Single Layer Artificial Neural Network: Perceptron

Samandarov Erkaboy Karimboyevich
Teacher of computational mathematics and information systems department of applied mathematics and intellectual technologies faculty of National university of Uzbekistan named after MirzoUlugbek

Abdurakhmonov Olim Nematullayevich
Senior teacher of computational mathematics and information systems department of applied mathematics and intellectual technologies faculty of National university of Uzbekistan named after MirzoUlugbek

Abstract: In this paper, we overview the perceptron that is simple form of artificial neural network. In particular, we will consider the structure of the perceptron and its features. Moreover, we will overview input values, weights, basis, output value, activation function of perceptron and the program in Python that implemented as well as we consider the use cases application of perceptron.

Keywords: Perceptron, artificial neural network, activation function, bias, weight, artificial intelligence.

INTRODUCTION

Whatever event inside a computer happens in a virtual world that is invisible to the human eye. What happens in the virtual world is modeled from the real world in which people live. The virtual world is digitally displayed objects, properties, algorithms of real world. Therefore, although this virtual world is invisible to the human eye, the events of virtual world occur in the same way as in the real world in which we live. In this paper, the perceptron that we are observed is similar to the

eyes of human. The perceptron is simple form of artificial neural network. Perceptron is also known as a single layer neural network.

Figure 1 illustrates a neuron (biological neuron) in the human eyes.

we briefly describe the components of biological neuron. First component of human eyesdendrite, is a branched process of a neuron that receives information through chemical synapses from the axons of other neurons and transmits it through an electrical signal to the body of the neuron from which it grows.

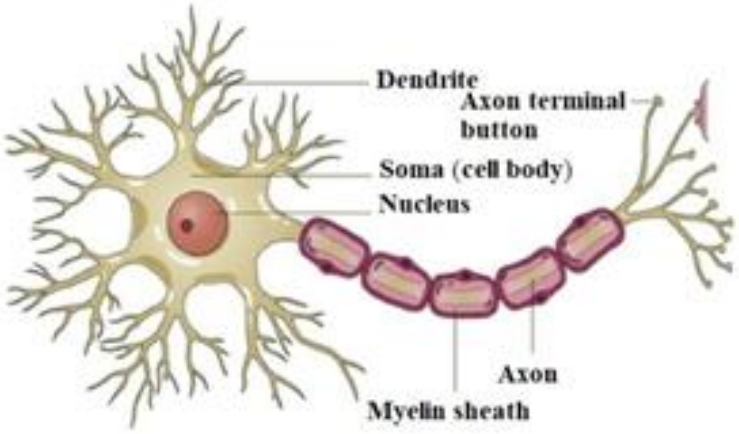

Figure 1

Next component is known as soma (cell body), the parts of an organism other than the reproductive cells. Third component nucleusthe central and most important part of an eye, forming the basis for its activity and growth. Myelin she a this substance that forms the myelin sheath of nerve fibers.

Axon is the fifth component. This componentneurite, along which nerve impulses travel from the cell body to innervated organs and other nerve cells.

Last component the axon terminal, also known as the synaptic bouton and terminal bouton, is the most distal portion of a neuron axon and is critical for neural communication.

Below we consider the perceptron that is the component of artificial intelligence based on biological neurons in the human eyes.

Perceptron (English perceptron from Latin. perceptio - perception; it. Perzeptron is a mathematical or computer model of information perception by the brain (cybernetic model of the brain), proposed by Frank Rosenblatt in 1958 and first implemented in the form of an electronic machine "Mark-1" in 1960. Perceptron became one of the first models of neural networks, and Mark-1 became the world's first neurocomputer.

As mentioned above biological neuron and it is components are depicted. Each component has own function and the components perform these functions. the functional principle of an artificial neural network is similar to the biological neuron's.

Thus, each component has a clearly defined function. Below we consider the perceptron which is a simple form of an artificial neural network. Figure 2 illustrates the perceptron that is used in artificial intelligence.

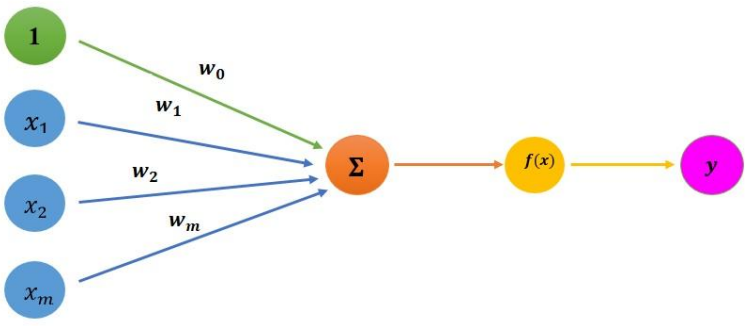

Figure 2

Each component of perceptron in figure 2 is considered in detail. $x_1, x_2 \ldots x_m$ input values. $w_1, w_2, \ldots w_m$ weights, w_0 bias, input values and weights are calculated based on the following formula

$$\sum_{i=1}^{n} x_i * w_i$$

Where n is number of input values and weights. After calculating sum $f(x)$ activation or step functions are used to create non-linear neural networks. These functions can change the value of neural networks to 0 or 1. that y The value received after the last step is the output value.

METHODS

A perceptron takes a vector of real –valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise.

More precisely, given inputs x_1 through x_n, the output $o(x_1, \ldots, x_n)$ computed by the perceptron is

$$o(x_1, \ldots, x_n) = \begin{cases} 1 \; if \; w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 \; otherwise \end{cases}$$

Where each $\boldsymbol{w_i}$ is a real-valued constant, or weight, that determines the contribution of input $\boldsymbol{x_i}$ to the perceptron output, $\boldsymbol{w_0}$ is a bias

We overview some of the following in relation to what is depicted in the figure 2 representing a neuron:

Step 1: Input signals weighted and combined as net input: Weighted sums of input signal reaches to the neuron cell through dendrites. The weighted inputs do represent the fact that different input signal may have different strength, and thus, weighted sum. This weighted sum can as well be termed as **net input** to the neuron cell.

Step 2: Net input fed into activation function: Weighted The weighted sum of inputs or net input is fed as input to what is called as activation function. The activation function is a non-linear activation function. The activation functions are of different types such as the following:

Unit step function;

Sigmoid function (Popular one as it outputs number between 0 and 1 and thus can be used to represent probability);

Rectilinear (ReLU) function;

Hyperbolic tangent;

The diagram below depicts different types of non-linear activation functions.

Figure 3 depicts following functions: a-Sigmoid function, b-Tanh function, c-ReLU function, d-LReLU function.

Step 3 – Activation function outputs binary signal appropriately: The activation function processes the net input based on the unit step (Heaviside**)** function andoutputs the binary signal appropriately as either 1 or 0. The activation function for perceptron can be said to be a unit step function. Recall that the unit step function, u(t), outputs the value of 1 when t >= 0 and 0 otherwise.

In the case of a shifted unit step function, the function u(t-a) outputs the value of 1 when t >=

a and 0 otherwise.

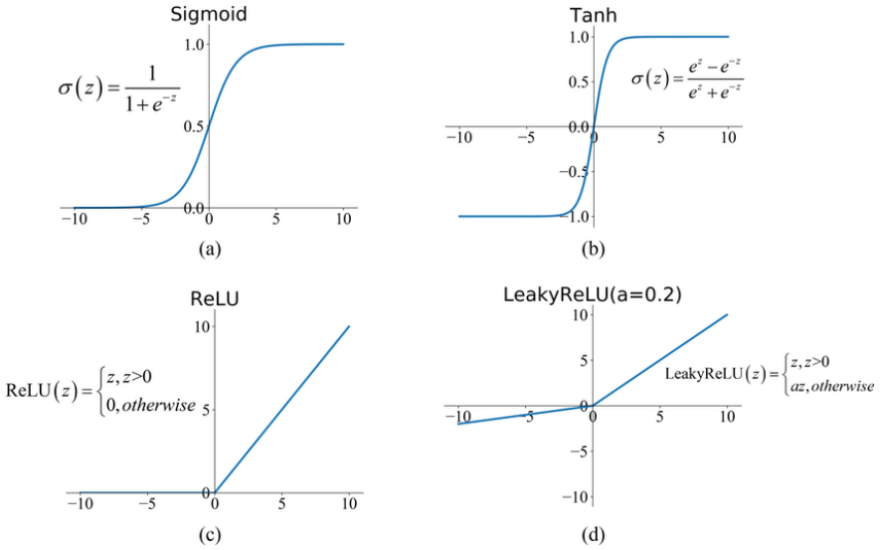

Figure 3

Step 3B – Learning input signal weights based on prediction vs actuals: A parallel step is a neuron sending the feedback to strengthen the input signal strength (weights) appropriately such that it could create an output signal appropriately that matches the actual value. The feedback is based on the outcome of the activation function which is a unit step function. Weights are updated based on the gradient descent learning algorithm. Here is my post on gradient descent. Here is the equation based on which the weights get updated:

$$w_i \leftarrow w_i + \Delta w_i$$

Where $\Delta w_i = \eta(t - o)x_i$

η −learning rate, t −target value, o −perceptron output, x_i −input value

The main goal of a perceptron is to make accurate classifications. To train a model to do this, perceptron weights must be optimizing for any specific classification task at hand.

The best weight values can be chosen by training a perceptron on labeled training data that assigns an appropriate label to each data sample (feature). This data is compared to the outputs of the perceptron and weight adjustments are made. Once this is done, a better classification model is created!

The first step in the perceptron classification process is calculating the weighted sum of the perceptron's inputs and weights.

To do this, multiply each input value by its respective weight and then add all of these products together. This sum gives an appropriate representation of the inputs based on their importance.

To increase the accuracy of a *perceptron's* classifications, its weights need to be slightly adjusted in the direction of a **decreasing** training error. This will eventually lead to minimized training error and therefore optimized weight values.

Each weight is appropriately updated with this formula:

$$weight = weight + (error * input)$$

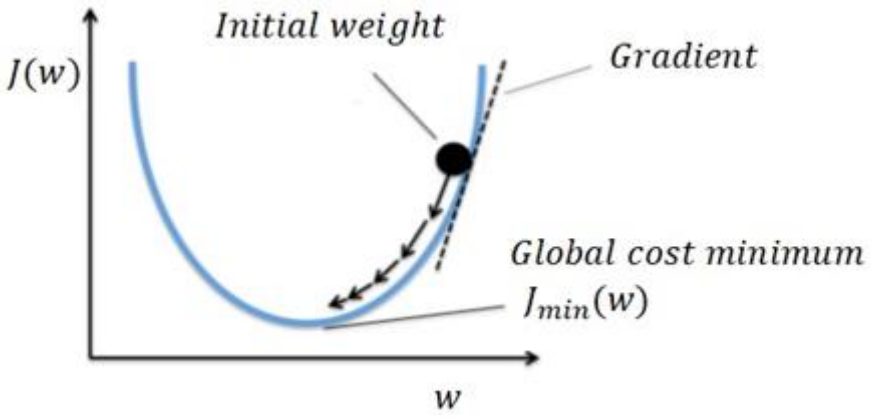

Figure 4

Perceptron – A single-layer neural network comprising of a single neuron

$$y = g(w_0 + \sum_{i=1}^{m} x_i w_i)$$

Where y −output value, g −non-linear activation function, w_0 −bias, x_i −input values, w_i −weights,

RESULTS

Since the output of a perceptron is binary, we can use it for binary classification, i.e., input values belong to only one of two classes. The classic examples used to explain what perceptrons can model are logic gates.

We consider the logic gates in the figure 5. A white circle means an output of 1 and a black circle means an output of 0, and the axes indicate inputs. E.g, when we input 1 and 1 to an AND gate, the output is 1, the white circle. We can create perceptrons that act like gates: they take 2 binary inputs and produce a single binary output.

However, perceptrons are limited to solving problems that are linearly separable. If two classes are linearly separable, this means that we can draw a single line to separate the two classes.

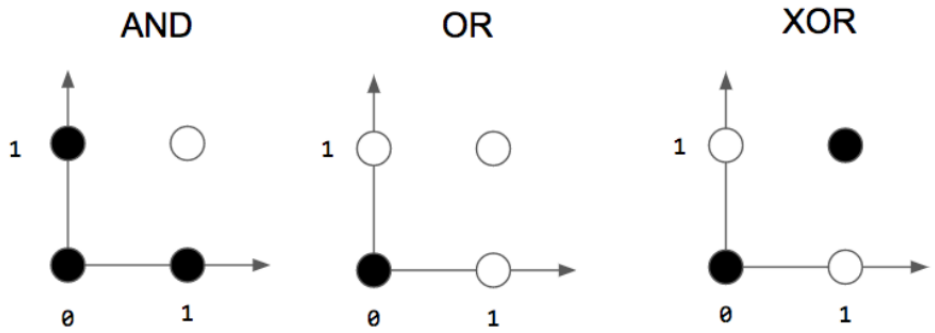

Figure 5

We can do this easily for the AND and OR gates, but there is no single line that can separate the classes for the XOR gate! This means that we can't use our single-layer perceptron to model an XOR gate.

An intuitive way to understand why perceptrons can only model linearly separable problems is to look the weighted sum equation (with the bias).

$$\sum_{i=1}^{N} W_i x_i + b$$

Above formula is very similar to the equation of a line. (Or, more generally, a hyperplane.) Hence, we create a line that everything on one side of the line belongs to one class and everything on the other side belongs to the other class.. This line is called the decision boundary, and, when we use a single-layer perceptron, we can only produce one decision boundary.

in practice, many problems are actually linearly separable. It can be shown that organizing multiple perceptrons into layers and using an intermediate layer, or hidden layer, can solve the XOR problem. This is the foundation of modern neural networks.

Single-Layer Perceptron Code

We will use object-oriented principles and create a class. In order to construct our perceptron, we need to know how many inputs there are to create our weight vector. The reason we add one to the input size is to include the bias in the weight vector.

importnumpy as np

class Perceptron(object):

"""Implements a perceptron network"""

def __init__(self, input_size, lr=1, epochs=100):

self.W = np.zeros(input_size+1)

 # add one for bias

self.epochs = epochs

self.lr = lr

We also need to implement our activation function. We can simply return 1 if the input is greater than or equal to 0 and 0 otherwise.

defactivation_fn(self, x):

#return (x >= 0).astype(np.float32)

return 1 if x >= 0 else 0

Finally, we need a function to run an input through the perceptron and return an output. Conventionally, this is called the prediction. We add the bias into the input vector. Then we can simply compute the inner product and apply the activation function.

def predict(self, x):

z = self.W.T.dot(x)

a = self.activation_fn(z)

return a

Now we can create a function, given inputs and desired outputs, carry out our perceptron learning algorithm. We keep updating the weights for a number of epochs, and iterate through the entire training set. We insert the bias into the input when performing the weight update.

Then we can create our prediction, compute our error, and perform our update rule.

```
def fit(self, X, d):
for _ in range(self.epochs):
for i in range(d.shape[0]):
x = np.insert(X[i], 0, 1)
y = self.predict(x)
e = d[i] - y
self.W = self.W + self.lr * e * x
```

Now that we have our perceptron coded, we can try to give it some training data and see if it works. One easy set of data to give is the AND gate. Here is a set of inputs and outputs.

```
if __name__ == '__main__':
 X = np.array([
[0, 0],
[0, 1],
[1, 0],
[1, 1]
 ])
d = np.array([0, 0, 0, 1])
perceptron = Perceptron(input_size=2)
perceptron.fit(X, d)
print(perceptron.W)
```

In just a few lines, we can start using our perceptron. At the end, we print the weight vector. Using the AND gate data, we should get a weight vector of$[-3, 2, 1]$. This means that the bias is -3 and the weights are 2 and 1 forx_1 andx_2, respectively.

To verify this weight vector is correct, we can try going through a few examples. If both inputs are 0, then the pre-activation will be$-3 + 0 * 2 + 0 * 1 = -3$. When applying our activation function, we get 0, which is exactly 0 AND 0. We can try this for other gates as well. Note that this is not the *only* correct weight vector. Technically, if there exists a single weight vector that can separate the classes, there exist an infinite number of weight vectors. Which weight vector we get depends on how we initialize the weight vector.

To summarize, perceptrons are the simplest kind of neural network: they take in an input, weight each input, take the sum of weighted inputs, and apply an activation function. Since they were modeled from biological neurons, they take and produce only binary values. In other words, we can perform binary classification using perceptrons. One limitation of perceptrons is that they can only solve linearly separable problems. In the real world, however, many problems are actually linearly separable. E.g, we can use a perceptron to mimic an AND or OR gate. However, since XOR is not linearly separable, we can not use single-layer perceptrons to create an XOR gate. The perceptron learning algorithm fits the intuition by Rosenblatt: inhibit if a neuron fires when it should not have, and excite if a neuron does not fire when it should have. We can take that simple principle and create an

update rule for our weights to give our perceptron the ability of learning.

DISCUSSION

Artificial neural networks are information processing systems that are inspired by our biological nervous system, such as how our brain processes information. Our human brain has around 100 billion neurons, and each neuron has a connection point between 1000 to 10,000. The human brain is designed to store information and extract it in a way where we can extract more than one piece of information from our memory whenever needed. The human brain is made up of thousands of powerful parallel processors.

Similarly, artificial neural networks have neurons placed similarly to the human mind. Each neuron is connected with other neurons with certain coefficients. When these networks are put through training, information gets passed through these connecting points, which helps the network learn. While artificial neural network is welcomed with open arms by most people, it does have its advantages as well as disadvantages.

Single layer can be used only for simple problems. However, its computation time is very fast.

Multi-layer is most of the neural networks expect deep learning. it uses one or two hidden layers. The main advantage is they can be used for difficult to complex problems. However, they need long training time sometimes.

Single-layer neural networks can also be thought of as part of a class of feedforward neural networks, where information only travels in one direction, through the inputs, to the output. Again, this defines these simple networks in contrast to immensely more complicated systems, such as those that use backpropagation or gradient descent to function. The Perceptron uses the class labels to learn model coefficients.

CONCLUSION

Perceptron mimics the neuron in the human brain.Perceptron is termed as machine learning algorithm as weights of input signals are learned using the algorithm.

Perceptron algorithm learns the weight using gradient descent algorithm. Both stochastic gradient descent and batch gradient descent could be used for learning the weights of the input signals.As a simplified neural network, perceptrons play a critical role in binary classification. A perceptron classifies data into two parts (0s and 1s)—a computer's primary language—binary. Because of that, perceptrons are also known as "linear binary classifiers."

REFERENCES

1. Minsky M., Papert S. Perceptrons. – 1969.
2. Rosenblatt F. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. – Cornell Aeronautical Lab Inc Buffalo NY, 1961.
3. Novikoff A. B. On convergence proofs for perceptrons. – STANFORD RESEARCH INST MENLO PARK CA, 1963.
4. Raiko T., Valpola H., LeCun Y. Deep learning made easier by linear transformations in perceptrons //Artificial intelligence and statistics. – PMLR, 2012. – С. 924-932.
5. Olazaran M. A sociological study of the official history of the perceptrons controversy //Social Studies of Science. – 1996. – Т. 26. – №. 3. – С. 611-659.
6. Marvin M., Seymour A. P. Perceptrons. – 1969.

7. Tattersall G. D., Foster S., Johnston R. D. Single-layer lookup perceptrons //IEE Proceedings F-Radar and Signal Processing. – IET, 1991. – Т. 138. – №. 1. – С. 46-54.
8. Chen H. C., Hu Y. C. Single-layer perceptron with non-additive preference indices and its application to bankruptcy prediction //International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. – 2011. – Т. 19. – №. 05. – С. 843-861.
9. Kanal L. N. Perceptron //Encyclopedia of Computer Science. – 2003. – С. 1383-1385.