Hyperiondev

TASK

Neural Networks I

Visit our website

Copyright © 2021 HyperionDev. All rights reserved.

Introduction

WELCOME TO THE FIRST NEURAL NETWORKS TASK!

The machine learning algorithms we have worked with until now are considered more traditional or classic approaches. This is because currently many of the most impressive machine learning applications now use neural networks. The next few tasks will teach you what neural networks are and how to use them.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

Copyright © 2021 HyperionDev. All rights reserved.

WHAT ARE NEURAL NETWORKS?

The “Neural” of “Neural Network” comes from the tiny cells called neurons that are found within human brains. These neurons interact with each other to communicate and process information based on which we see, hear, move, think, make decisions, and generally function. A typical brain contains an estimated 100 billion neurons. Each neuron is made up of a single cell body with a number of connections (dendrites) coming off that carry information toward the cell body (inputs), and a single connection (axon) that carries information away from it (the output). Neurons are so tiny that about 100 of their cell bodies could fit into a single millimetre.

(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

Copyright © 2021 HyperionDev. All rights reserved.

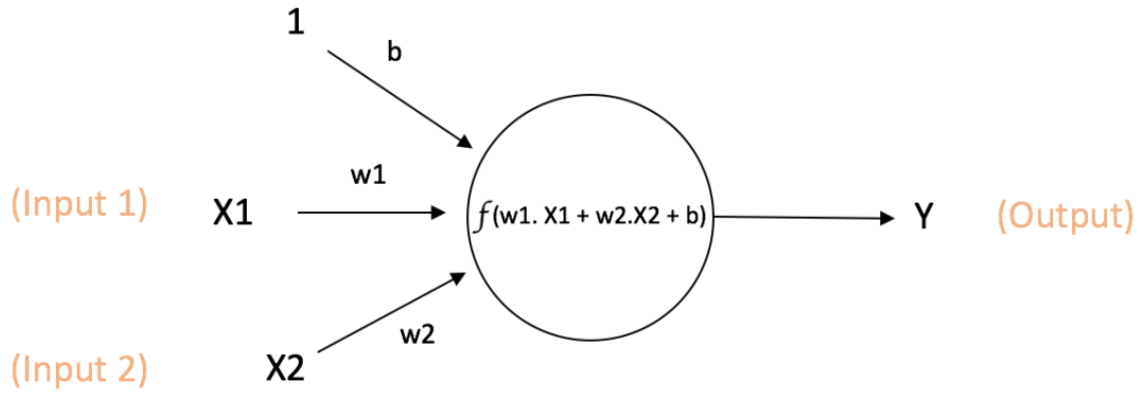
intricate sets of instructions we feed into them. Brains, on the other hand, learn slowly, but can spontaneously put information together in astounding new ways. They are able to recognise original patterns, forge new connections, and see the things they've learned in a completely different light.

For example, if you show a toddler pictures of cats and dogs and tell them which one is which, they'll very quickly be able to differentiate between them and identify cats and dogs. 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.



$$\text{Output of neuron} = Y = f(w1.\,X1 + w2.X2 + b)$$

Image source: (Ujjwalkarn, 2016)

Copyright © 2021 HyperionDev. All rights reserved.

The simplest type of neural network is called a feedforward neural network. Neurons are grouped together 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.

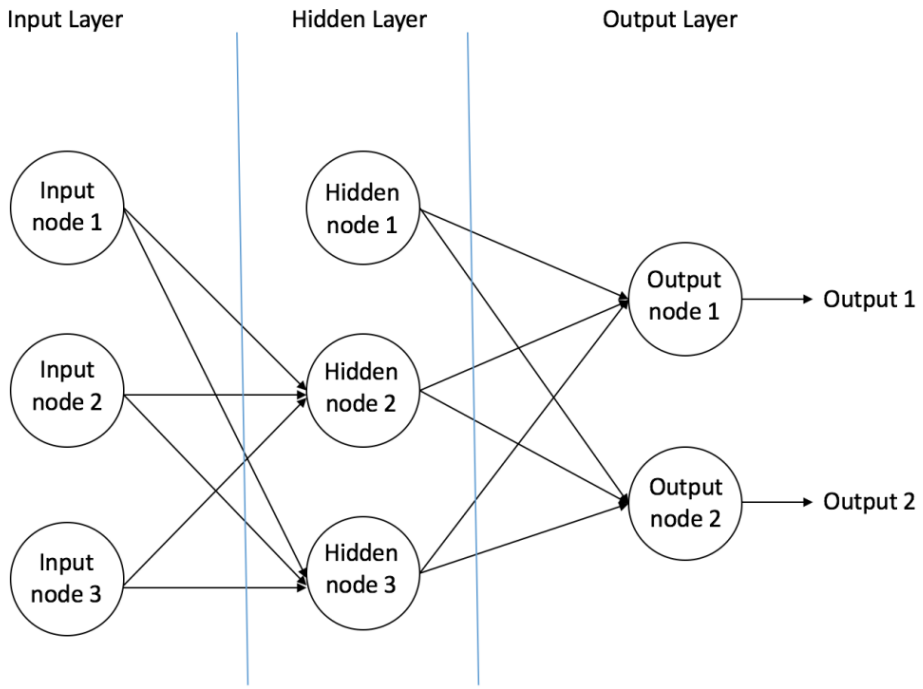


Image source: (Ujjwalkarn, 2016)

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 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:

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

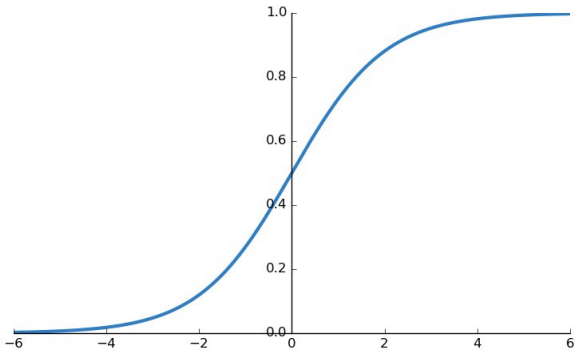
The weights of the neurons in a layer are updated through a process called backpropagation. In a nutshell, backpropagation is a process that looks at the error of the network and works back through the network, looking at how the output of nodes needs to change so as to minimise the error. Calculus is involved in this process, so we won't go into much detail here, but suffice to say that this process is

Copyright © 2021 HyperionDev. All rights reserved.

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) in order 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 neural networks able 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].



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

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.

```
class OurNeuralNetwork:

  #initialise the parameters
```


Copyright © 2021 HyperionDev. All rights reserved.

```
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:

Copyright © 2021 HyperionDev. All rights reserved.

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

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:

$$a_1 = w_1 * x_1 + w_2 * x_2 + b_1$$

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 a1 we use:

$$h_1 = sigmoid(a_1)$$

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:

$$a_3 = w_5 * h_1 + w_6 * h_2 + b_3$$

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

$$h_3 = sigmoid(a_3)$$

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

Copyright © 2021 HyperionDev. All rights reserved.

Instructions

The exercises in these tasks require users to run Jupyter Notebooks through the Anaconda environment. Please make sure that you download the Anaconda Distribution from: **https://www.anaconda.com/distribution/**.

Compulsory Task 1

Launch Jupyter Notebook via your Anaconda environment, upload the file named **L3T12_Intro_NN_CT1.ipynb** from your Task 12 folder, and follow the instructions. Submit your Task in .ipynb extension format.

If you are having any difficulties, please feel free to contact our specialist team **on Discord** for support.

Things to look out for:

1. Make sure that you have installed and set up all programs correctly. You have set up **Dropbox** correctly if you are reading this, but **Python or Notepad++** may not be installed correctly.
2. If you are not using Windows, please ask a reviewer for alternative instructions.

Completed the task(s)?

Ask an expert to review your work!

Review work

Copyright © 2021 HyperionDev. All rights reserved.

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/**

Copyright © 2021 HyperionDev. All rights reserved.