TASK

Exploring Neural Networks

Visit our website

Introduction

WELCOME TO THE EXPLORING NEURAL NETWORKS TASK!

While the algorithms we've explored so far have laid the foundation for machine learning, a new era has dawned with the rise of neural networks. This shift is driven by the remarkable capabilities of neural networks, which outperform traditional approaches in many applications. In this task, you will learn about the building blocks of neural networks and how to build a simple feedforward neural network.

WHAT ARE NEURAL NETWORKS?

Inspired by the human brain, neural networks are computational models that mimic the interconnected network of neurons. These neurons act as information processing units, receiving signals via numerous dendrites and transmitting a single output signal through an axon. Similar to how our brains learn through experience, neural networks are trained on vast amounts of data, allowing them to identify patterns and make connections between information. This allows them to perform tasks like image recognition, language translation, and even creative writing. Interestingly, a typical human brain houses an estimated 100 billion neurons, each roughly 100 times smaller than a millimeter – a testament to the brain's intricate and compact design.

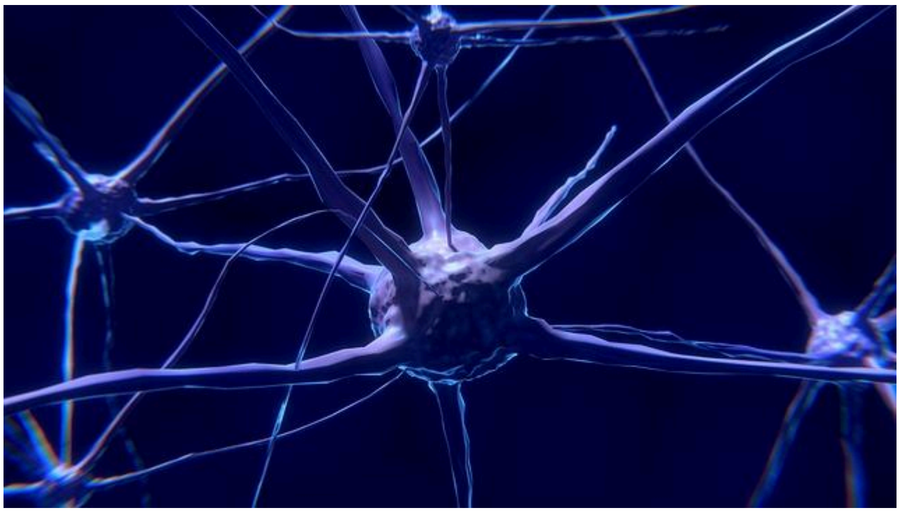


Image source: (Medical Xpress, 2019)

Inside a computer, there exists a minuscule switching device called a transistor that is roughly equivalent to a brain cell. The latest, most advanced microprocessors contain over 2 billion transistors and even a basic microprocessor has about 50 million transistors. These transistors are packed onto an integrated circuit that is smaller than a postage stamp.

This is where the comparison between computers and human brains ends. Brains and computers are otherwise structured in very different ways. Transistors in a computer are wired in simple, serial chains where each one is connected to maybe two or three others. Neurons, on the other hand, are densely interconnected in complex ways where each one is connected to thousands of others.

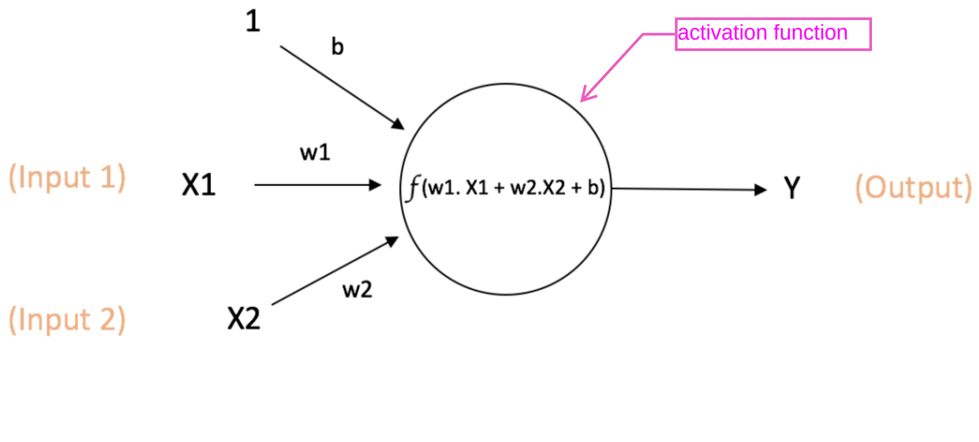
This structural difference between brains and computers causes them to "think" in completely different ways. Computers are designed perfectly for storing huge amounts of information and rearranging it in a number of ways according to the intricate sets of instructions we feed into them. Brains, on the other hand, learn slowly, but can spontaneously put information together in astounding new ways. People can recognise original patterns, forge new connections, and see the things they've learned in a completely different light.

For example, if you show a toddler pictures of cats and dogs and tell them which one is which, they'll soon learn to tell the difference. But writing a computer program that can identify cats and dogs in a series of pictures is very hard. This task, that is done effortlessly and almost unconsciously by the human brain, is an extremely daunting problem for a regular computer.

The advent of powerful neural networks changed this. A neural network, in a sense, simulates the network of neurons that make up a human brain. With these simulations, recognising cats and dogs in pictures, and many other tasks, have become solvable by computers.

THE BUILDING BLOCKS OF NEURAL NETWORKS

Similarly to how the brain makes use of interconnected neurons, neural networks are composed of interconnected artificial neurons — represented as nodes with weighted inputs and an output. At the node, the inputs are added together along with a constant (called a bias) and passed through an activation function to produce an output.



Output of neuron = Y= f(w1. X1 + w2.X2 + b)

Image source: (Ujjwalkarn, 2016)

The simplest type of neural network is called a feedforward neural network. Neurons are grouped to form 'layers', of which there are three main types: an **input layer, hidden layers**, and an **output layer**. The job of an input layer is to accept raw features from an external source and pass these onto the hidden layers; this layer does not perform any computation. Hidden layers perform the computation of the activation function from the inputs, and produce an output. This output can be passed on as inputs to other hidden layers, or to an output layer, in which final computations are performed to produce a prediction.

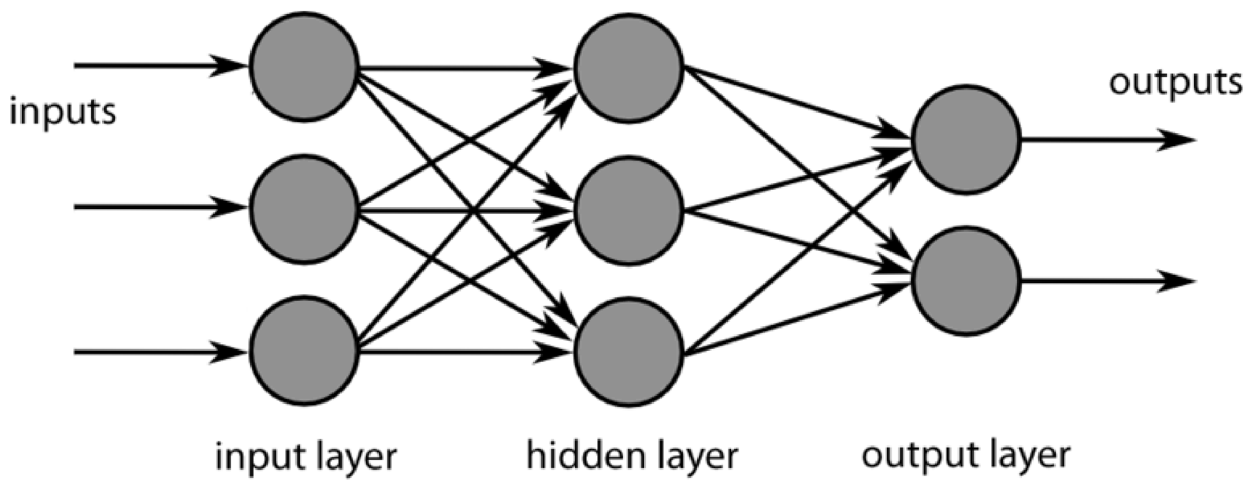


Image source: Wikimedia Commons

The feedforward neural network above has an input layer with three neurons, a hidden layer with three neurons, and an output layer with two neurons.

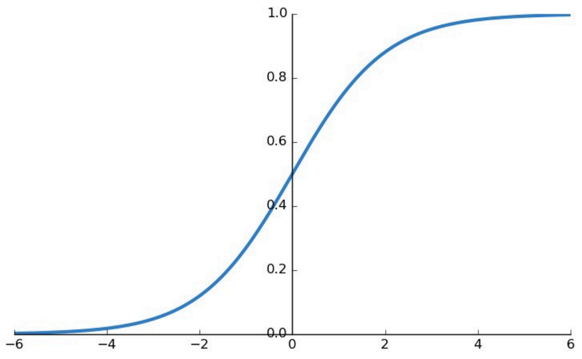
Let's have a closer look at the computation that happens at each hidden neuron. Mathematically, an artificial neuron is represented as follows, where f is an activation function, w_i are input weights, X_i are inputs, and $Y$is the output:

$$f(w_1 * X_1 + w_2 * X_2 \; + \; b) \; = \; Y.$$

The weights of the neurons in a layer are updated through a process called backpropagation. In a nutshell, backpropagation is a process that looks at the error of the network and works back through the network, looking at how the output of nodes needs to change to minimise the error. Calculus is involved in this process, so we won't go into much detail here but suffice to say that this process is used to adjust the weights and bias and that the activation function needs to be non-linear (i.e. not in the form of a straight line) for this to work.

The job of activation functions is to transform the sum of the weighted inputs so that it becomes non-linear and within a predictable range, and hence suitable for backpropagation. You may have noticed that without the activation function f, the computation at a neuron would look very similar to that of linear regression — hence this is a crucial element aiding the neural network's ability to solve more complex tasks.

Common activation functions include the **sigmoid function**, **hyperbolic tangent function (tanh)**, and **ReLu** function. Each of these functions transforms an input into a bounded range: the sigmoid function to the range [0:1], tanh to [-1:1], and ReLu to [0:1].



$$S(t) = \frac{1}{1 + e^{-t}}.$$

Sigmoid Function

If we look at the sigmoid function in more detail (see above image), we see that input values are adjusted to the range [0:1] along an S-shaped curve. This means that large positive input values tend to the value 1, while very large negative input values tend to 0.

WRITING YOUR FIRST FEEDFORWARD NEURAL NETWORK

We will now use Python to implement a simple feedforward neural network with two neurons in the input layer, two neurons in the first hidden layer and one neuron in the output layer.

```python
class OurNeuralNetwork:

  #initialise the parameters
  def __init__(self):
    self.w1 = np.random.randn()
    self.w2 = np.random.randn()
    self.w3 = np.random.randn()
    self.w4 = np.random.randn()
    self.w5 = np.random.randn()
    self.w6 = np.random.randn()
    self.b1 = 0
    self.b2 = 0
    self.b3 = 0

  def sigmoid(self, x):
    return 1.0/(1.0 + np.exp(-x))

  def feedforward(self, x):
    self.x1, self.x2 = x
    self.a1 = self.w1*self.x1 + self.w2*self.x2 + self.b1
    self.h1 = self.sigmoid(self.a1)
    self.a2 = self.w3*self.x1 + self.w4*self.x2 + self.b2
    self.h2 = self.sigmoid(self.a2)
    self.a3 = self.w5*self.h1 + self.w6*self.h2 + self.b3
    self.h3 = self.sigmoid(self.a3)
    return self.h3
```

In the code above, the class **`OurNeuralNetwork`** has three functions. Let's take a closer look at each of these functions.

The **`__init__`** function initialises all the parameters of the network including weights and biases. All 6 weights are initialised randomly and the 3 biases are set to zero.

The **`sigmoid`** function is defined next. This is used as the activation function for each of the neurons in the network. Remember that the sigmoid function is defined by:

$$S(t) = \frac{1}{1 + e^{-t}}.$$

Finally, we have the `feedforward` function, which takes an input **x** and computes the output.

Remember that to calculate the output of each neuron we use:

$$y = f(x_1 * w_1 + x_2 * w_2 + b)$$,

where:

- **x_1, x_2:** the output data from the previous layer or an external source;
- **w_1, w_2:** the weight of each input which is assigned on the basis of its relative importance to other inputs;
- **b:** the bias, which is a constant value;
- **f:** the activation function which is some form of computation that transforms the inputs; and,
- **y:** the output result of the transformed data from the neuron.

We first initialise two local variables and equate to input **x** which has two features. For each of the three neurons (two hidden and one output), we first calculate the weighted sum of its inputs plus the bias and store them in the variables **a_1**, **a_2**, and **a_3**. For example to calculate the weighted sum of its inputs plus the bias for the first neuron we use:

$$\mathbf{a_1 = w_1 * x_1 + w_2 * x_2 + b_1}$$

We then apply the activation function, sigmoid, to the output of the weighted sum of its inputs plus the bias and store them in the variables **h_1**, **h_2**, and **h_3**.

For example, to apply the activation function to the output **a_1** we use:

$$\mathbf{h_1 = sigmoid(a_1)}$$

This process is repeated for the second neuron to get **a_2** and **h_2**.

The outputs of the two neurons in the hidden layer then act as the input to the third output neuron. To calculate the weighted sum of the output neuron's inputs plus the bias we therefore use:

$$\mathbf{a_3 = w_5 * h_1 + w_6 * h_2 + b_3}$$

By applying the sigmoid function on **a_3** we get our final goal—the predicted output:

$$\mathbf{h_3 = sigmoid(a_3)}$$

And there you have it, the basics of a neural network.

.

Practical Task

- Create a copy of the **neural_network.ipynb** file and rename it **neural_network_task.ipynb**.
- Follow the instructions in the neural_network Jupyter notebook to complete the following tasks.
  - Work out what values would model an AND gate
  - Do the same for the NOR gate and the NAND gate
  - Combine the gates discussed
  - Finish a version of an XOR gate that more closely resembles a neural network by determining the shapes the weights and biases need to have.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCES

Medical Xpress. (2019). Synchronized or independent neurons: This is how the brain encodes information. Retrieved 28 August 2020, from **https://medicalxpress.com/news/2019-09-synchronized-independent-neurons-brain-encodes.html**

Ujjwalkarn. (2016). A Quick Introduction to Neural Networks. Retrieved 28 August 2020, from **https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/**