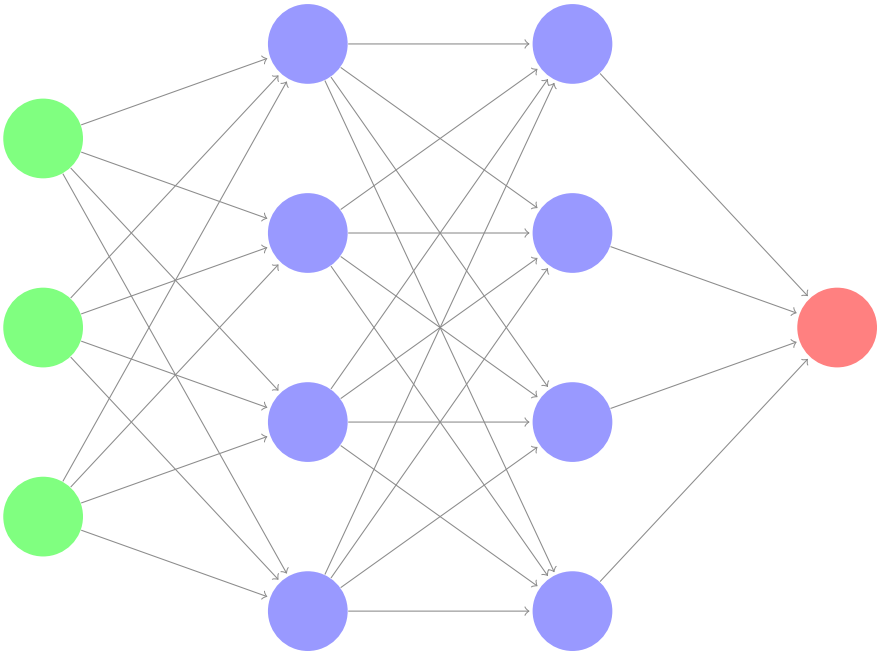
the math behind a neural network



olivia

hugo lageneste
hugo@olivia-ai.org

the math behind an artificial neural network, used for olivia-ai.org • 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 characteristics

In the precedent figure, the 1 stands for true and 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 only in parameter those 4 examples and can predict with different examples if a person is diabetic or not?

We can model this program as an ANN with 3 input neurons which represent the Obesity, Exercise and Smoking columns and a single output neuron which represents the Diabetic column.

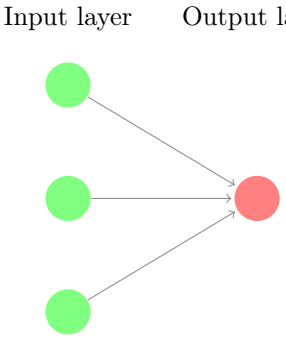


Figure 2: Diagram of the ANN to model the example

2 Neural network operation

The input values will be inserted in the input neurons and the network will operate as a "function" to say if the person is diabetic or not. Each liaison between each neuron is a value called weight and is unique. And each neuron holds a specific value, calculated with the previous neuron values and the weights. The whole goal of the neural network is to find the correct weights to make the final predictions right.

To make the weights right, we will proceed in different steps. First of all, the neural network will compute the value of the output layer's neurons named the prediction. Then, the network will make the difference between the prediction and the actual output and will adjust the weights to make the

[1] Artificial Neural Network

prediction more accurate. This process will be repeated until the success rate is convenient.

After the construction of the neural network, we will be able to send values like 1, 1 and 1 (so it means that the persons is obese, does exercise and smokes) and get an answer from the neural network which tells us that the person is diabetic.

A Forward propagation

As seen in the Introduction, each neuron holds a value contained between 0 and 1, calculated with the values of the previous neurons, the weights and a bias. We are going to see how these values are calculated.

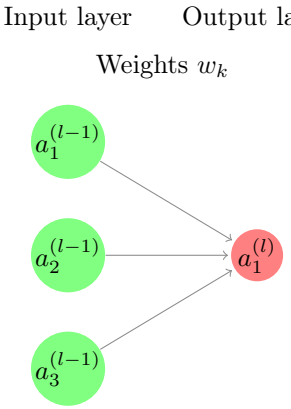


Figure 3: Diagram of a neural network, the exponent represents the index of the layer and the subscript the index of the neuron then a is the value held in the neuron.

Let's create a notation, $z_1^{(l)}$ which is the bias added to 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 :

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

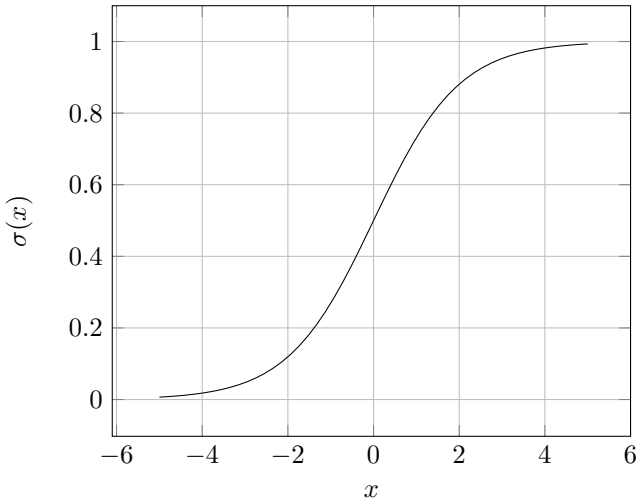


Figure 4: Graphical representation of the sigmoid function

Thus,

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

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

This calculus can be too visualised with matrix

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

B Back propagation

1 Cost function

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

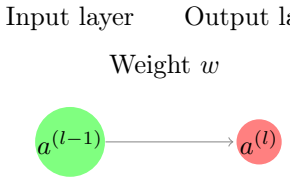


Figure 5: Diagram of the ANN to model the example of the cost function

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 prediction more accurate. Let's introduce the cost function which calculates 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.

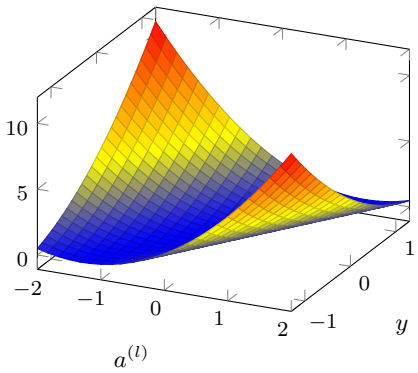


Figure 6: Cost function graphical representation of the neural network, figure 5

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.

$$w = w + \alpha\frac{\partial C_1}{\partial w}$$

C Complex example

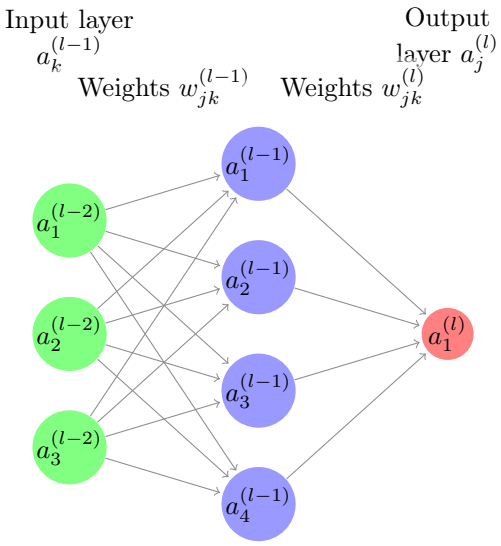


Figure 7: Diagram of a more complex ANN

We will model this problem with matrices.

1 Forward propagation

First of all we are going to create a matrix for each weights' layer

$$w^{(l-2)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix}$$

$$w^{(l-1)} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \\ w_{41} \end{bmatrix}$$

A layer is represented by a t by $k^{(l)}$ Matrix, where t is the number of training examples and k the number of nodes in a layer

$$a^{(l-2)} = \begin{bmatrix} a_1^{(l-2)} & a_2^{(l-2)} & a_3^{(l-2)} \end{bmatrix}$$

Then, to calculate the next layer's values we need to express z to compute a.

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

2 Back propagation

Because we have now a hidden layer, we will need to express the partial derivative of C with respect to $w^{(l)}$ couched in terms of l.
The adjustments will take the following form

$$w^{(l)} = w^{(l)} + \alpha \frac{\partial C}{\partial w^{(l)}}$$

Here, α represents the learning rate.
Thus, we need to compute this derivative $\frac{\partial C}{\partial w^{(l)}}$. Using the chain rule,

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

First of all let's compute $\frac{\partial z^{(l)}}{\partial w^{(l)}}$

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

Then, we'll compute $\frac{\partial C}{\partial z^{(l)}}$ couched in terms of $z^{(l+1)}$ in order to model backpropagation in programming.

$$\frac{\partial C}{\partial z^{(l)}} = \frac{\partial C}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}}$$

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

Thus, after adjusting the terms to make dot products working we got

$$\frac{\partial C}{\partial z^{(l)}} = \left(w^{(l+1)^T} \cdot \frac{\partial C}{\partial z^{(l+1)}} \right) \times \sigma\prime \left(z^{(l)} \right)$$

And,

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

As you can see here, to compute the adjustments for the weights you need the derivative of C with respect to the next z so we will need to calculate the derivative of C with respect to z for the last layer so we never run get out of the range. Here L is the last layer

$$\frac{\partial C}{\partial z^{(L)}} = 2(a^{(L)} - y) \sigma\prime \left(z^{(L)} \right)$$

$$\frac{\partial C}{\partial z^{(L)}} = 2(a^{(L)} - y) \sigma \left(z^{(L)} \right) \left(1 - \sigma \left(z^{(L)} \right) \right)$$
$$\frac{\partial C}{\partial z^{(L)}} = 2(a^{(L)} - y) a^{(L)} \left(1 - a^{(L)} \right)$$

Here we have all the ressources we need to build an artificial neural network.