Notes on Data Science and Physics

Dowling Wong

June 2023

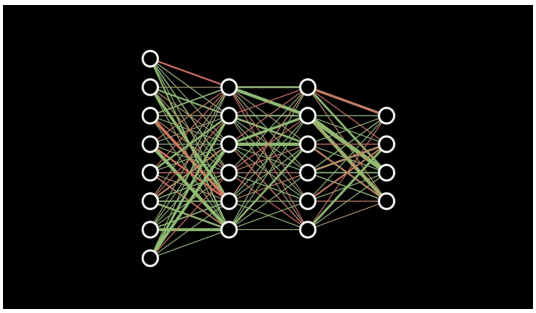
1 Prelude

This document has been written as Dowling's learning notes for 2023 summer research and **MIT Data Science and Physics**. The course **MIT Data Science and Physics** has been taught by Phil in 2023 spring.

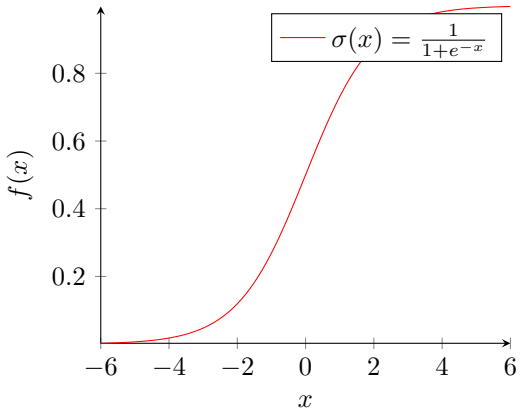
2 Basic concepts

2.1 Neural network

The neural network is used to recognize certain patterns. For example, consider a picture with a resolution 128x128=16384 pixels, after grey scale picture, each pixel has a grey scale value between 0 and 1(pure black=1, pure white=0), so the sample layer has a sample of 16384 floats, named neuron means a unit holds a value. Between result and sample layers are the hidden layers, stored some specific geometric patterns in the picture. To move from sample to hidden layers, we need a connection that gives weight neurons of the previous layer, and add them up to produce the value for new layer of neurons, the weighted sum could be at any place on the axis.



For each neuron in the new layer, we want them to be normalized between 0 and 1, to achieve this, we introduce the sigmoid function, a common function for activate with a value between 0 and 1 is usually the Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$



$$\omega_n = \Sigma\sigma(x)(a_i x_i - 10)$$

So the activation here is actually how positive the number in the new neuron cell is. To active meaningfully, we need to set a bias for inactivity, let's say the bias=10. The bias tells how high the neuron need to have, to be active. In the new layer, each neuron has a bias. To continue, assume we have 2 hidden layers of 16 neurons, and the final result has 10 neurons for 10 numbers from 0 to 9. By calculating, then we have total of 16384x16+16x16+16x10=262,560 weights and 16+16+10=36 bias. So the term learning simply just means finding the gigantic number of right weights and biases. A point need to mention here, all works and layers here is to get the neuron with largest number, the most most possible result.

Denote the numbers in neurons as $a_i^{(j)}$, in which (j) denotes layer number and i represents the number of that neuron in the (j) th layer.

$$a_0^{(1)} = \sigma(\omega_{0,0}a_0^{(0)} + \omega_{0,1}a_1^{(0)} + ... + \omega_{0,n}a_n^{(0)})$$

This is actually a vector equation:

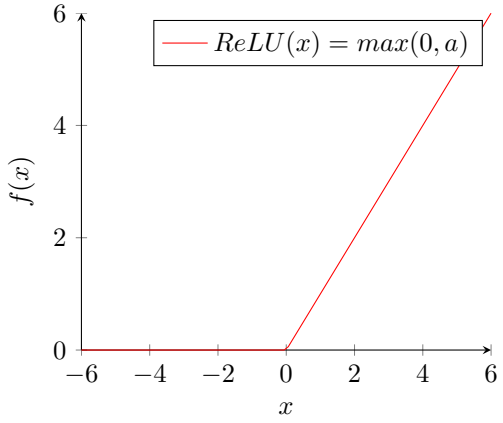
$$a^{(1)} = \sigma(\begin{bmatrix} \omega_{0,0} & \omega_{0,1} & ... & \omega_{0,n} \\ \omega_{1,0} & \omega_{1,1} & ... & ... \\ \vdots & \vdots & \vdots & \vdots \\ ... & ... & ... & \omega_{n,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}) = \sigma(Wa^{(0)} - b)$$

As corresponding code:

```
class Network(object):
    def __init__(self, *args, **kwargs):
        #this code is written in Python, here has initial weights and biases

    def feedforward(self, a):
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a) + b)
        return a
```

One last thing to mention for the neural network is the activation function of Sigmoid is considered old school nowadays, possibly another activation function might be more widely used: ReLU(x)=max(0, a). Easier for computation and rectified linear units. Split at threshold x=0, left is inactive, and right is active.



2.2 ROC curve