World Journal of Engineering Research and Technology
WJERT
www.wjert.org
SJIF Impact Factor: 5.924

ARTIFICIAL NEURAL NETWORKS

Raj Chauhan*

Kunskapsskolan Gurgaon, India.

Article Received on 07/06/2022 **Article Revised on 28/06/2022** **Article Accepted on 18/07/2022**

***Corresponding Author**
Raj Chauhan
Kunskapsskolan Gurgaon, India.

The artificial neural network (ANN), or simply neural network, is a data processing system consisting of a large number of simple, highly interconnected processing elements evolved from the idea of simulating the human brain. Neural networks are often capable of doing things which humans do well but which conventional computers find difficult to emulate. Neural networks have emerged in the past few years as an area for research, development and application to a variety of real world problems. Neural networks exhibit characteristics and capabilities not provided by any other technology. Examples include natural language processing, humanlike handwriting, reading typewritten text, face recognition, medical imaging, weather and load forecasting, modeling complex systems that cannot be modelled mathematically and many more.

INTRODUCTION

The Artificial Neural Network is one of the most recognized and well known algorithms associated with Artificial Intelligence. It can be used to emulate humanlike performance in many tasks.

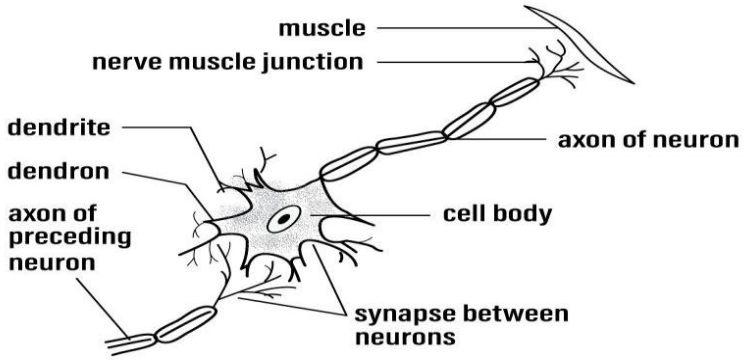
This article attempts to familiarize with the basic concepts of Artificial Neural Networks functioning.

Basic principles of neural networks

Artificial Neural Networks simulate the thinking and processing procedures of the human brain by modeling the network of neurons in the human brain. Basic components of a neuron are a cell body, a stem like structure called the axon and branch like structures called dendrites Fig.1. The axon produces electrical pulses emitted by the neurons. Dendrites

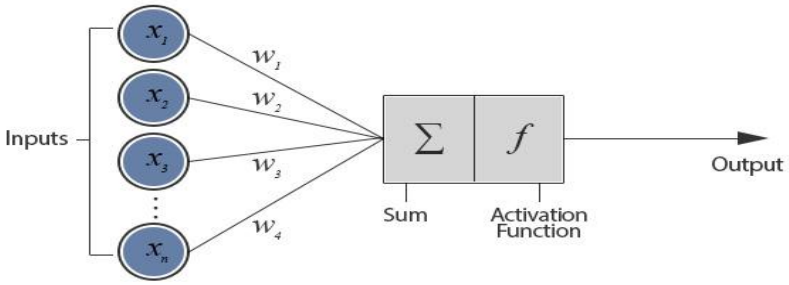
receive input from neurons by means of specialized contacts called synapses, which act as weights to the input information. The neuron is fired only when the weighted sum of the inputs is above a certain threshold. Information received by neurons can be processed in parallel, series or a combination of both. For some stimuli, the reaction of the brain is typecast: for a certain input, only a specific output is obtained. A lot of research and development have been done on simulation of processes in neural networks that can match inputs to required outputs and incorporate variations in input patterns to account for the corresponding output patterns.

Neurons can interact in feed-forward, feedback, fully connected or partially connected. The connections of the neurons in the brain as well as in the neural network model are shown in Fig. 2. The feedback path influences the nature of adaptivity and trainability. If the selection of weights and thresholds in a neural net is automated, then this could be thought of as a learning mechanism. This learning capability of neural nets distinguishes them from conventional computer software. Neural nets show potential for ever-improving performance through dynamic learning.



Biological neurons

Fig. 1: (Source: Google).



Artificial neurons

Fig. 2: (Source: Google)

A single neuron in a neural net that can be represented as in Fig.2, where x_i is the input, w_i is weight carried by the input x_i and the box represents the linear combination of weighted inputs.

The output of box is passed through a non-linear function called the activation function Fig.3. For nerve connections in the representation of Fig.1 to be an accurate approximation to an actual neuron, all neurons connected together must form a stable system. This can be achieved by modifying the weights.

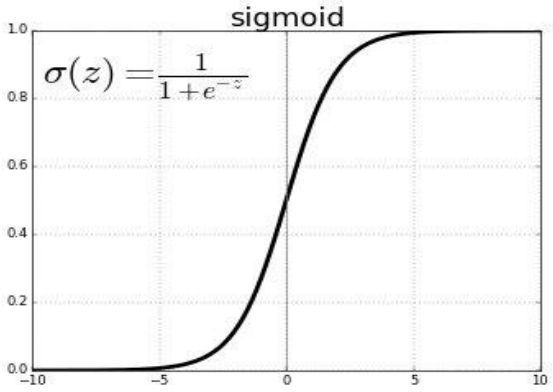


Fig. 3: (Source: Google).

Neural nets can be viewed as either of the following.

- A set of non-linear differential equations
- A non-linear transformation between input and output.

Data processing in neural nets is non-algorithmic; therefore the influence of approximations of mathematical modeling is reduced to a minimum. Further, neural nets are capable of handling uncertainties so that results obtained through trained neural nets, even with partial inputs, may be very close to the results with complete inputs. Depending on the neuron interaction, neural networks can be classified as.

- Feed forward neural networks
- Feedback neural networks

Feed forward neural networks

Here neurons are arranged in directed graphs. Inputs are applied to the layers and outputs are collected. Stability is not a problem here because these networks are loop free. The computation time is the time required for signals to propagate and output to settle.

Feedback neural networks

Here, neurons are arranged in the form of undirected graphs. The connections in this case are symmetrical and bidirectional. Feedback neural network models are sequential or asynchronous. Here the system is initialized and then it evolves to a final state in the course of time. The stability of this type of neural net is analyzed with the help of energy functions defined in terms of states of neurons, weights and thresholds.

Training rules or algorithms

There are several algorithms for training of neural networks. One popular training rule for multilayer neural network is the back propagation algorithm. This algorithm is able to model nonlinear relationships between the inputs and outputs. Figure x shows a multilayer network. In a multilayer network, the first set of neurons connecting to the inputs serve only as distribution points and perform no input summation.

For each j^{th} neuron or node in the hidden layer, or the output layer, the input is a weighted sum (the sum of the inputs x_i multiplied by their respective weights w_{ji}) given by,

$$h_j(\bar{x},\bar{w}) = \sum_{i=0}^{n} x_i w_{ji}$$

Where 'i' refers to the neuron in the preceding layer, and w_{ji} is the connection weight from neuron i to neuron j. The neuron output is a function of the neuron input which can be written as:

$$O_j(\bar{x},\bar{w}) = f(h_j(\bar{x},\bar{w}))$$

Now this function is a non-linear function as there is usually no linear relationship between the input and output of the neuron. This is the activation function. The primary role of the activation function is to transform the weighted sum of input from the node into an output value to be fed to the next hidden layer or into an output. There are a number of activation functions that can be used. The most common activation function is the sigmoidal function:

$$\sigma(z) = 1 / (1+e^{-z})$$

The node output is therefore given by,

$$O_j(\bar{x},\bar{w}) = 1 / (1 + e^{-h_j(\bar{x},\bar{w})})$$

There are many variations to the exponent of 'e' in the given node output expression in terms of adding a threshold value and some other constants in various implementations, but the general form remains the same.

The range of the sigmoidal function is between 0 and 1. For a large negative domain, the value is close to 0, for a zero domain value is 0.5 and for large positive domain it is close to

1. This allows a smooth transition between the low and high outputs of the neuron. We can see that the output depends only on the activation, which in turn depends on the values of the inputs and their respective weights.

The purpose of the training process is to get a desired output when certain inputs are given. Since the error is the difference between the actual and the desired output, the error depends on the weights, and we need to adjust the weights in order to minimize the error. We can define the error function for the output of each neuron as.

$E_j(\bar{x},\bar{w}, d) = (1/2)(O_j(\bar{x},\bar{w}) - d_j)^2$

Sum of squares of error is taken as it is always positive. It is multiplied by ½ to help keep taking the derivative of the error more convenient.

The error is dependent on the inputs, outputs, and the weights, the weights are therefore adjusted using the method of steepest decent as,

$\Delta w_{ji} = -\eta(\partial E / \partial w_{ji})$

The η in front of the gradient is called the learning rate. In order to take small steps towards minimizing the error and not jump past the minima, we need to define a small learning rate.

During the training phase the weights are adjusted until the output from the network is within desirable error limits.

Implementation and understanding

Let us take a simple ANN structure consisting of an input layer with two input nodes, a hidden layer with two nodes, and an output layer with one node.

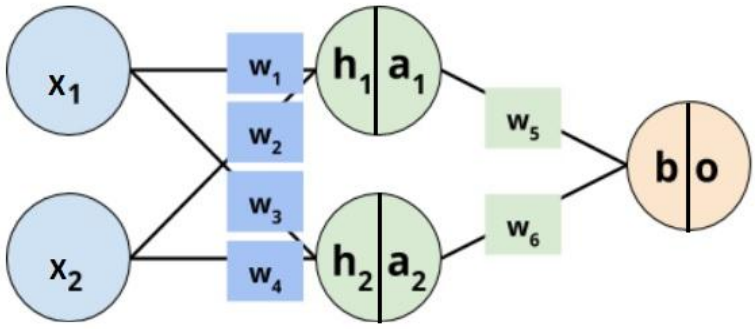


Fig. 4.

The steps to code for a simple ANN can be broken into four parts:

1. Creating the Network
2. Forward Propagation

3. Back Propagation
4. Testing

The activation function we will be using is the sigmoid function (Fig. 3). Activation function is used to make neural network able to map non-linear relationships between inputs and outputs.

Let x_1, x_2, h_1, h_2, and b be the inputs to the input layer, hidden layer and the outer layer of the neural network.

Feed Forward Flow:

$h_1 = x_1.w_1 + x_2.w_2$ - - - eq. (i)

$h_2 = x_1.w_3 + x_2.w_4$ - - - eq. (ii)

Each input connects to every node in the hidden layer.

In the hidden layer there is the activation function that is applied to each of the hidden layer inputs to send an input to the output layer.

$a_1 = \sigma(h_1)$, $a_2 = \sigma(h_2)$ - - - eq. (iii)

a_1, a_2 form the outputs of the hidden layer.

Then the same logic applied in eq.(i) and eq.(ii) is applied as we move from the hidden layer to the output layer.

$b = a_1.w_5 + a_2.w_6$ - - - eq. (iv)

And finally the output is formed by applying the activation function to the output layer's input.

$o = \sigma(b)$

Substituting value of b in terms of inputs i and weights w, we get

$o = \sigma(\sigma(x_1.w_1 + x_2.w_2).w_5 + \sigma(x_1.w_3 + x_2.w_4).w_6)$ - - - eq. (v)

This is the algebraic expression for the output in terms of the inputs and weights in forward propagation. This can be easily coded in the forward propagation module.

Now we shift focus to the modification of the weights during back propagation:

Starting with w_6 we have

$w_{6new} = w_{6old} - \eta(\partial E / \partial w_6)$

Now, $\partial E / \partial w_6 = \partial E / \partial o \,.\, \partial o / \partial b \,.\, \partial b / \partial w_6$

Using the aforementioned equations we get,

$\partial E / \partial w_6 = \partial(0.5(o - d)^2) / \partial o \,.\, \partial(\sigma(b)) / \partial b.\, \partial b / \partial w_6$

We know $\sigma'(z) = \sigma(z)(1 - \sigma(z))$

Let (actual output – desired output) = $(o - d) = \Delta$

This gives,

$w_{6new} = w_{6old} - \eta \{\sigma(b).(1 - \sigma(b)).\sigma(h_2).\Delta\}$

$w_{5new} = w_{5old} - \eta \{\sigma(b).(1 - \sigma(b)).\sigma(h_1).\Delta\}$

$w_{4new} = w_{4old} - \eta \{\sigma(b).(1 - \sigma(b)).\sigma(h_2).\Delta.(1 - \sigma(h_2)).x_2.w_6\}$

$w_{3new} = w_{3old} - \eta \{\sigma(b).(1 - \sigma(b)).\sigma(h_2).\Delta.(1 - \sigma(h_2)).x_1.w_6\}$

$w_{2new} = w_{2old} - \eta \{\sigma(b).(1 - \sigma(b)).\sigma(h_1).\Delta.(1 - \sigma(h_1)).x_2.w_5\}$

$w_{1new} = w_{1old} - \eta \{\sigma(b).(1 - \sigma(b)).\sigma(h_1).\Delta.(1 - \sigma(h_1)).x_1.w_5\}$

This is fairly straight forward to code in the back propagation method to make changes to the weights. The forward propagation and the back propagation are put in a loop until the weights are suitably modified to arrive at an acceptable error limit.

A Python program was created to implement the network. The forward propagation was run, the results printed, and the back propagation was run in a loop of 100,000.

Following was the result obtained.

for the 0 epoch, the output is 0.488127

for the 1 epoch, the output is 0.488290

for the 2 epoch, the output is 0.488453

for the 3 epoch, the output is 0.488616

for the 4 epoch, the output is 0.488779

for the 5 epoch, the output is 0.488942

for the 6 epoch, the output is 0.489105

. . .

We see that initially the output is quite far from the desired result of 1.

for the 99993 epoch, the output is 0.981085

for the 99994 epoch, the output is 0.981085

for the 99995 epoch, the output is 0.981086

for the 99996 epoch, the output is 0.981086

for the 99997 epoch, the output is 0.981086

for the 99998 epoch, the output is 0.981086

for the 99999 epoch, the output is 0.981086

Process finished with exit code 0

As the iterations increase, the output converges towards 1.

CONCLUSION

We have learnt how to create a simple neural network. We can now apply this knowledge to implement more complex neural network structures, with use of various biases, thresholds, and abruptness factor. Other activation functions and their specific applications can also be explored. This article paves a path to exploring the vast area of Artificial Neural Networks.

SOURCES

1. https://ieeexplore.ieee.org/document/483329
2. http://wsc10.softcomputing.net/ann_chapter.pdf
3. https://www.geeksforgeeks.org/activation-functions-neural-networks/
4. https://machinelearningmastery.com/weight-initialization-for-deep-learning-neural-networks/
5. https://www.v7labs.com/blog/neural-networks-activation-functions
6. https://hmkcode.com/ai/backpropagation-step-by-step/
7. http://neuralnetworksanddeeplearning.com/chap2.html
8. https://pathmind.com/wiki/backpropagation
9. https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/