Mathematical Modelling and Simulation

Artificial Neural Networks

Motivation

Mathematical Modelling and Simulation

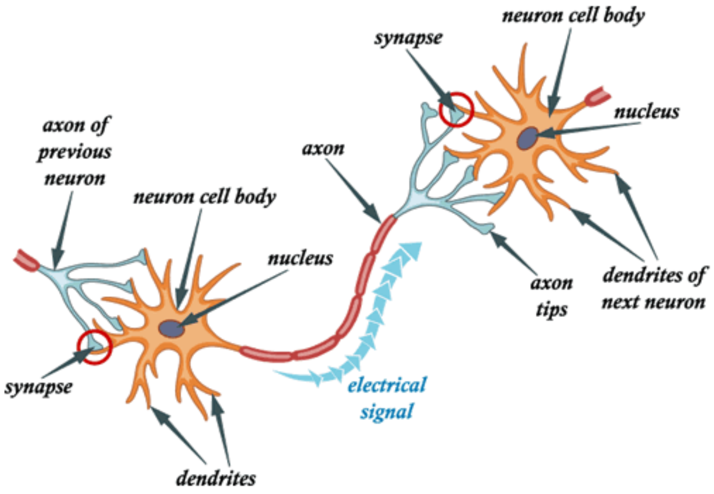
Source: https://www.analyticsindiamag.com/artificial-neural-networks-101/

Motivation

Trying to imitate the human brain

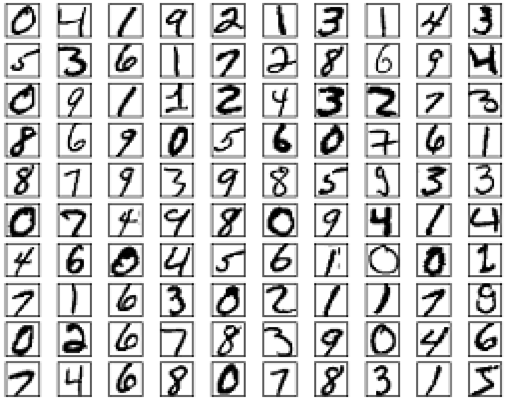
Same structure to process information

activating neurons



Simplest and most common application:

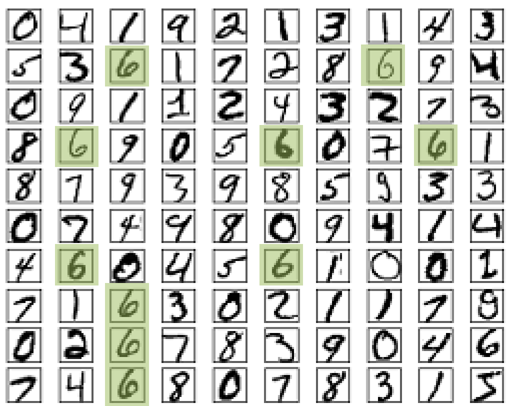
image recognition



Motivation

Simplest and most common application:

image recognition



6

Motivation

Simplest and most common application:

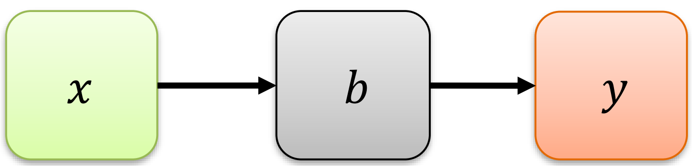
image recognition

Source: http://neuralnetworksanddeeplearning.com/chap1.html

Neural Networks - Structure

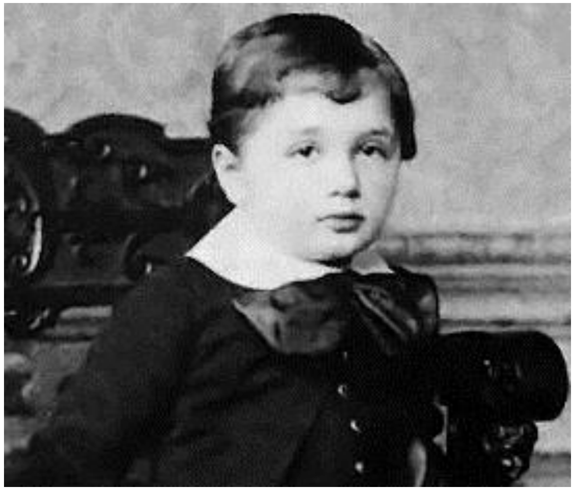
Different layers in a neural network:

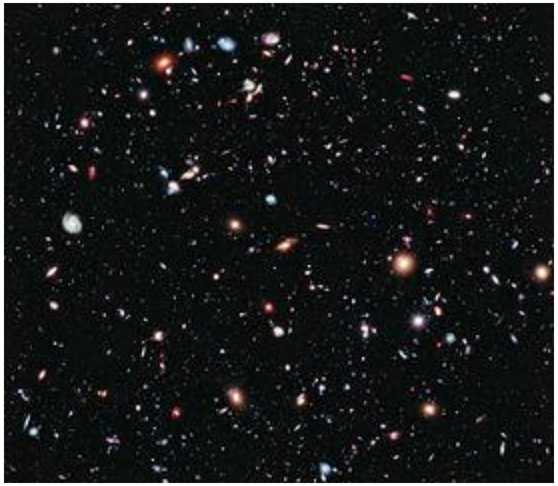
Input layer	defined by the chosen input data structure
Hidden layers	one or more layers with hidden neurons
Output layer	defined by the desired output



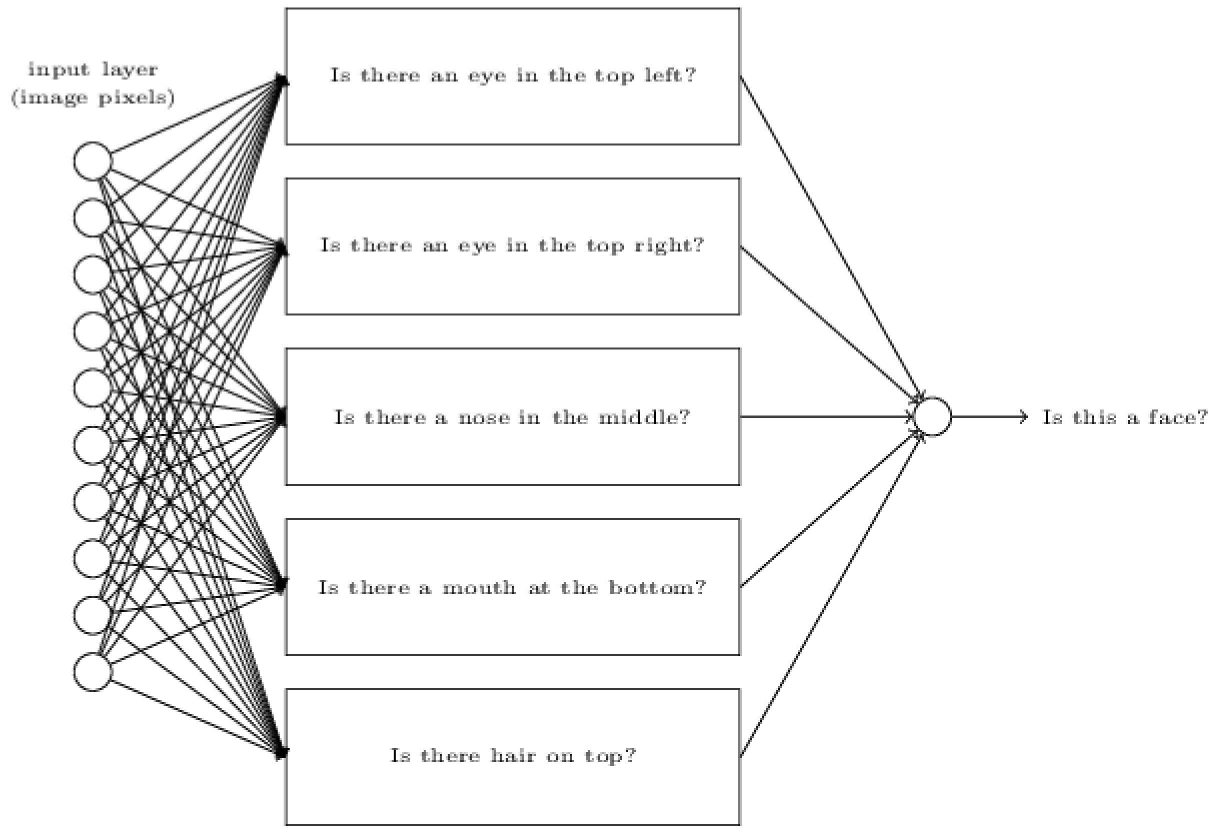
Example – Human Face

Can we understand how such "intelligent" networks work?



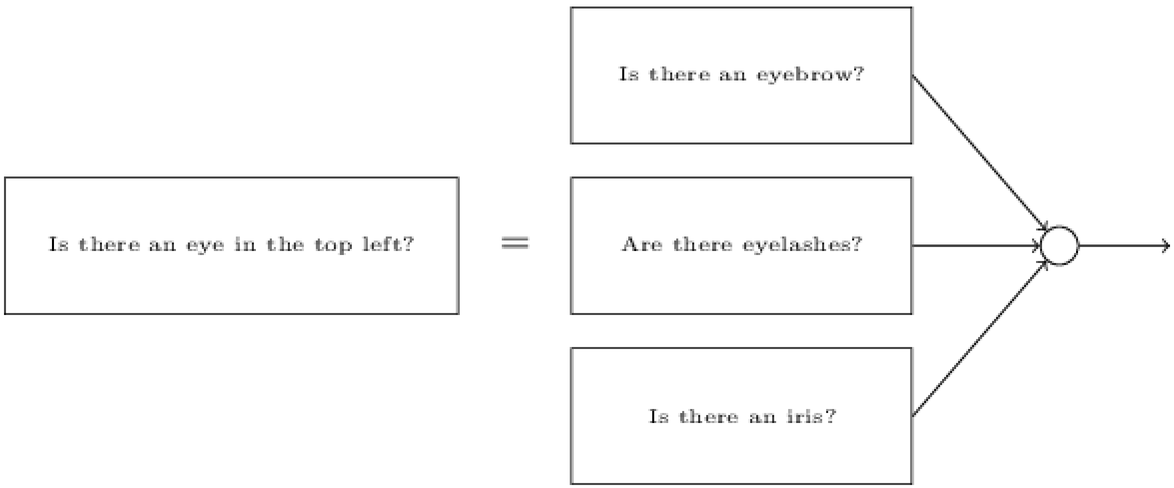


Example – Human Face

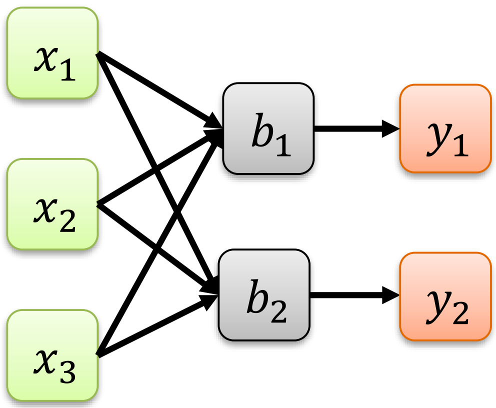


Example – Human Face

Mathematical Modelling and Simulation

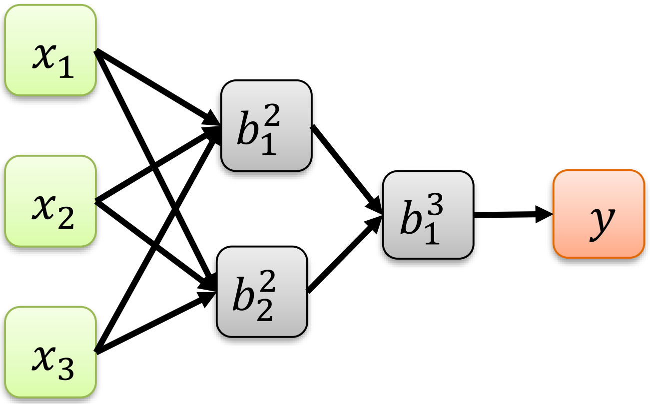


Neural Networks - Structure



Different structures depending on the desired input and output are possible

One could chose certain connections or connect all the elements with each other

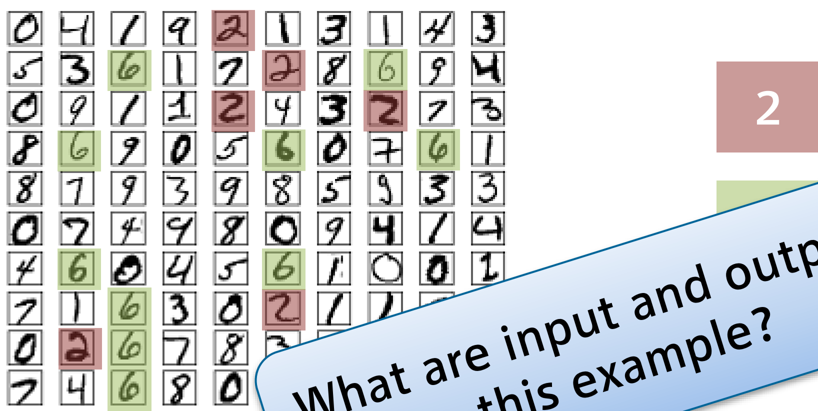


Structure of the hidden layers is "flexible"

Motivation

Simplest and most common application:

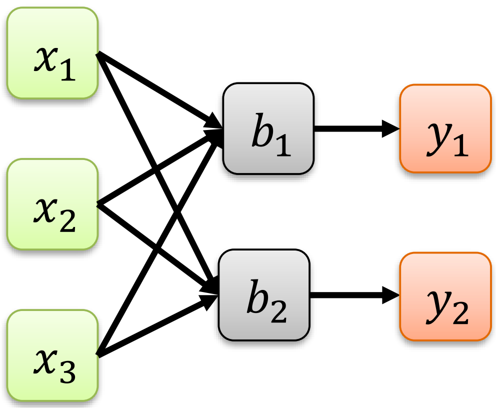
image recognition

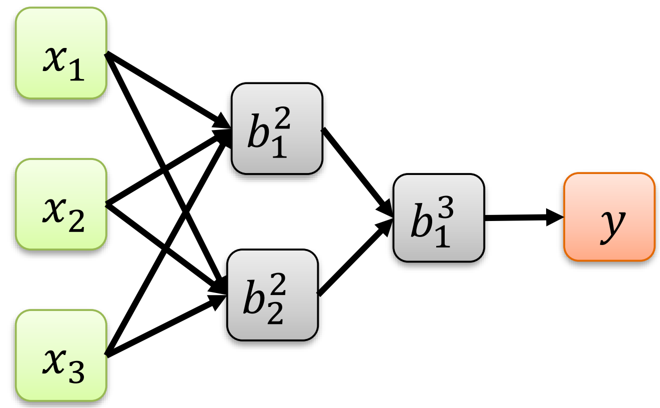


What are input and output in this example?

Source: http://neuralnetworksanddeeplearning.com/chap1.html

Neural Networks – Input & Output



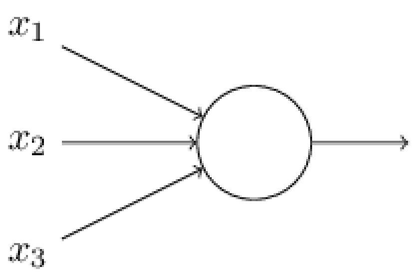


Input: $x \in \mathbb{R}^n, n \in \mathbb{N}$
Output: $y \in \mathbb{R}^m, m \in \mathbb{N}$

History - Perceptron Network I

Mathematical Modelling and Simulation

- „perceptron“ in 1950s by Frank Rosenblatt and inspired by Warren McCulloch and Walter Pitts



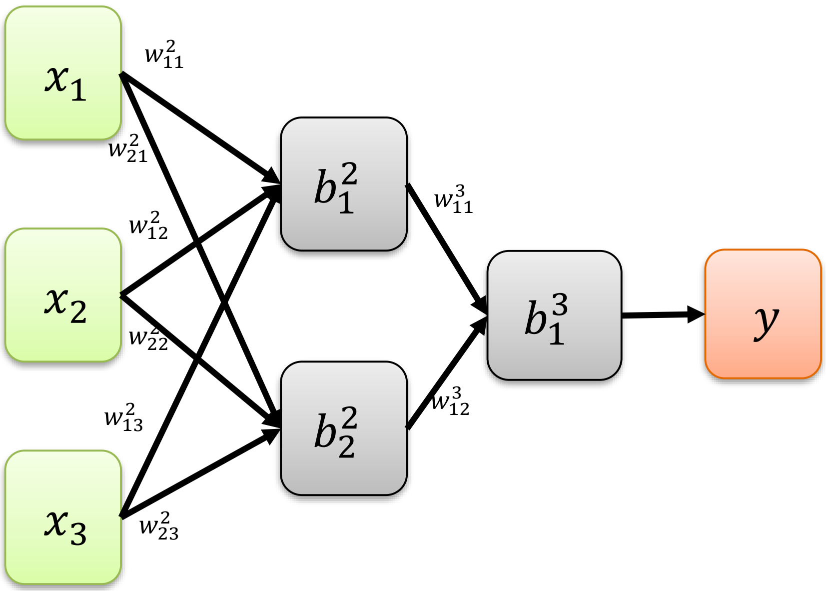
$$\text{output} = \begin{cases} 0 & , \sum w_i x_i \leq tr \\ 1 & , \sum w_i x_i > tr \end{cases}$$

several binary input creating
single binary output

- perceptron useable for logical functions
- Very sensitive regarding weights

Neural Networks - Weights

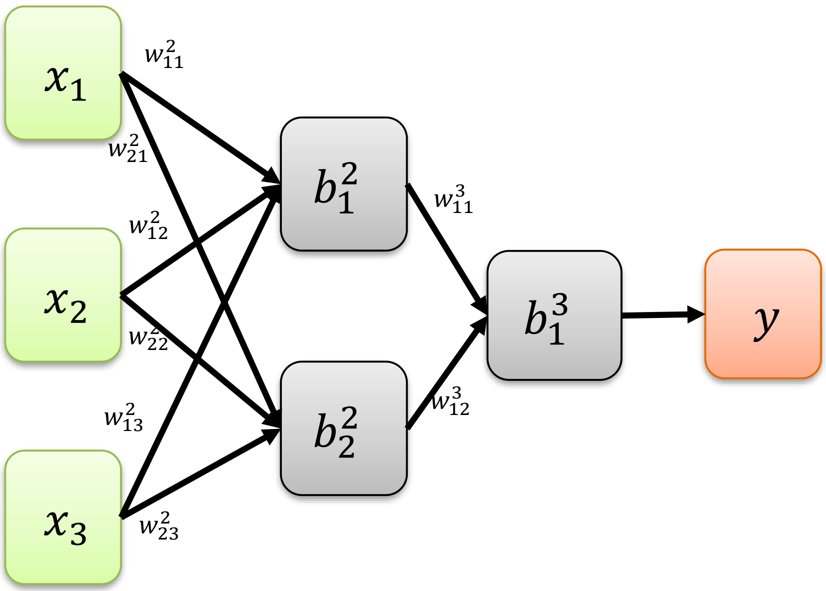
Mathematical Modelling and Simulation



weights : $w_{ij} \in \mathbb{R}$

Neural Networks - Weights

Mathematical Modelling and Simulation



Every edge gets a certain weight:

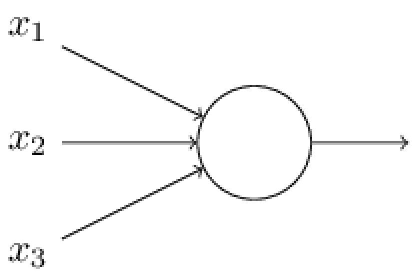
$w_{jk}^l < 0$ damping
$w_{jk}^l > 0$ amplifying

weights :$w_{jk}^l \in \mathbb{R}$

History - Perceptron Network II

Mathematical Modelling and Simulation

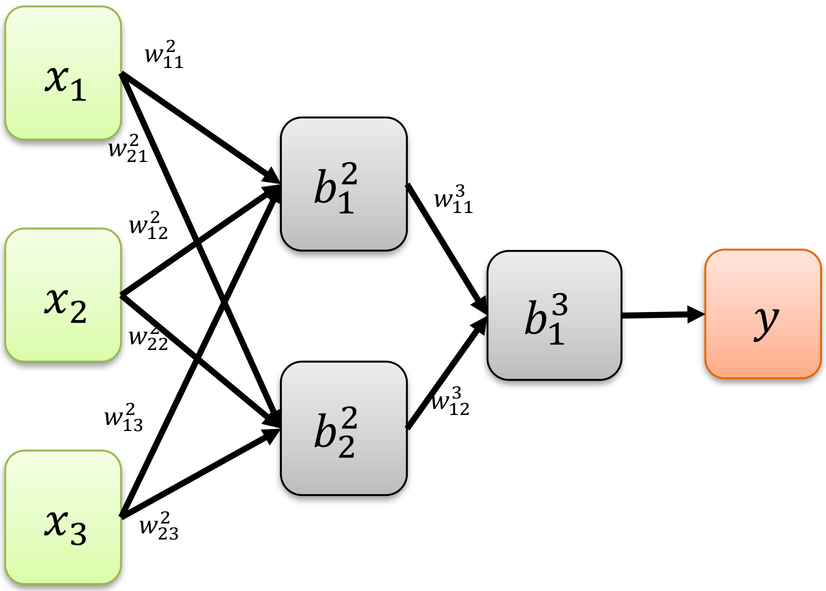
- „perceptron“ in 1950s by Frank Rosenblatt and inspired by Warren McCulloch and Walter Pitts



$$\text{output} = \begin{cases} 0 & , \sum w_i x_i + b \leq 0 \\ 1 & , \sum w_i x_i + b > 0 \end{cases}$$

Different formulation with $b = -tr$

Neural Networks - Bais



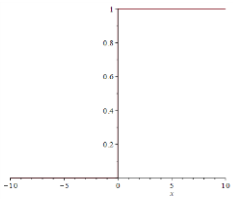
Bias: $b_i^j \in \mathbb{R}$
measure possibility of firing

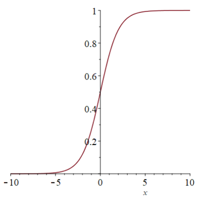
(other way of threshold)

Perceptron to Neural Network III

$$\text{output} = \begin{cases} 0 & , \sum w_i x_i + b \leq 0 \\ 1 & , \sum w_i x_i + b > 0 \end{cases}, \quad b = -tr$$

“Smooth improvement”:
introducing „sigmoid neurons“





Low sensitive regarding weights and bias

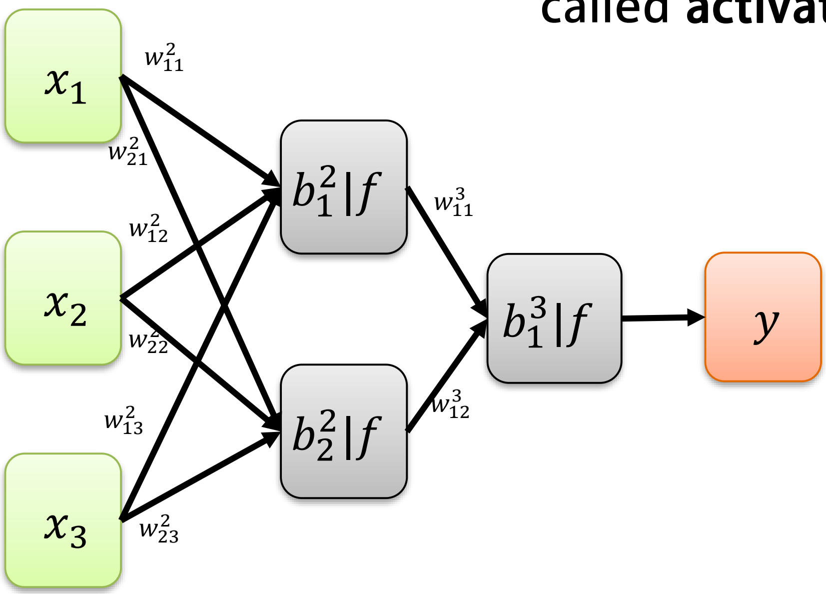
$$\Delta output \approx \sum_j \frac{\partial output}{\partial w_j} \Delta w_j + \frac{\partial output}{\partial b} \Delta b$$

arbitrary input creating
Output $\in (0,1)$

Neural Networks – Activation Function

For every incoming signal a function f is applied at the neuron

called **activation function**



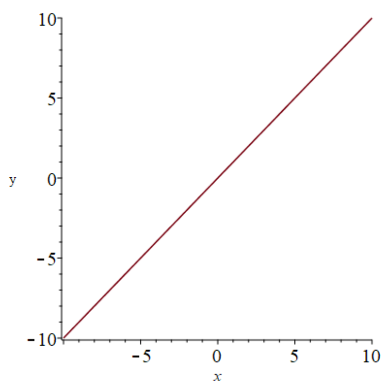
Activation function :
$f \in C(\mathbb{R})$

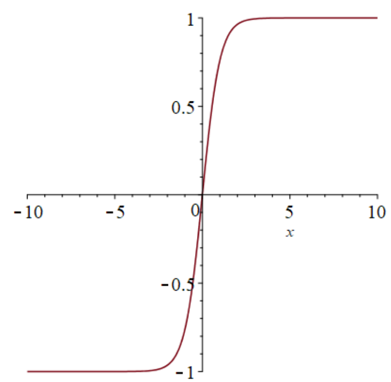
Neural Networks – Activation Function

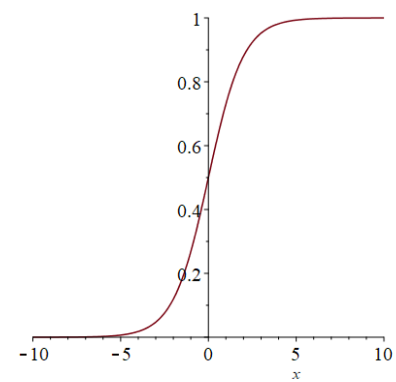
Activation functions influence complexity of the neural network:

- Linear function
- Tangens hyperbolic
- Sigmoid function

Activation function :
$f \in C(\mathbb{R})$





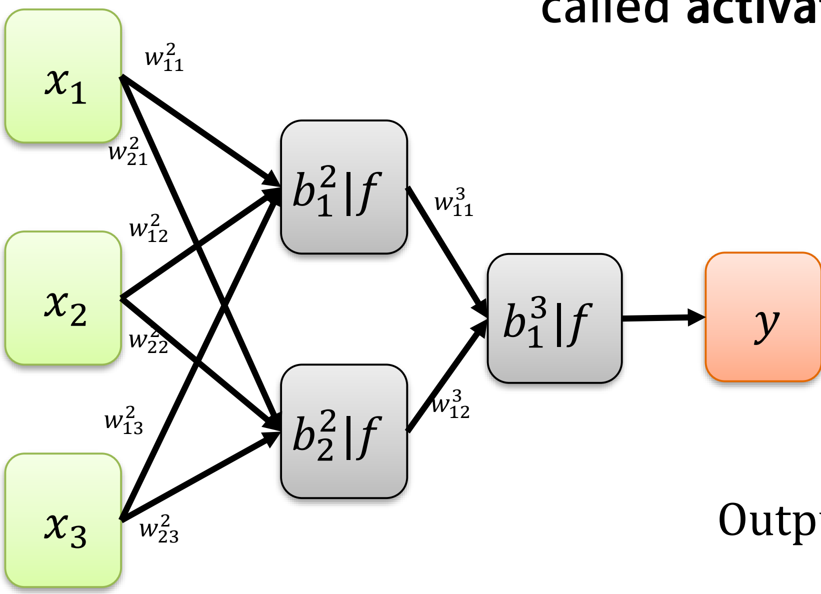


Neural Networks – Activation Function

Mathematical Modelling and Simulation

For every incoming signal a function f is applied at the neuron

called **activation function**

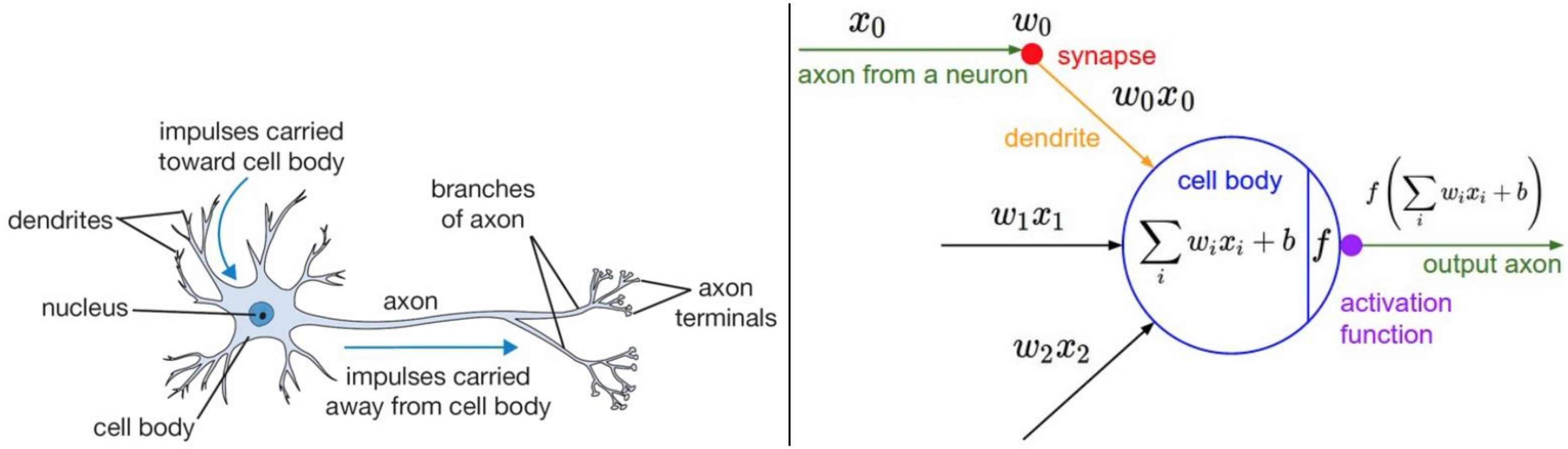


$$\text{Output}_{b_k^2} = f\left(\sum_{i=1}^{3} w_{ki}^2 x_i + b_k^2\right), k \in \{1,2\}$$

$$y = f\left(\sum_{i=1}^{2} w_{1i}^3 Output_{b_i^2} + b_1^3\right)$$

Mathematical Modelling and Simulation

Imitation of the human brain



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Example – Classify Handwritten Digits

Mathematical Modelling and Simulation

Simplest and most common application:

image recognition

Source: http://neuralnetworksanddeeplearning.com/chap1.html

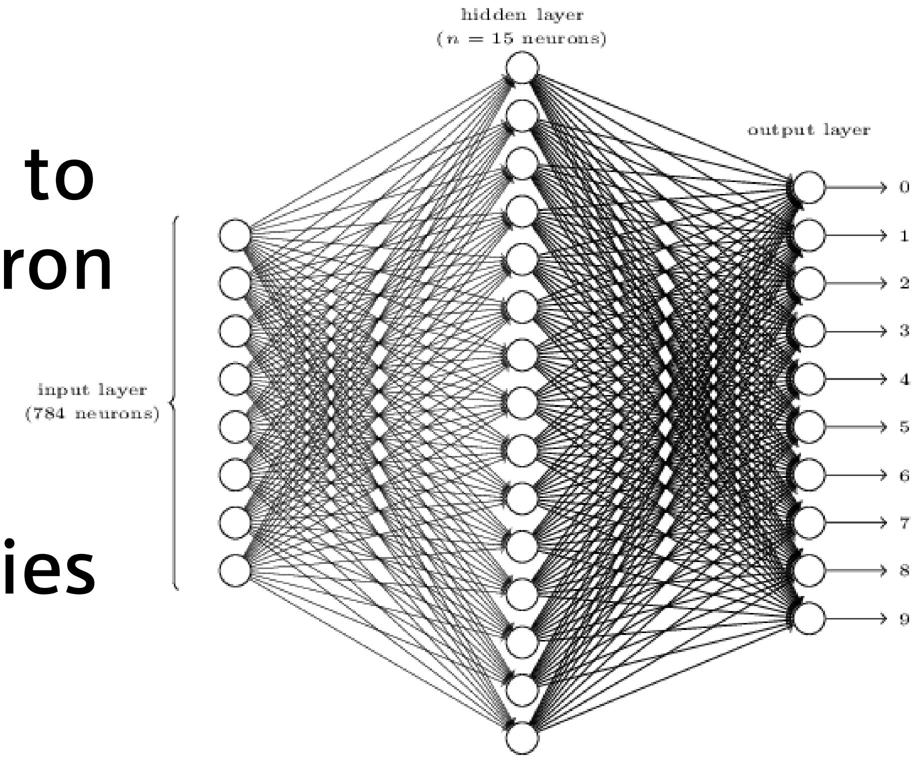
Example – Classify Handwritten Digits

Mathematical Modelling and Simulation

- Image with input pixel with $0 \leq p \leq 1$

- Output signal close to 1 then number neuron fires

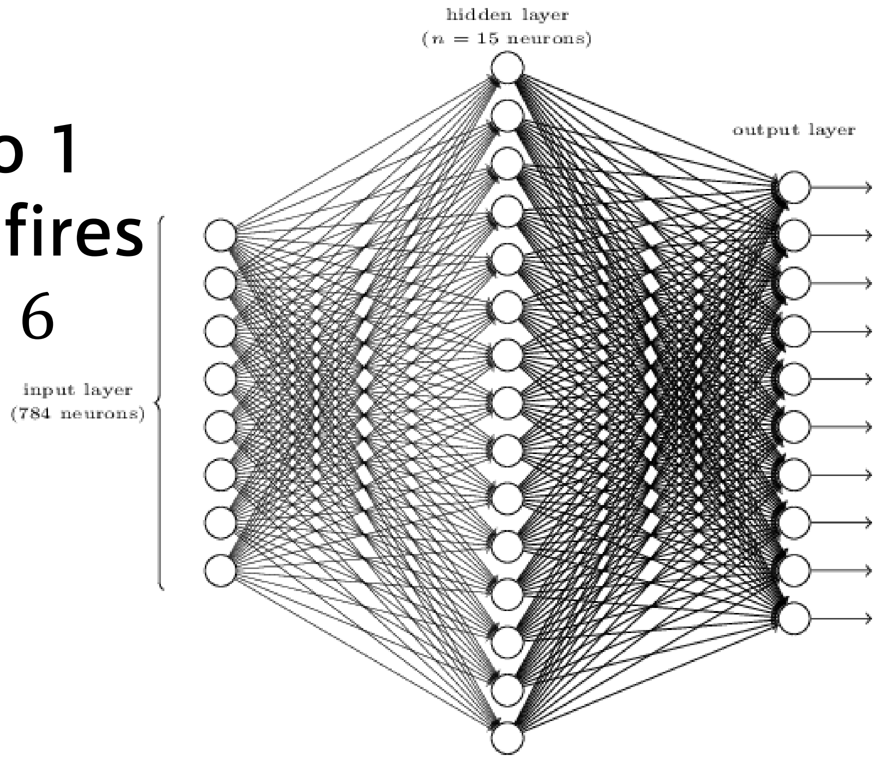
- Hidden layer classifies partial shapes



Example – Classify Handwritten Digits

- Input and output pairs
  $N: [0,1]^{784} \to [0,1]^{10}$

- Output signal close to 1 then number neuron fires
  $(0,0,0,0,0,1,0,0,0,0)^T \cong 6$



Neural Networks - Definitions

Mathematical Modelling and Simulation

Mostly used

- Feedforward
- MLP (multilayer perceptron)

Different approach

- Feedback loops => recurrent neural networks
- Activation like a wave
- More human like but harder to train

Neural Networks – Definitions

Deep learning

- More than one hidden layer
- More complex training
- Enable implementation of complex concepts

Online (incremental) learning

- Mini-patch size is 1
- „Learning by doing“

Neural Networks - Training

Mathematical Modelling and Simulation

How to determine weights and bias??

Through hard training

Training – Gradient Descent

Mathematical Modelling and Simulation

How to determine weights and bias??

- Creating/measuring input x output a pairs
- Define cost function

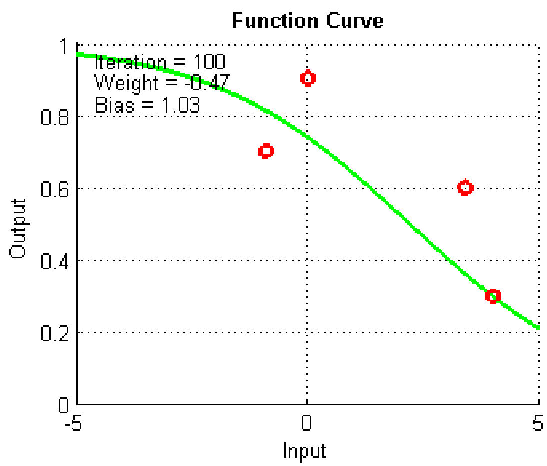
$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - a||^2$$

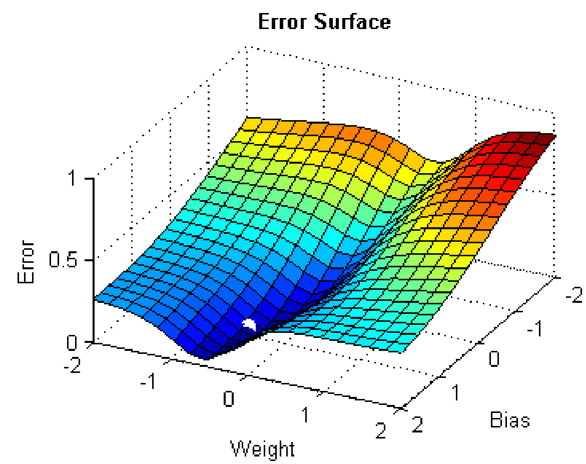
MSE

- solve minimisation problem:
  Use *gradient descent* for adjusting w and b

Training – Gradient Descent

Imagine a ball in a valley searching for the global minimum





Choose $(\triangle w, \triangle b)$ so that $\triangle C \approx \frac{\partial C}{\partial w} \triangle w + \frac{\partial C}{\partial b} \triangle b$ is negative

Training – Gradient Descent

Mathematical Modelling and Simulation

with $\Delta v = (\Delta w, \Delta b)^T$ and $\nabla C = \left(\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}\right)^T$

reformulate

$$\Delta C = \frac{\partial C}{\partial w}\Delta w + \frac{\partial C}{\partial b}\Delta b = \nabla C \cdot \Delta v$$

choosing

$$\Delta v = -\eta \nabla C$$

to guarantee negativity

η ... learning rate

$$\boxed{v \to v' = v - \eta \nabla C}$$

Training – Gradient Descent

Mathematical Modelling and Simulation

The resulting update rule for weights and bias

$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_k \to b_k' = b_k - \eta \frac{\partial C}{\partial b_k}$$

Necessary to get $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_k}$

Training – Gradient Descent

Mathematical Modelling and Simulation

$C = \frac{1}{n} \sum_x c_x$ for all n inputs x

With $C_x = \frac{||y(x) - a||^2}{2}$

$$C(a) = \frac{1}{2n} \sum_x ||y(x) - a||^2$$

$$\nabla C = \frac{1}{n} \sum_x \nabla C_x$$

Training – Stochastic Gradient Descent

Mathematical Modelling and Simulation

Taking only part of the inputs x

$\Longrightarrow$ a mini-patch size $m < n$

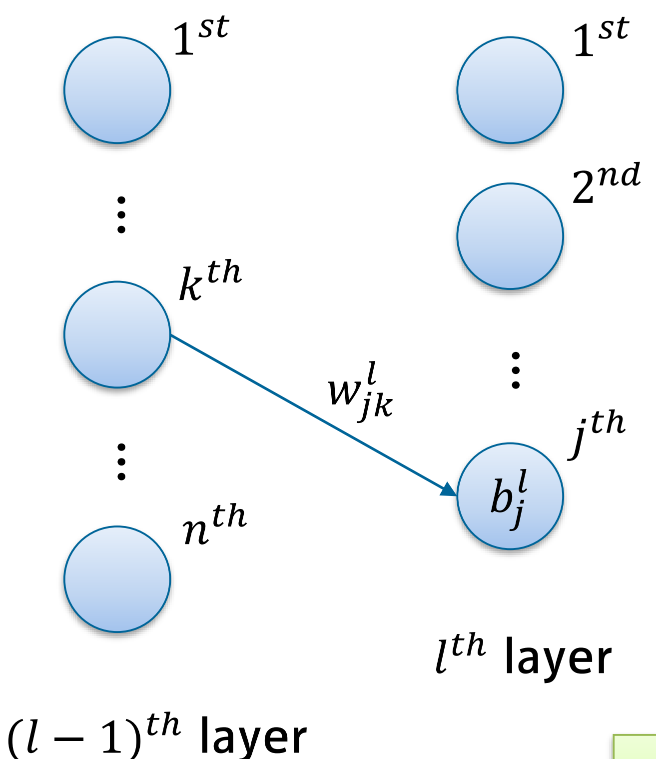
$$\nabla C = \frac{1}{m} \sum_{j=1}^{m} \nabla C_{x_j}$$

Going through all mini-patches in input

$\Rightarrow$ One epoch

Training – Backpropagation

Mathematical Modelling and Simulation



$$a_j^l = \sigma\Big(\sum_k w_{jk}^l\, a_k^{l-1} + b_j^l\Big)$$

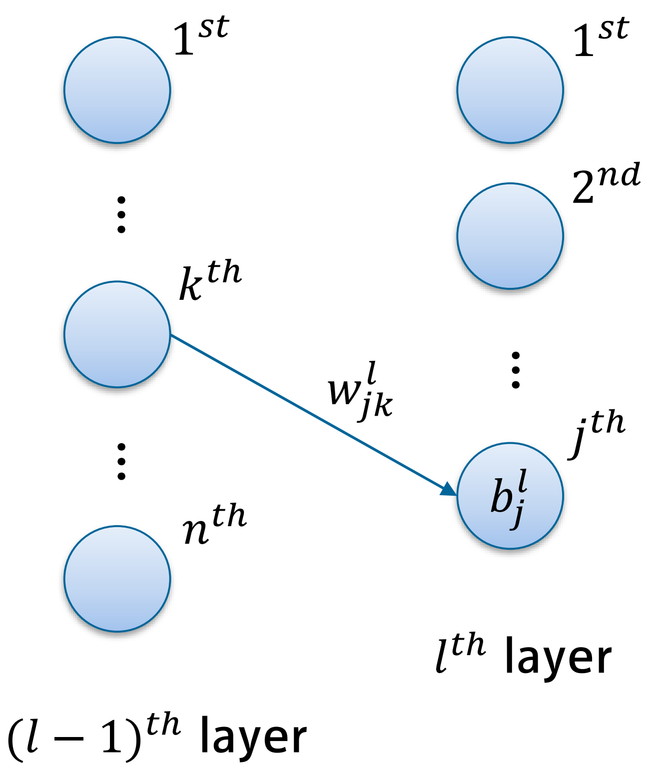
$$a^l = \sigma\left(w^l a^{l-1} + b^l\right) = \sigma\left(z^l\right)$$

$$l \in \{2, \ldots L\}$$

$$C_x = \frac{1}{2}\left\|y - a^L\right\|^2 = \frac{1}{2}\sum_j \left(y_j - a_j^L\right)^2$$

Training – Backpropagation

Mathematical Modelling and Simulation



Changing w^l_{jk} a little propagates through later layers

Calculating the Error the network beginning with the final output

Training – Backpropagation

δ_j^l ... calculate error done by the j^{th} neuron in the l^{th} layer using

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

($\frac{\partial C}{\partial a_j^l}$ is also possible but more complicated)

How do we calculate that error in detail??

Training – Backpropagation

Mathematical Modelling and Simulation

Starting with the last layer

$$\delta_j^L = \frac{\partial C}{\partial z_j^L} = \cdots = \left(a_j^L - y_j\right) \cdot \sigma'(z_j^l)$$

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

Hadamard Product: $s \odot t$

$$s \in \mathbb{R}^j, t \in \mathbb{R}^j : (s \odot \mathrm{t})_{\mathrm{j}} = \mathrm{s}_{\mathrm{j}} \cdot t_j$$

Continuing with the other layers

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

$$\delta_j^l = \cdots = \left(w^{l+1}\right)^T \delta^{l+1} \odot \sigma'(z^l)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \qquad \frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

Ready to implement our backpropagation!

Training – Backpropagation Algorithm

Mathematical Modelling and Simulation

Input x … set $a^1 = x$

Feedforward: $z^l = w^l a^{l-1} + b^l,\ a^l = \sigma\left(z^l\right),\ l \in \{2, \dots, L\}$

Output error: $\delta^L = (a^L - y) \odot \sigma'(z^L)$

Backpropagate: $\delta^l = \left(w^{l+1}\right)^T \delta^{l+1} \odot \sigma'\left(z^l\right),\ \ l \in \{L-1, L-2, \dots, 2\}$

Output: $\frac{\partial C}{\partial b_j^l} = \delta_j^l$ and $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Training – Algorithm

Mathematical Modelling and Simulation

Input a set of trainings data

For each training data x:

set $a^{x,1} = x$

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

Output error: $\delta^{x,L} = (a^{x,L} - y) \odot \sigma'(z^{x,L})$

Backpropagate: $\delta^{x,l} = \left(w^{l+1}\right)^T \delta^{x,l+1} \odot \sigma'(z^{x,l})$

Gradient descent:

$w^l \rightarrow w^l - \frac{\eta}{n} \sum_x \delta^{x,l} \left(a^{x,l-1}\right)^T$

$b^l \rightarrow b^l - \frac{\eta}{n} \sum_x \delta^{x,l}$

Neural Network – Definitions

Hyper Parameters

- Hidden neurons
- Learning rate
- Mini-patch size
- Epochs

History of Backpropagation

- 1970s : first applied but not appreciated
- 1986 : in paper of Rumelhart, Hilton & Williams published and accepted
- Now a days: classical “workhorse”

Neural Network – History

Mathematical Modelling and Simulation

History of deep neural networks

- 1950s & 60s : early approaches of networks with perceptron
- 1980s & 90s : already used backpropagation to train deep networks with stochastic gradient descent
- 2006 : 5-10 hidden layers trainable
- 2017 (11) : sigmoid function was replaced by the linear rectifier (especially in deep learning)

References

- Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015
  http://neuralnetworksanddeeplearning.com/chap1.html

- https://towardsdatascience.com/improving-vanilla-gradient-descent-f9d91031ab1d?gi=1a44ba5ceb4a