2IMM20 - Foundations of datamining

Assignment 3

Students:
Joris van der Heijden (0937329)
Bram van der Pol (0780042)

Email addresseses:
j.j.m.v.d.heijden@student.tue.nl
a.f.v.d.pol@student.tue.nl

Supervisors:
Dr.ir. Joaquin Vanschoren

Eindhoven, April 15, 2018

Contents

1 Backpropagation

1.1 Math

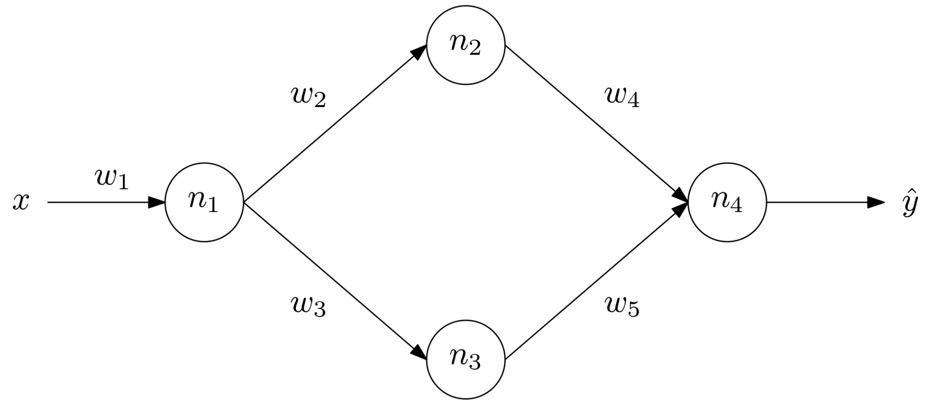

Figure 1: *Neural network*

The activation layers n are given by:

$$
\begin{aligned}
n_4 &= \sigma(z_4), \quad z_4 = n_2 w_4 + n_3 w_5 \\
n_3 &= \sigma(z_3), \quad z_3 = n_1 w_3 \\
n_2 &= \sigma(z_2), \quad z_2 = n_1 w_2 \\
n_1 &= \sigma(z_1), \quad z_1 = w_1 x
\end{aligned}
\tag{1}
$$

The loss function for 1 sample is given by:

$$L = \frac{1}{2}(n_4 - y)^2 \tag{2}$$

The derivatives w.r.t. the weights are given by:

$$\frac{\partial L}{\partial w_5} = \frac{\partial C}{\partial n_4}\frac{\partial n_4}{\partial z_4}\frac{\partial z_4}{\partial w_5} = \underbrace{(n_4 - y)\sigma'(z_4)}_{\alpha} n_3 \tag{3}$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial C}{\partial n_4}\frac{\partial n_4}{\partial z_4}\frac{\partial z_4}{\partial w_4} = \alpha n_2 \tag{4}$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial C}{\partial n_4}\frac{\partial n_4}{\partial z_4}\frac{\partial z_4}{\partial n_3}\frac{\partial n_3}{\partial z_3}\frac{\partial z_3}{\partial w_3} = \alpha w_5 \sigma'(z_3) n_1 \tag{5}$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial C}{\partial n_4}\frac{\partial n_4}{\partial z_4}\frac{\partial z_4}{\partial n_2}\frac{\partial n_2}{\partial z_2}\frac{\partial z_2}{\partial w_2} = \alpha w_4 \sigma'(z_2) n_1 \tag{6}$$

$$\frac{\partial L_{n_2}}{\partial w_1} = \frac{\partial C}{\partial n_4}\frac{\partial n_4}{\partial z_4}\frac{\partial z_4}{\partial n_2}\frac{\partial n_2}{\partial z_2}\frac{\partial z_2}{\partial n_1}\frac{\partial n_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = \alpha w_4 \sigma'(z_2) w_2 \sigma'(z_1) x \tag{7}$$

$$\frac{\partial L_{n_3}}{\partial w_1} = \frac{\partial C}{\partial n_4}\frac{\partial n_4}{\partial z_4}\frac{\partial z_4}{\partial n_2}\frac{\partial n_2}{\partial z_3}\frac{\partial z_3}{\partial n_1}\frac{\partial n_1}{\partial z_1}\frac{\partial z_1}{\partial w_1} = \alpha w_4 \sigma'(z_3) w_3 \sigma'(z_1) x \tag{8}$$

$$\frac{\partial L}{\partial w_4} = \frac{\partial L_{n_2}}{\partial w_1} + \frac{\partial L_{n_3}}{\partial w_1} \tag{9}$$

The compute graph of the neural network is chosen as a sigmoid:

$$\begin{aligned} \sigma(z) &= \frac{1}{1+e^{-z}} \\ \sigma'(z) &= \frac{e^{-z}}{(exp^{-z}+1)^2} \end{aligned} \tag{10}$$

The update expressions are given by:

$$w_i^{new} = w_i - \alpha \frac{\partial L}{\partial w_i}, \ i = 1, \ldots, 5 \tag{11}$$

First the activation layers are calculated using: $\mathbf{w} = [2\ 1\ 2\ 4\ 1]$ and $x = 2,\ y = 3$.

$$\begin{aligned} z_1 &= w_1 x = 4 & n_1 &= \sigma(z_1) = 0.982 \\ z_2 &= n_1 w_2 = 0.982 & n_2 &= \sigma(z_2) = 0.7275 \\ z_3 &= n_1 w_3 = 1.964 & n_3 &= \sigma(z_3) = 0.877 \\ z_4 &= n_2 w_4 + n_3 w_5 = 3.787 & n_4 &= \sigma(z_4) = 0.977 \end{aligned} \tag{12}$$

The initial error is: 2.0446. Now by back propagation the new weights are given by:

$$w_{new}^{hand} = \begin{bmatrix} 2.0003 & 1.0034 & 2.0005 & 4.0032 & 1.0038 \end{bmatrix} \tag{13}$$

The python code:

```
class MyGraph(object):

    def __init__(self, x, y, weigths):
        ''' x: input
            y: expected output
            w: initial weight
            b: initial bias '''

        self.weights = [VariableNode(weight) for weight in weigths]

        self.z1 = MultiplicationNode([ConstantNode(x),self.weights[0]]
        self.n1 = SigmoidNode([self.z1])
        self.z2 = MultiplicationNode([self.n1, self.weights[1]])
        self.n2 = SigmoidNode([self.z2])
        self.z3 = MultiplicationNode([self.n1, self.weights[2]])
        self.n3 = SigmoidNode([self.z3])
        self.z4 = AdditionNode([MultiplicationNode([
                                    self.n2,
                                    self.weights[3]]),
                                    MultiplicationNode([
                                    self.n3,
                                    self.weights[4]])])
        self.n4 = SigmoidNode([self.z4])
        self.graph = MSENode([self.n4,ConstantNode(y)])

    def forward(self):
        return self.graph.forward()

    def backward(self, d):
        self.graph.backward(d)
    def set_weights(self, new_weights):
        for i in len(new_weights):
            self.weights[i].output = new_weights[i]

    def get_weights(self):
        return [weight.output for weight in self.weights]
```

The structure is created using the multiplication and addition nodes. First is started with the n_1 node, thereafter the n_2 and n_3 nodes and finally n_4 is constructed using the addition node. This gives the desired shape.

$$w_{new}^{python} = \begin{bmatrix} 2.000 & 1.003 & 2.0004.003 & 1.004 \end{bmatrix} \quad (14)$$

As can been seen in 13 and 14 is that the handmade calculations are more accurate but the numbers calculated by the python code are equal to the handmade calculations.

2 Assignment 2

3 Assignment 3