# Deep learning - Chapter 4 + 5

Onno Huygen

October 30, 2016

# Overview

- Sigmoid acitivation function:

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

where:

$$z = \sum_i w_i a_i + b = \vec{w} \cdot \vec{a} + b$$

And for the derivative:

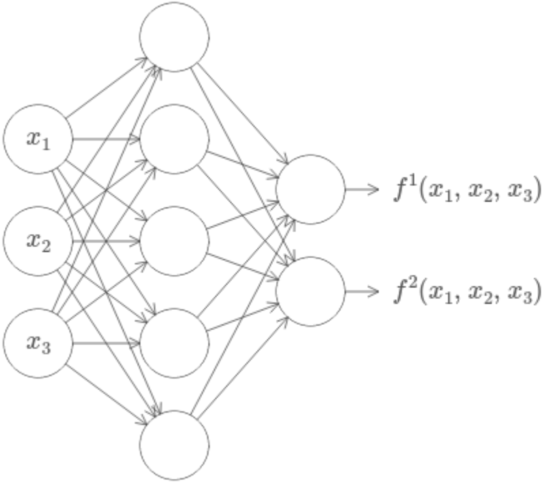
$$\frac{d\sigma(z)}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

Introduction

4. Universality theorem

5. Learning rate instability

- Universality: A network with just one hidden layer can simulate any smooth function.



# Caveats

- In practice, networks do not give exact functions
- Rather, we can always design and train a network such that the error is smaller than an arbitrary $\epsilon$

$$|g(x_i) - f(x_i)| \leq \epsilon$$

- Visual 'proof': build up approximation of function from blocks

Consider one neuron with one input. For a small change in the bias:

$$b \mapsto b' = b + \Delta b$$

And thus:

$$e^{-(w \cdot x + b + \Delta b)} = e^{-(w \cdot (x + \frac{\Delta b}{w}) + b)}$$

$$\sigma(x) \mapsto \sigma'(x) = \sigma(x + \frac{\Delta b}{w})$$

$\Rightarrow$ A change in a bias shifts the function by $\frac{\Delta b}{w}$

Define width of sigmoid to be $\omega := (x_1 - x_2)$ given by $\sigma(x_1) = 0,9$ and $\sigma(x_2) = 0.1$. Solve for $x_{1,2}$:

$$\begin{aligned}
(1 + e^{-wx_1 - b})^{-1} &= 0,9 \\
e^{-wx_1 - b} &= \frac{0,1}{0,9} \\
-wx_1 - b &= \ln\left(\frac{0,1}{0,9}\right) = \ln(0,1) - \ln(0,9) \\
x_1 &= \frac{\ln(0,9) - \ln(0,1) - b}{w}
\end{aligned}$$

Similarly:

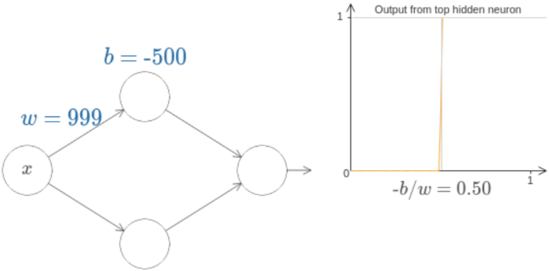
$$x_2 = \frac{\ln(0,1) - \ln(0,9) - b}{w}$$

And thus:

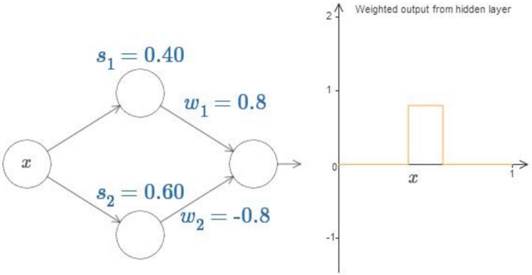
$$\omega = \frac{2\ln(0,9) - 2\ln(0,1) - \cancel{b} + \cancel{b}}{w}$$

Approximating step function:

- Width only depends on $w$. Choose $w$ high, say $w = 1000$. Width $\omega$ is inversely proportional.
- To shift step function to point $a$, choose bias to be $-w \times a = -1000 \times a$
- Call these functions $s_a(x)$ as a step function centered at point a.

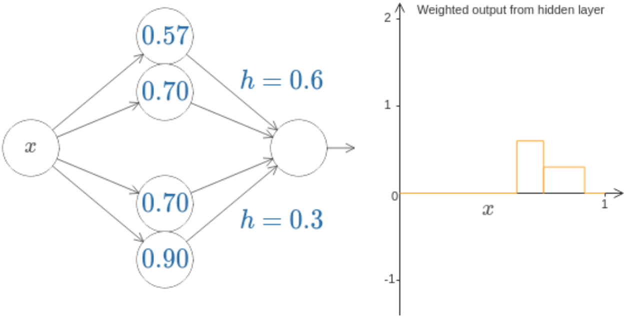


Now combine two of these functions with weights $w_1 s_a(x) + w_2 s_{a+\Delta a}(x)$. If we choose $w_2 = -w_1$, the step functions are of equal height and opposite sign. We get a block:

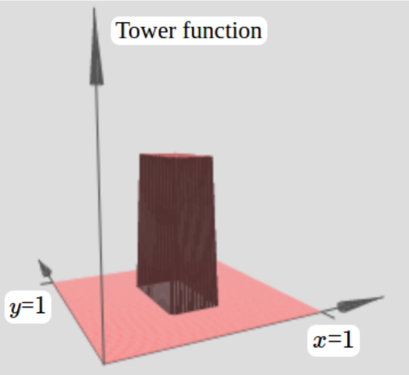



(We can also make a 'hole' by exchanging the weights)

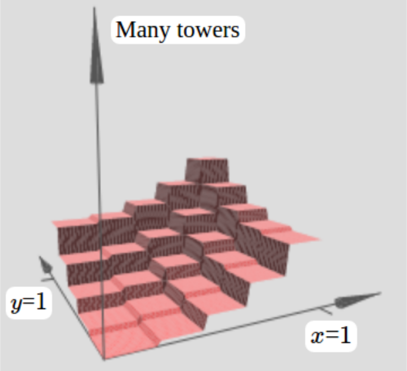
Adding pairs of neurons with weights of equal magnitude and opposite sign adds more blocks



Similarly, for multiple variables, it is possible to make a tower function by using four neurons:



Multiple towers can be used to form any function

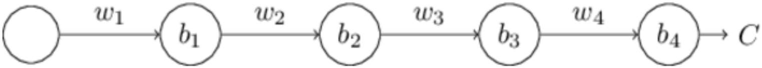


It is not necessary to choose sigmoid function. Restrictions:

- Function is smooth from $z = -\infty$ to $z = \infty$
- $\lim_{z \to -\infty} \sigma(z) = 0$ and $\lim_{z \to \infty} \sigma(z) = \alpha$ where $\alpha$ is any positive number

- While a network with one layer can simulate any function, demanding it is one layer deep could produce extremely complicated networks.
- Ideally one learning rate throughout whole network. Then measure of training (number of epochs) is meaningful for every layer.
- Remember learning rate for a neuron $\propto \frac{\partial C}{\partial w_i}$

Consider a neuron chain consisting of one neuron per layer with $N$ layers.



Let $a_i$ be the output of layer $i$ and $z_i = a_{i-1} \cdot w_i + b_i$ (thus $a_i = \sigma(z_i)$). Compute using the chain rule:

- Last layer:

$$\frac{\partial C}{\partial w_N} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial z_N} \frac{\partial z_N}{\partial w_N}$$

Let $a_i$ be the output of layer $i$ and $z_i = a_{i-1} \cdot w_i + b_i$ (thus $a_i = \sigma(z_i)$). Compute using the chain rule:

- Last layer:

$$\frac{\partial C}{\partial w_N} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial z_N} \frac{\partial z_N}{\partial w_N}$$

- One deeper:

$$\frac{\partial C}{\partial w_{N-1}} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial z_N} \frac{\partial z_N}{\partial a_{N-1}} \frac{\partial a_{N-1}}{\partial z_{N-1}} \frac{\partial z_{N-1}}{\partial w_{N-1}}$$

Let $a_i$ be the output of layer $i$ and $z_i = a_{i-1} \cdot w_i + b_i$ (thus $a_i = \sigma(z_i)$). Compute using the chain rule:

- Last layer:

$$\frac{\partial C}{\partial w_N} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial z_N} \frac{\partial z_N}{\partial w_N}$$

- One deeper:

$$\frac{\partial C}{\partial w_{N-1}} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial z_N} \frac{\partial z_N}{\partial a_{N-1}} \frac{\partial a_{N-1}}{\partial z_{N-1}} \frac{\partial z_{N-1}}{\partial w_{N-1}}$$

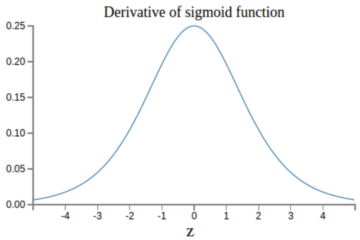
- And one deeper:

$$\frac{\partial C}{\partial w_{N-1}} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial z_N} \frac{\partial z_N}{\partial a_{N-1}} \frac{\partial a_{N-1}}{\partial z_{N-1}} \frac{\partial z_{N-1}}{\partial a_{N-2}} \frac{\partial a_{N-2}}{\partial z_{N-2}} \frac{\partial z_{N-2}}{\partial w_{N-2}}$$

General equation (product index decreases):

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial a_N} \left( \prod_{j=N}^{(i+1)} \frac{\partial a_j}{\partial z_j} \frac{\partial z_j}{\partial a_{j-1}} \right) \times \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial w_i}$$

The $\frac{\partial z_i}{\partial a_{i-1}}$ are just $w_i$. The $\frac{\partial a}{\partial z}$ are given by the derivative of the sigma function (introduction 2). It looks like this:



Thus, for each layer deeper, the product gets an additional term $\frac{\partial\sigma(z_i)}{z_i} \times w_{i-1}$. From previous slide, $|\frac{\partial\sigma(z_i)}{\partial z_i}| \leq \frac{1}{4}$. Typically, $w_i$ are of order 1. Thus, the learning rate decreases for deeper layers.

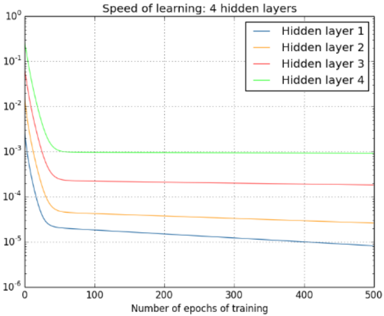



Figure : y-axis: learning rate.

Opposite can happen as well: for very high weights, the learning rate explodes.

Low learning rate associated with stable point for which $\frac{\partial C}{\partial w_i}$ is small. But this is certainly not the case! Weights and biases initiated randomly, but still low learning rate.

Solution? Different activation function with larger $|\frac{\partial f(z_i)}{\partial z_i}|$?

Low learning rate associated with stable point for which $\frac{\partial C}{\partial w_i}$ is small. But this is certainly not the case! Weights and biases initiated randomly, but still low learning rate.

Solution? Different activation function with larger $|\frac{\partial f(z_i)}{\partial z_i}|$?

No this decreases the vanishing learning rate problem but increases the exploding rate problem.

For multiple neurons per layer, we just change to a matrix/vector equation, but we have the same problem. Other problems (requires further reading):

- The sigmoid function can cause saturation of activation functions at 0. Learning rate is extremely low for activation functions that are saturated.
- Choosing non-random initial conditions still impacts the ability to train a network. Thus, if choosing smart weights is not possible, the learning ability depends on random weights.