural network architectures.

Neurons can exhibit either excitatory or inhibitory behavior depending on the nature of their synaptic connections. Excitatory synapses promote the depolarization of the postsynaptic neuron, thereby facilitating signal propagation, while inhibitory synapses cause hyperpolarization, which suppresses the activation of the neuron. This biological mechanism is conceptually replicated in ANN models through positive and negative weight values that influence the activation state of artificial neurons.

A standard neural network architecture consists of three primary components: an input layer, one or more hidden layers, and an output layer. The input layer receives a fixed number of binary values representing the input vector, where each input node corresponds to one feature or signal. These input values are directly connected to each neuron in the hidden layer through weighted links, forming a fully connected structure. Each neuron (or perceptron) in the hidden layer calculates a weighted sum of its inputs, followed by a threshold activation function that determines whether the neuron becomes active. The hidden layer serves as a transformation layer that extracts patterns or performs intermediate computations based on the weight configuration. The output layer receives input from the hidden layer and generates the final output prediction based on the overall activation pattern.

Mathematically, the output of a neuron can be represented as:

$$y = f\left(\sum_{i=1}^{n} \text{weight[i][j] inputSignal[i] - threshold} \right)$$

Weight initialization is a fundamental aspect of neural network design, as it determines how the input signals are propagated through the network and significantly influences the final output behavior. In this specific implementation, the weight matrix is constructed using an alternating pattern, where the sign of the weights alternates based on a frequency determined by powers of two. This deterministic pattern reflects logical structures and ensures that each perceptron in the hidden layer receives a unique combination of weighted inputs.

The forward pass is a key computational process in feedforward neural networks, where data flows in one direction from the input layer to the hidden layer(s), and finally to the output layer. During this process, each neuron computes a dot product between the input vector and its corresponding weight vector, resulting in an intermediate sum. This sum is then evaluated using a threshold function, typically a step function, which produces a binary output (1 or 0) based on whether the sum exceeds a predefined threshold. In the output layer, a final decision is made by aggregating the outputs of the hidden layer, often using majority voting or another rule-based approach. This mechanism enables the network to perform classification or pattern recognition tasks effectively without requiring iterative learning or backpropagation.

2. Results and Data Analysis

The first step is to decide how many input neurons and perceptrons are needed. If we have only 2 inputs, we can use 4 perceptrons. If we have 3 inputs, we can use 8 perceptrons. In general, for n input neurons, we can determine the number of required perceptrons using the formula :

$number\ of\ perceptrons\ =\ 2^{n}$. So, for a case with 11 input neurons, we use: $2^{11}\ =\ 2048\ perceptrons$. Once the number of perceptrons is decided, the next step is to initialize the weight matrix between the input and the hidden layer. The weights are initialized using a block-wise alternating pattern based on the input index *i* The weight from input *i* to perceptron *j* is determined using the equation:

$$W[i][j]\ =\ (-\ 1)^{[\frac{j}{2^{i}}]\ \%2}$$

This equation ensures that each row of the weight matrix (corresponding to each input neuron) alternates between +1 and -1 in a repeating pattern, with increasing block sizes as *i* increases.

this will be the example when using 4 input :

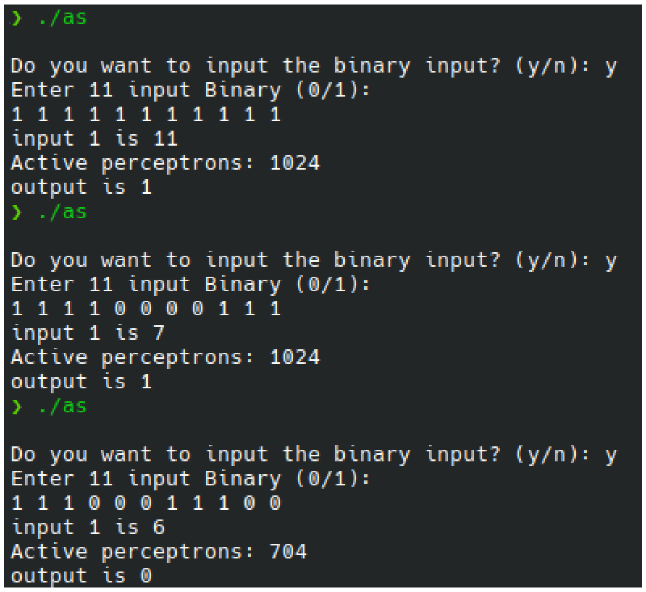
perceptron (16)	input 0	input 1	input 2	input 3
0	-1	-1	-1	-1
1	1	-1	-1	-1
2	-1	1	-1	-1
3	1	1	-1	-1
4	-1	-1	1	-1
5	1	-1	1	-1
6	-1	1	1	-1
7	1	1	1	-1
8	-1	-1	-1	1
9	1	-1	-1	1
10	-1	1	-1	1
11	1	1	-1	1
12	-1	-1	1	1
13	1	-1	1	1
14	-1	1	1	1
15	1	1	1	1

so it can initialize forming the input size then it goes to hidden layer that are computed for each perceptron. After initializing the weights, we calculate the hidden layer weighted sum for each perceptron using:

$$h_{j}\ =\ \sum_{i=0}^{n-1}\ x_{i}W_{ij}$$

then need to applies threshold to determine the output each perceptron while input get sum with weight then the output of the weight sum function will comparing with theta for example theta = 1, hidden_layer_sum = [3,-2,0,1] the threshold will apply 1 0 0 1 because perceptron -2 and perceptron 0 is not greater equal than the threshold(theta). this converts continuous weighted sums into discrete 0/1 values, simplifying the decision-making process. then last is output layer that is final_output, input for this is hidden layer output vector. the function calculate all the values sum while each 1 in hidden layer output represents an activate perceptron. the final output is determined by comparing sum of all the values in hidden layer output to a threshold while the threshold is 2^n / 2. this function mimic simple form of ensemble learning, where the "wisdom of the crowd" determines the final decision.

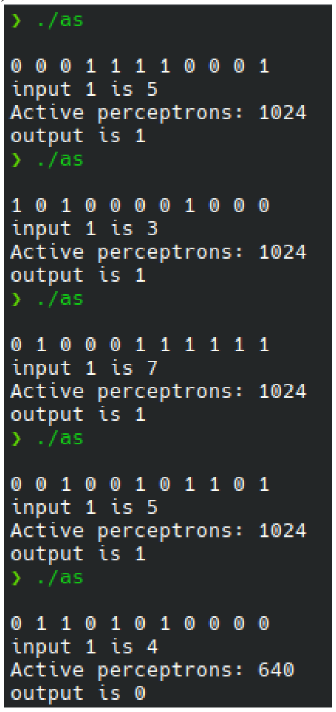
result of the program is like this



```
❯ ./as

Do you want to input the binary input? (y/n): y
Enter 11 input Binary (0/1):
1 1 1 1 1 1 1 1 1 1 1
input 1 is 11
Active perceptrons: 1024
output is 1
❯ ./as

Do you want to input the binary input? (y/n): y
Enter 11 input Binary (0/1):
1 1 1 1 0 0 0 0 1 1 1
input 1 is 7
Active perceptrons: 1024
output is 1
❯ ./as

Do you want to input the binary input? (y/n): y
Enter 11 input Binary (0/1):
1 1 1 0 0 0 1 1 1 0 0
input 1 is 6
Active perceptrons: 704
output is 0
```

(user Input binary)

for the test case program(random input)



```
❯ ./as

0 0 0 1 1 1 1 0 0 0 1
input 1 is 5
Active perceptrons: 1024
output is 1
❯ ./as

1 0 1 0 0 0 0 1 0 0 0
input 1 is 3
Active perceptrons: 1024
output is 1
❯ ./as

0 1 0 0 0 1 1 1 1 1 1
input 1 is 7
Active perceptrons: 1024
output is 1
❯ ./as

0 0 1 0 0 1 0 1 1 0 1
input 1 is 5
Active perceptrons: 1024
output is 1
❯ ./as

0 1 1 0 1 0 1 0 0 0 0
input 1 is 4
Active perceptrons: 640
output is 0
```

(using random seed for randomize the input)

3. Conclusion

This neural network employs a structured architecture where the number of hidden layer perceptrons grows exponentially with the input size, following the rule 2^n perceptrons for n inputs (e.g., 11 inputs into $2^{11} = 2048$ perceptrons). The weights between the input and hidden layers are initialized in alternating blocks of -1 and +1, creating a pattern that encodes all possible input combinations. For instance, with 2 inputs, weights alternate every 1 column for the first input and every 2 columns for the second. During computation, each hidden layer perceptron calculates a weighted sum of inputs, applies a threshold (e.g., $\theta = 1$) to produce binary outputs (0 or 1), and these results are aggregated at the output layer. The final decision uses majority voting: if at least half the perceptrons activate (e.g., ≥ 1024 out of 2048), the output is 1; otherwise, it is 0.