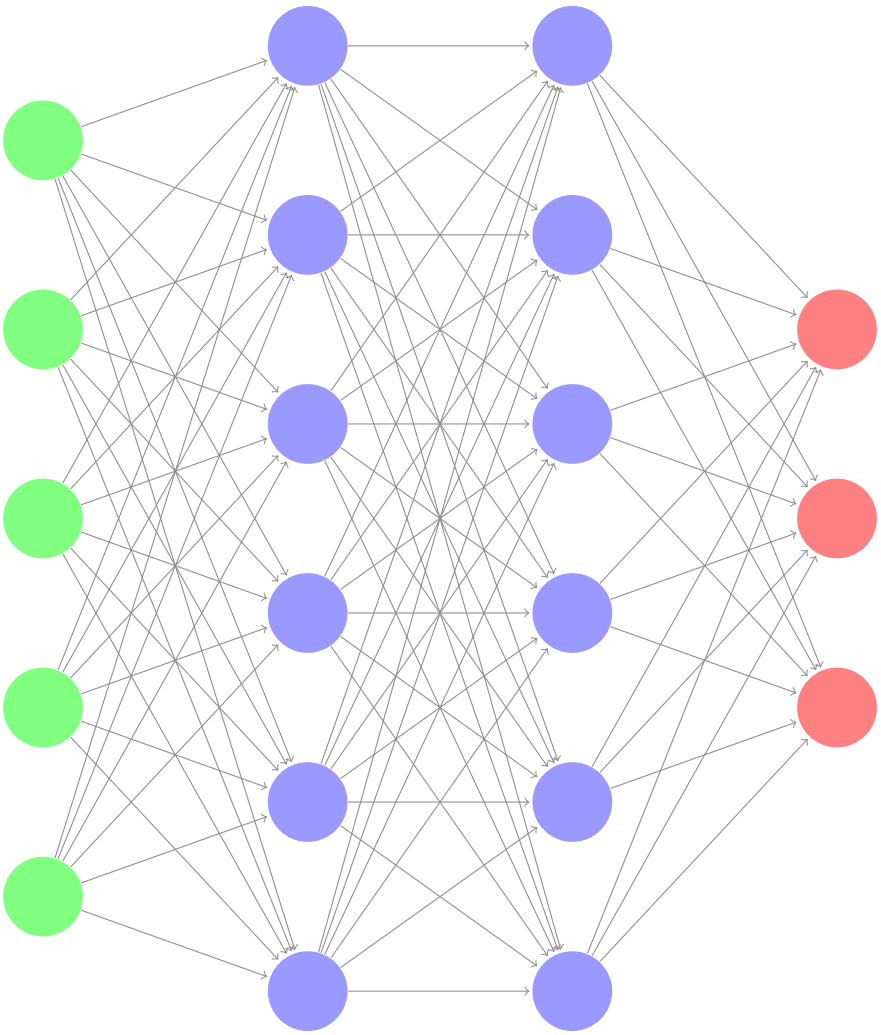
The math behind an artificial neural network

Hugo Lageneste

January 2020



Contents

The math behind an artificial neural network

Introduction

1 Application example

Let's take an example to see how an ANN[1] works.

Obesity	Exercise	Smoking	Diabetic
1	0	0	1
0	1	0	0
0	0	1	0
1	1	0	1

Figure 1: Example of a set of persons with Obesity, Exercise, Smoking and Diabetic properties. The 1 stands for true, the 0 for false.

We can observe that in this example, a person with diabetes is inevitably obese. What if we want a program which takes in parameter those 4 examples and can predict if a person is diabetic or not?

This problem can be modeled as an ANN with 3 input nodes which represent each property: the Obesity, Exercise and Smoking and a single output node which represents the Diabetic column.

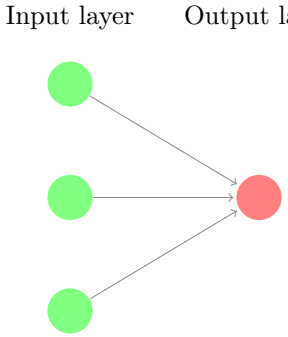


Figure 2: Diagram of the ANN to model the example

2 Neural network operation

Each property is represented as an input node so the network will operate as a function to say if the person is diabetic or not. The input nodes from a same layer are not link together but each node is linked to every node from the previous layer and from the next layer. Links between nodes are represented as weights and are unique to each liaison. A unique value is held in each node which is calculated with previous nodes values and weights. The whole goal of the neural network is to adjust the weights to make the final predictions right.

To adjust the weights, we will proceed in different steps. First of all, the neural network will compute the output values named the prediction. Then, the network will use a function to calculate the error of accuracy and adjust the weights. This process will be repeated until the success rate is convenient.

[1] Artificial Neural Network

A Forward propagation

As seen in the Introduction , each node holds a value contained between 0 and 1, and calculated with values of the previous neurons, weights and biases.

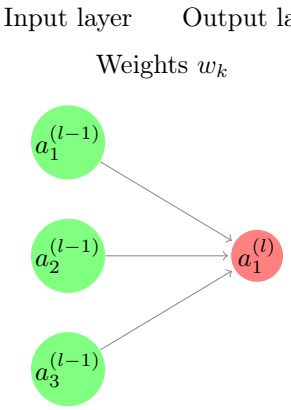


Figure 3: Diagram of a neural network, a is a node's value, its exponent represents the index of the layer and its subscript the index of the neuron then.

Let $z_1^{(l)}$ be the sum of the bias and the dot product of weights w_k and values of the previous neurons $a_k^{(l-1)}$.

$$z_1^{(l)} = b_1^{(l)} + \sum_{k=1}^{n_{(l-1)}} a_k^{(l-1)} w_k$$

In order to keep the values in the neurons between 0 and 1, we are going to use the sigmoid function which is defined on $]0, 1[$, noted :

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

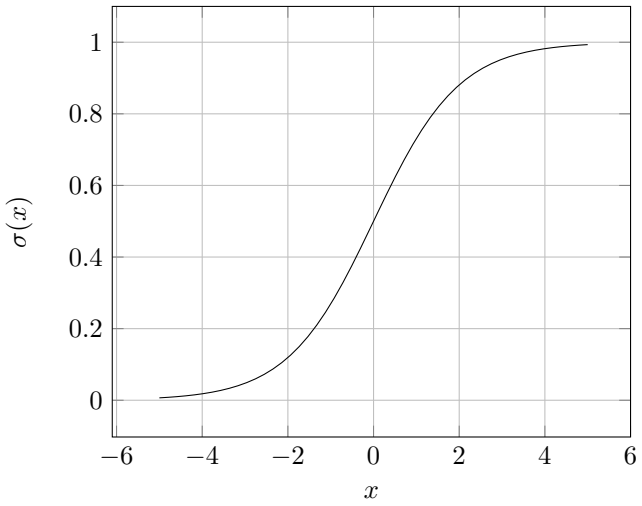


Figure 4: Graphical representation of the sigmoid function

Thus,

$$a_1^{(l)} = \sigma\left(z_1^{(l)}\right)$$

$$a_1^{(l)} \in]0, 1[$$

This calculus can be visualised with matrix too

$$a_1^{(l)} = \sigma\left(\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} a_1^{(l-1)} \\ a_2^{(l-1)} \\ a_3^{(l-1)} \end{bmatrix} + \begin{bmatrix} b_1^{(l)} \end{bmatrix}\right)$$

B Back propagation

1 Cost function

Let's take a simple example of a one-input-layer-one-output-layer-neural-network

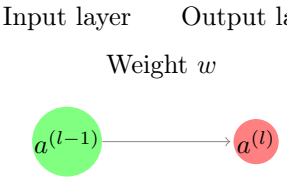


Figure 5: Diagram of the ANN to model the example of the cost function

Once again,

$$z^{(l)} = b + a^{(l-1)}w$$
$$a^{(l)} = \sigma\left(z^{(l)}\right)$$

The goal is now to adjust the weights to make the predictions more accurate. Let C_1 be the cost function/error function which is the square difference between the prediction and the actual output y.

$$C_1\left(a^{(l)}, y\right) = \left(a^{(l)} - y\right)^2$$

The smaller the cost function is, the more accurate the predictions are, mathematically the goal is to minimize the cost function.

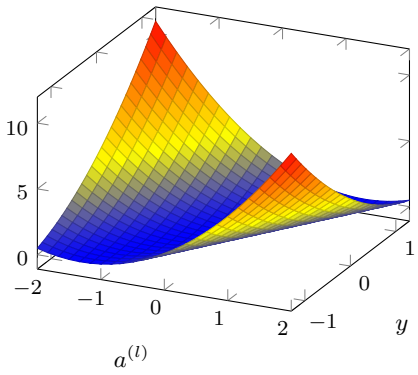


Figure 6: Graphical representation of cost function from the previous neural network, figure 5

2 Gradient descent

Now we will need to understand how sensitive the cost function is to small changes to w_k because remember from 2, the goal is to adjust weights. Thus, we will determine the partial derivative of C with respect to w using the chain rule.

$$\frac{\partial C_1}{\partial w} = \frac{\partial C_1}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} \frac{\partial z^{(l)}}{\partial w}$$

Indeed,

$$\frac{\partial C_1}{\partial a^{(l)}} = 2\left(a^{(l)} - y\right)$$

$$\frac{\partial a^{(l)}}{\partial z^{(l)}} = \frac{\partial \sigma\left(z^{(l)}\right)}{\partial z^{(l)}} = \sigma\prime\left(z^{(l)}\right)$$
$$\frac{\partial z^{(l)}}{\partial w} = \frac{\partial\left(b + a^{(l-1)}w\right)}{\partial w} = a^{(l-1)}$$

All together, it gives us

$$\frac{\partial C_1}{\partial w} = 2\left(a^{(l)} - y\right)\sigma\prime\left(z^{(l)}\right)a^{(l-1)}$$

We will use this formula to calculate the adjustments to make to the weights multiple times until the predictions are accurate.

C Complex example

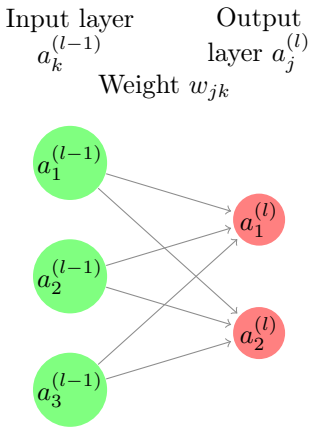


Figure 7: Diagram of a more complex ANN

We will model this problem with matrix because it is easier

1 Forward propagation

First of all we are going to create a matrix of weights and of bias

$$w = \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix}$$
$$b = \begin{bmatrix} b_1 & b_2 \\ b_1 & b_2 \\ b_1 & b_2 \end{bmatrix}$$

Then, let's calculate the predictions a. We consider that we have 3 set of input examples.

$$z = \begin{bmatrix} a_1^{(l-1)} & a_2^{(l-1)} & a_3^{(l-1)} \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix} \cdot w + b$$
$$a = \sigma(z)$$

2 Back propagation

We will use the cost function derivative to calculate the adjustments to make to the weights

$$e = a - \begin{bmatrix} a_1^{(l)} & a_2^{(l)} \\ \dots & \dots \\ \dots & \dots \end{bmatrix}$$

$$w_{adj} = 2e \cdot \sigma\prime(z) \cdot \begin{bmatrix} a_1^{(l-1)} & a_2^{(l-1)} & a_3^{(l-1)} \\ \dots & \dots & \dots \\ \dots & \dots & \dots \end{bmatrix}$$

We can now add up the adjustments to the weights

$$w = w + w_{adj}$$

And repeat the process until the success rate is >95%