Data Mining - Neural Networks

Dr. Jean-Michel RICHER

FACULTÉ DES SCIENCES
Unité de formation et de recherche
DÉPARTEMENT INFORMATIQUE

université angers

2018

jean-michel.richer@univ-angers.fr

Outline

1. Introduction

What we will cover

- basics of Artificial Neural Networks
- the perceptron
- the multi-layer network
- the sigmoid function
- backpropagation
- Synaptic.js

2. History and working principle

FACULTÉ DES SCIENCES
Unité de formation et de recherche
DÉPARTEMENT INFORMATIQUE

Artificial Neural Networks

- NNs, ANNs or **Connectionist Systems** are computing systems inspired by the biological neural networks that constitute animal brains
- based on a collection of connected units or nodes called artificial neurons
- they try to model how neurons in the brain function
- such systems **learn** or progressively improve their performance by considering examples (training phase)

Note: strong and weak AI, intelligence = calculation ?

Specific Artificial Neural Networks

- for **image** recognition: **Convolutional Neural Network** (CNN or ConvNet), a variation of multilayer perceptrons designed to require minimal preprocessing
- for **speech** recognition: **Time Delay Neural Network** (TDNN)

A first example: MNIST

- the MNIST database of handwritten digits of 28×28 pixels
- 784 inputs and 10 outputs
- database of 60.000 examples and a test set of 10.000
- smallest error rate of 0.35% with 6-layers NN (Ciresan et al., 2010)
- smallest error rate of 0.23% with Convolutional Network (Ciresan et al., 2012)

McCulloch and Pitts, 1943

- Warren S. McCulloch, a neuroscientist, and Walter Pitts, a logician explain the complex decision processes in a brain using a linear threshold gate
- takes a sum and returns 0 if the result is below the threshold and 1 otherwise.
- very simple: binary inputs and outputs, threshold step activation function, no weighting of inputs

Donald O. Hebb, 1949

- Hebbian rule basis of nearly all neural learning procedures
- connection between two neurons is strengthened when both neurons are active at the same time
- this change in strength is proportional to the product of the two activities
- use weights

Rosenblatt, 1958

- Frank Rosenblatt, a psychologist at Cornell, was working on understanding the comparatively simpler decision systems present in the eye of a fly, which underlie and determine its flee response.
- he proposed the idea of a Perceptron (Mark I Perceptron)
- an algorithm for pattern recognition
- simple input output relationship, modeled on a McCulloch-Pitts
- perceptron learning: weights are adjusted only when a pattern is misclassified

Bernard Widrow, Marcian E. Hoff, 1960

- professor Widrow and his student Hoff introduced the ADALINE (ADAptive LInear NEuron)
- a fast and precise adaptive learning system: least mean squares filter (LMS)
- delta rule: minimises the output error using (approximate) gradient descent
- found in nearly every analog telephone for real-time adaptive echo filtering

Note: Hoff received his master's degree from Stanford University in 1959 and his PhD in 1962, father of the microprocessor at Intel

Minsky and Papert, 1969

- Marvin Minsky and Seymour Papert led a campaign to discredit neural network research
- all neural networks suffer from the same *fatal flaw* as the perceptron (XOR)
- they left the impression that neural network research was a dead end

Note: Minsky (MIT) is known for co-founding the field of AI, Papert (MIT) developed the Logo programming language

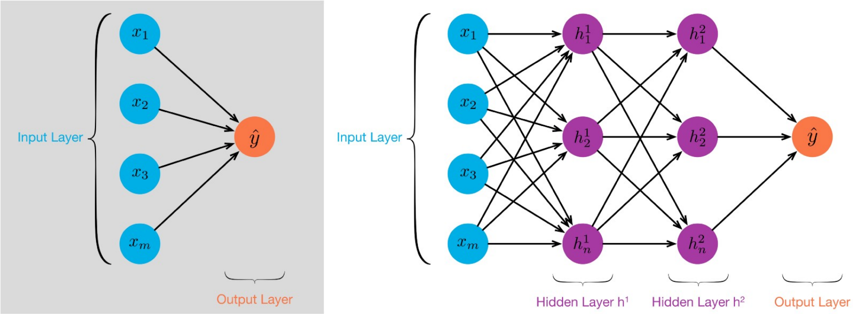
Paul Werbos, 1974

- in 1974 developed the back-propagation learning method although its importance wasn´t fully appreciated until a 1986
- accelerates the training of multi-layer networks
- input vector is applied to the network and propagated forward from the input layer to the hidden layer, and then to the output layer
- an error value is then calculated by using the desired output and the actual output for each output neuron in the network.
- the error value is propagated backward through the weights of the network beginning with the output neurons through the hidden layer and to the input layer

Geoffrey Hinton, David Rumelhart, Ronald Williams , 1986

- **Backpropagation**: repeatedly adjust the weights so as to minimize the difference between actual output and desired output
- **Hidden Layers**: neuron nodes stacked in between inputs and outputs, allow NN to learn more complicated features (such as XOR logic)

Figure: from the course of Nahua Kang on towardsdatascience.com



Deep Learning

- Deep Learning is about constructing machine learning models that learn a hierarchical representation of the data
- Neural Networks are a class of machine learning algorithms
- example: NVIDIA CUDA Deep Neural Network library (cuDNN) is a GPU-accelerated library of primitives for deep neural networks.

The Artificial Neuron

- connected with n input channels x_1 to x_n
- each has a synaptic weight w_1 to w_n
- there is a bias b
- use an activation function f_a

The output is defined as:

$$y = f_a(\sum_{i=1}^{n} x_i \times w_i + b)$$

Neuron formula

Can be modified by incorporating the bias into the $x_i \times w_i$

- set $x_0 = 1$
- $w_0 = b$

the formula becomes:

$$y = f_a\left(\sum_{i=0}^{n} x_i \times w_i\right)$$

The perceptron

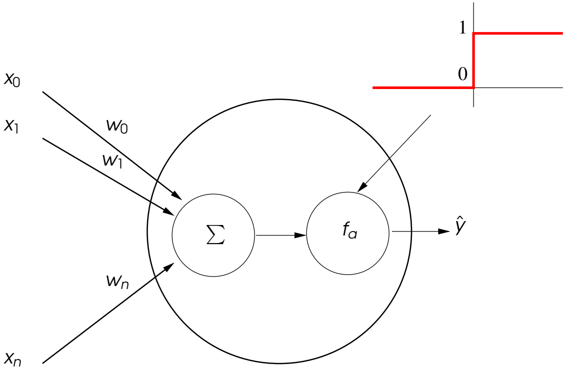
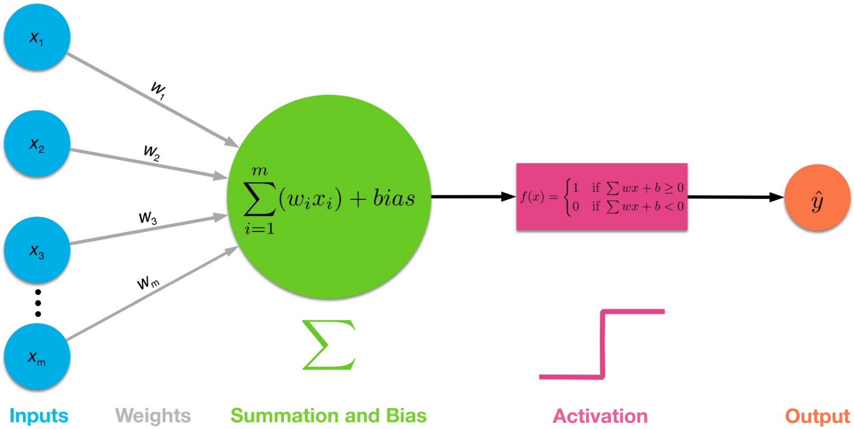


Figure: from the course of Nahua Kang on towardsdatascience.com



Neuron activation

The heaviside (thresold or binary) function is of the form

$$y = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i \times x_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

The perceptron is a simple model of prediction.

Learn with perceptron

initialize *w*;
while *not convergence* **do**
  compute errors;
  update *w* from errors;
end

Algorithm 1: Perceptron learning scheme

$$w_j = w_j + \eta(y - \hat{y}) \times x_j$$

where η is the learning constant (not too big, not too small) between 0.05 and 0.15

AND / OR

The perceptron can implement boolean formulas like the boolean OR or the AND

a	b	$a \vee b$	$a \wedge b$
0	0	0	0
0	1	1	0
1	0	1	0
1	0	1	1

Exemple with AND

$$X = \begin{bmatrix} x_0 & x_1 & x_2 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- $\eta = 0.1$
- $w = [0.1, 0.2, 0.05]$

Exemple with AND - first case

- take $X_0 = [1, 0, 0]$, $y_0 = 0$
- $\sum w_i \times x_i = 0.1 \times 1 + 0.2 \times 0 + 0.05 \times 0 = 0.1$
- $f_a(0.1) = 1 = \hat{y}$
- $w_0 = w_0 + \eta(y - \hat{y}) \times X_0^0 = 0.1 + 0.1 \times (0 - 1) \times 1 = \mathbf{0}$
- $w_1 = w_1 + \eta(y - \hat{y}) \times X_0^1 = 0.2 + 0.1 \times (0 - 1) \times 0 = 0.2$
- $w_2 = w_2 + \eta(y - \hat{y}) \times X_0^2 = 0.05 + 0.1 \times (0 - 1) \times 0 = 0.05$

continue with $X_1 = [1, 0, 1]$, ...

Exemple with AND - Convergence

- $w = [-0.30000001, 0.22, 0.10500001]$
- result is

x_0	x_1	x_2	y_p
1.	0.	0.	0
1.	0.	1.	0
1.	1.	0.	0
1.	1.	1.	1

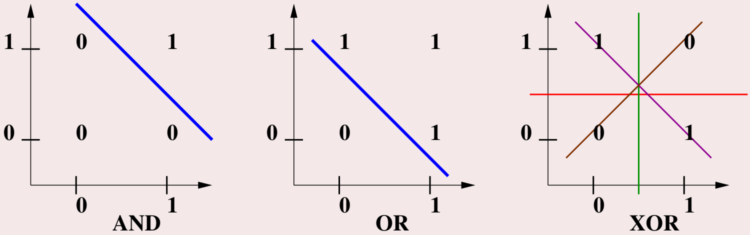
- It works !

Exercise

- Try to implement the perceptron in python, C++ or Java
- and test it for the boolean AND and OR

Why XOR is not possible with a perceptron ? (1/2)

The one layer perceptron acts as a linear separator:



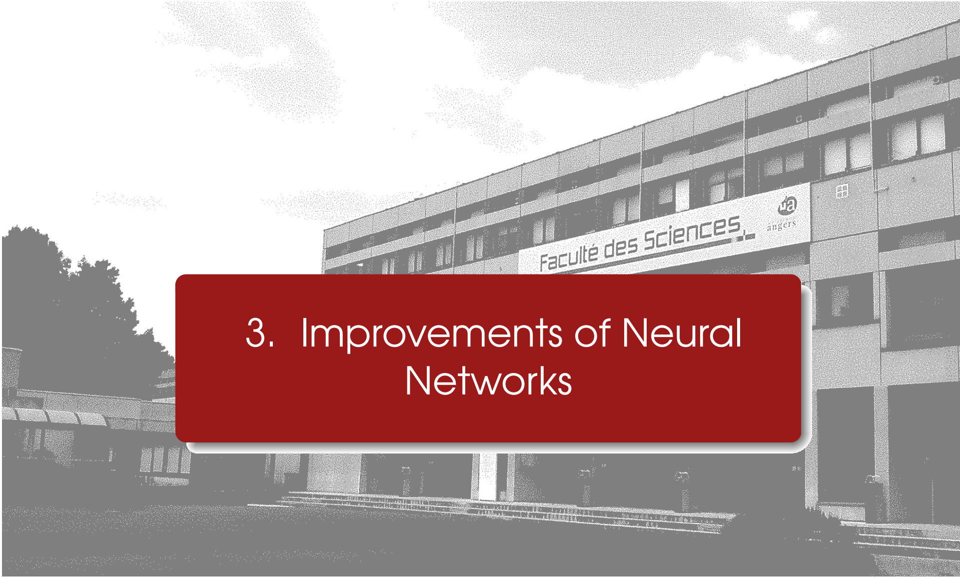
Why XOR is not possible with a perceptron ? (2/2)

a	b	a XOR b	equation	
0	0	0	$w_0 + 0 \times w_1 + 0 \times w_2 \leq 0$	(1)
0	1	1	$w_0 + 0 \times w_1 + 1 \times w_2 > 0$	(2)
1	0	1	$w_0 + 1 \times w_1 + 0 \times w_2 > 0$	(3)
1	1	0	$w_0 + 1 \times w_1 + 1 \times w_2 \leq 0$	(4)

adding (1) and (4) and then (2) and (3):

$$\begin{array}{ll} (1)+(4) & 2w_0 + w_1 + w_2 \leq 0 \\ (2)+(3) & 2w_0 + w_1 + w_2 > 0 \end{array}$$

impossible !



3. Improvements of Neural Networks

Heaviside problem

If the activation function is linear then the final output is still a linear combination of the input data

Sigmoid

A **sigmoid** function is a real function (special case of the logistic function):

- bounded (min, max)
- differentiable
- has a characteristic "S"-shaped curve

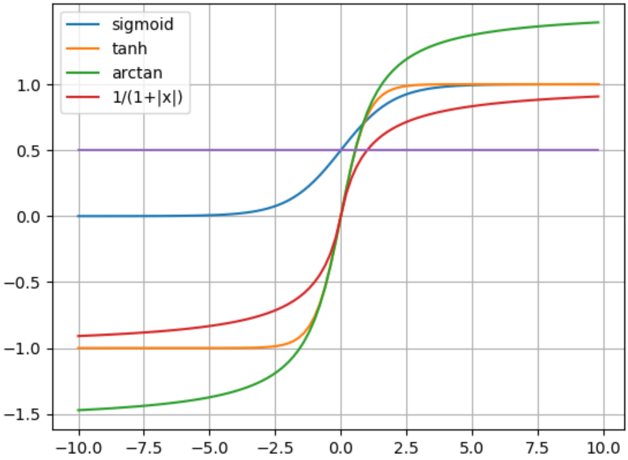
$$s(x) = \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

université angers
FACULTÉ DES SCIENCES
Unité de formation et de recherche
DÉPARTEMENT INFORMATIQUE

Other sigmoid-like functions

- hyperbolic tangent: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- arctangent function: $arctan(x)$
- error function: $\frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$

See wikipedia for a complete list

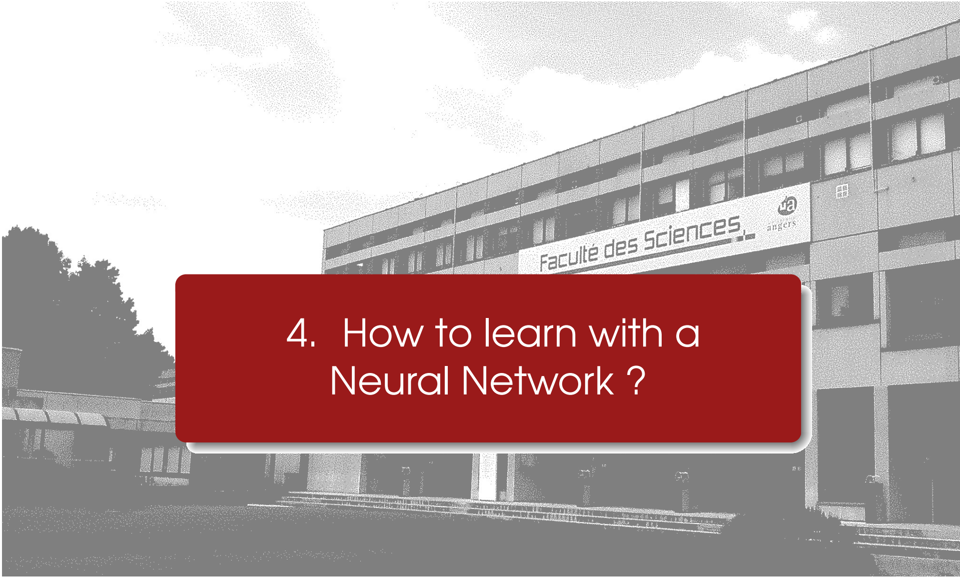

sigmoid
tanh
arctan
1/(1+|x|)
1.0
0.5
0.0
−0.5
−1.0
−1.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

Characteristic features of the sigmoid function

Properties of the sigmoid function

- output values range from 0 to 1
- the curve crosses 0.5 at $x = 0$
- simple derivative of $s(x) \times (1 - s(x))$
- used for models where you have to predict probability of an output

See math.stackexchange.com for demonstration of the derivative



4. How to learn with a Neural Network ?

Notations

we will use L to refer to a layer

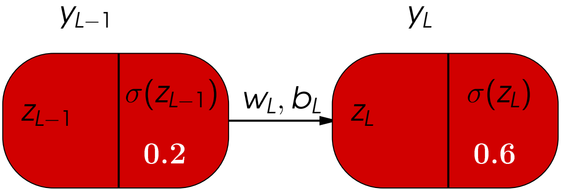
- y_L represents the output of layer L
- x_{L-1} represents the input layer for the computation of y_L
- w_L is the vector of weights

the output is then computed by

$$y_L = \sigma(w_L \times x_{L-1} + b_L)$$

where σ is the sigmoid activation function and b_L is the bias

The backpropagation



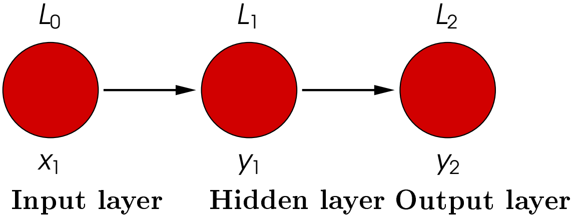
To simplify understanding we will write:

$$y_L = \sigma(z_L)$$

with

$$z_L = w_L \times x_{L-1} + b_L$$

Imagine you want to build a NN to implement the XOR function using a hidden layer:



we propagate the input values to the output layer:

$$y_1 = \sigma(w_1 \times x_0 + b_1)$$

$$y_2 = \sigma(w_2 \times y_1 + b_2)$$

We then can compare y_2 to the expected value y_{exp}

Error function and gradient

If y_2 and y_{exp} (the expected value for the output) are different we need to modify the w_i and b_i, for this we compute the error as:

$$E(y_2) = \frac{1}{2}(y_{exp} - y_2)^2$$

which in fact results from:

$$E(y_2) = \frac{1}{2}(y_{exp} - \sigma(W_2 \times \sigma(W_1 \times x_0 + b_1) + b_2)^2$$

and in fact the error depends of w_1, b_1, w_2, b_2

Error function and gradient

We will use the gradient of E to determine the influence of the w_L's and the bias b_L's:

$$\nabla E = \left(\frac{\partial E}{\partial w_L}, \quad \frac{\partial E}{\partial b_L} \right)$$

- $+\nabla E$ is the direction to **increase** the function
- $-\nabla E$ is the direction to **decrease** the function

How to compute $\frac{\partial E}{\partial w_L}$?

Remember that

$$
\begin{aligned}
z_L &= W_L \times y_{L-1} + b_L \\
y_L &= \sigma(z_L) \\
E &= \frac{1}{2}(y_{exp} - y_L)^2
\end{aligned}
$$

So the derivative of E with respect to w_L can be rewritten:

$$
\frac{\partial E}{\partial w_L} = \left(\frac{\partial E}{\partial y_L}\right)\left(\frac{\partial y_L}{\partial z_L}\right)\left(\frac{\partial z_L}{\partial w_L}\right)
$$

How to compute $\frac{\partial E}{\partial w_L}$?

Remember that

$$\begin{aligned} \frac{\partial E}{\partial y_L} &= \frac{1}{2} \times 2 \times (y_{exp} - y_L) \times -1 \\ \frac{\partial y_L}{\partial z_L} &= \sigma'(z_L) \\ \frac{\partial z_L}{\partial w_L} &= y_{L-1} \end{aligned}$$

So the derivative of E with respect to w_L is:

$$\frac{\partial E}{\partial w_L} = - \times (y_{exp} - y_L) \times \sigma'(z_L) \times y_{L-1}$$

The backpropagation

What about $\frac{\partial E}{\partial b_L}$?

Following the same demonstration, we get:

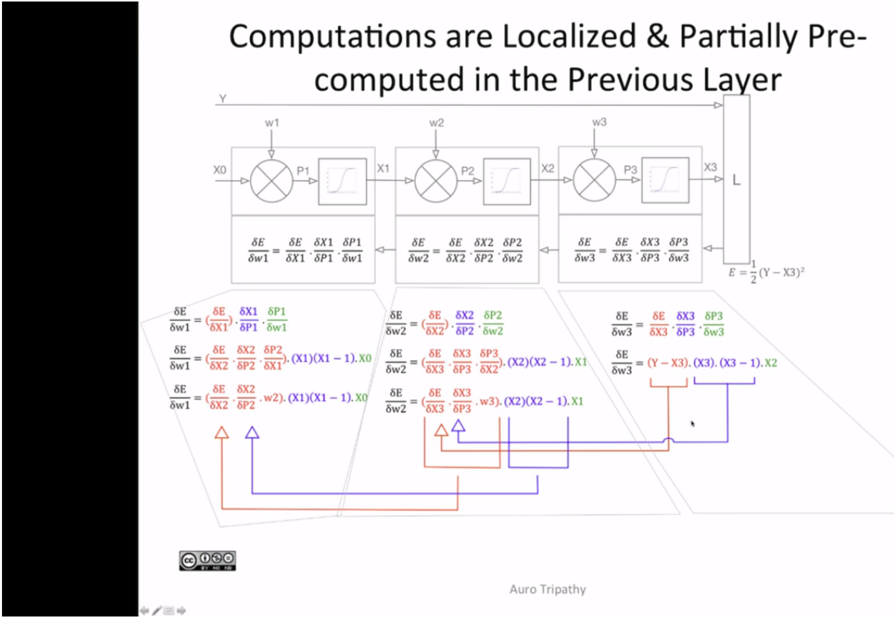
$$\frac{\partial E}{\partial b_L} = - \times (y_{exp} - y_L) \times \sigma'(z_L)$$

The backpropagation

Last step

- we have to sum all errors for each input data
- then propagate the change to the previous layer using the gradient:

```
L2_delta = L2_error * sigmoid_deriv(L2)
L1_error = L2_delta.dot(w2.T)
L1_delta = L1_error * sigmoid_deriv(L1)
w2 += L1.T.dot(L2_delta) * eta
w1 += L0.T.dot(L1_delta) * eta
```


Computations are Localized & Partially Pre-computed in the Previous Layer
Y
w1
w2
w3
X0
P1
X1
P2
X2
P3
X3
L
δE/δw1 = δE/δX1 · δX1/δP1 · δP1/δw1
δE/δw2 = δE/δX2 · δX2/δP2 · δP2/δw2
δE/δw3 = δE/δX3 · δX3/δP3 · δP3/δw3
E = 1/2 (Y − X3)²
δE/δw1 = (δE/δX1) · δX1/δP1 · δP1/δw1
δE/δw1 = (δE/δX2 · δX2/δP2 · δP2/δX1). (X1)(X1 − 1). X0
δE/δw1 = (δE/δX2 · δX2/δP2 . w2). (X1)(X1 − 1). X0
δE/δw2 = (δE/δX2) · δX2/δP2 · δP2/δw2
δE/δw2 = (δE/δX3 · δX3/δP3 · δP3/δX2). (X2)(X2 − 1). X1
δE/δw2 = (δE/δX3 · δX3/δP3 . w3). (X2)(X2 − 1). X1
δE/δw3 = δE/δX3 · δX3/δP3 · δP3/δw3
δE/δw3 = (Y − X3). (X3). (X3 − 1). X2
Auro Tripathy

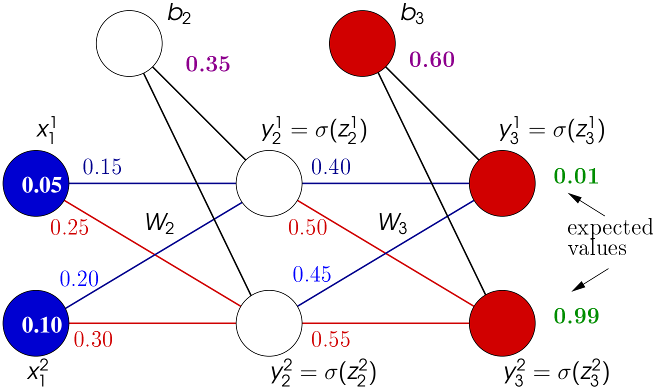
How to design a NN ?

Design of Neural Network

1. collect data (data structure ?)
2. normalize data
3. define training sets (Fold technique)
4. define a test set (or use one of the folds)
5. train the network using backpropagation
6. test result

Follow the tutorial of **Jason Brownlee** on the net called *How to Implement the Backpropagation Algorithm From Scratch In Python*, November 2016.

Backpropagation example



define W_2 and W_3 as matrices:

$$W_2 = \begin{bmatrix} 0.15 & 0.2 \\ 0.25 & 0.3 \end{bmatrix} = \begin{bmatrix} w_2^{1,1} & w_2^{1,2} \\ w_2^{2,1} & w_2^{2,2} \end{bmatrix}$$

$$W_3 = \begin{bmatrix} 0.4 & 0.45 \\ 0.5 & 0.55 \end{bmatrix} = \begin{bmatrix} w_3^{1,1} & w_3^{1,2} \\ w_3^{2,1} & w_3^{2,2} \end{bmatrix}$$

Propagate values of x_1^1 and x_1^2 by computing z_2^1 and y_2^1:

$$
\begin{aligned}
z_2^1 &= b_2 + w_2^{1,1} \times x_1^1 + w_2^{1,2} \times x_1^2 \\
z_2^1 &= 0.35 + 0.15 \times 0.05 + 0.2 \times 0.1 = 0.3775 \\
y_2^1 &= \sigma(z_2^1) \\
y_2^1 &= 1/(1 + e^{-0.3775}) = 0.5932
\end{aligned}
$$

Repeat the process for z_2^2 and y_2^2:

$$
\begin{aligned}
z_2^2 &= b_2 + w_2^{2,1} \times x_1^1 + w_2^{2,2} \times x_1^2 \\
z_2^2 &= 0.35 + 0.25 \times 0.05 + 0.3 \times 0.1 \quad = 0.3925 \\
y_2^2 &= \sigma(z_2^1) \\
y_2^2 &= 1/(1 + e^{-0.3925}) \quad = 0.5968
\end{aligned}
$$

To simplify the computation we coud write:

$$\begin{bmatrix} z_2^1 \\ z_2^2 \end{bmatrix} = \underbrace{\begin{bmatrix} w_2^{1,1} & w_2^{1,2} \\ w_2^{2,1} & w_2^{2,2} \end{bmatrix}}_{W_2} \times \underbrace{\begin{bmatrix} x_2^1 \\ x_2^2 \end{bmatrix}}_{X_1} + \underbrace{b_2 \times \begin{bmatrix} 1 \\ 1 \end{bmatrix}}_{B_2}$$

or

$$Z_2 = W_2 \times X_1 + B_2$$

and then

$$Y_2 = \sigma(Z_2)$$

Backpropagation example

Propagate values of y_2^1 and y_2^2 by computing z_3^1 and y_3^1:

$$
\begin{aligned}
z_3^1 &= b_3 + w_3^{1,1} \times y_2^1 + w_3^{1,2} \times y_2^2 \\
z_3^1 &= 0.6 + 0.4 \times 0.5932 + 0.45 \times 0.5968 \quad = 1.1059 \\
y_3^1 &= \sigma(z_3^1) \\
y_3^1 &= 1/(1 + e^{-1.1059}) \quad = 0.7513
\end{aligned}
$$

Backpropagation example

Do the same for z_3^2 and y_3^2:

$$
\begin{aligned}
z_3^2 &= b_3 + w_3^{2,1} \times y_2^1 + w_3^{2,2} \times y_2^2 \\
z_3^2 &= 0.6 + 0.5 \times 0.5932 + 0.55 \times 0.5968 &= 1.2249 \\
y_3^2 &= \sigma(z_3^1) \\
y_3^2 &= 1/(1 + e^{-1.2249}) &= 0.7729
\end{aligned}
$$

Backpropagation example

Compute the error of the network where y_{exp} is the vector of expected values:

$$E(y^3) = \frac{1}{2}\sum_{i=1}^{2}(y^i_{exp} - y^i_3)^2$$

$$E(y^3) = \frac{1}{2}\left((y^1_{exp} - y^1_3)^2 + (y^2_{exp} - y^2_3)^2\right)$$

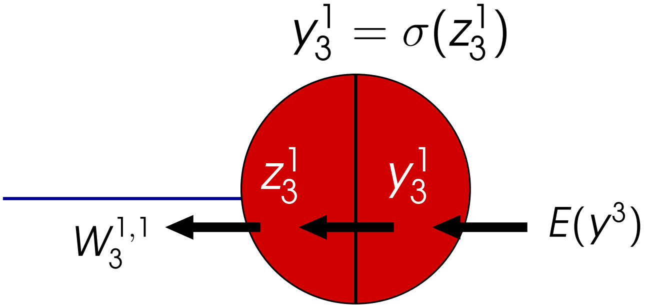
$$E(y^3) = \frac{1}{2}\left((0.01 - 0.7513)^2 + (0.99 - 0.7729)^2\right)$$

$$E(y^3) = \frac{1}{2}(0.5496 + 0.0471)$$

$$E(y^3) = \frac{1}{2}(0.5496 + 0.0471) = 0.2983$$

université angers | FACULTÉ DES SCIENCES Unité de formation et de recherche DÉPARTEMENT INFORMATIQUE

We need to compute the gradient of the error to update W_3:



Backpropagation example

We apply the *chain rule* for $w_3^{1,1}$:

$$\frac{\partial E(y^3)}{\partial w_3^{1,1}} = \frac{\partial E(y^3)}{\partial y_3^1} \times \frac{\partial y_3^1}{\partial z_3^1} \times \frac{\partial z_3^1}{\partial w_3^{1,1}}$$

where

$$\frac{\partial E(y^3)}{\partial y_3^1} = 2 \times \frac{1}{2} \times (y_{exp}^1 - y_3^1) \times -1$$

$$\frac{\partial y_3^1}{\partial z_3^1} = \sigma'(z_3^1) = y_3^1 \times (1 - y_3^1)$$

$$\frac{\partial z_3^1}{\partial w_3^{1,1}} = y_2^1$$

université angers — FACULTÉ DES SCIENCES Unité de formation et de recherche DÉPARTEMENT INFORMATIQUE

Backpropagation example

$$
\begin{aligned}
\frac{\partial E(y^3)}{\partial w_3^{1,1}} &= -(y_{exp}^1 - y_3^1) \times y_3^1 \times (1 - y_3^1) \times y_2^1 \\
&= -(0.01 - 0.7513) \times 0.7513 \times (1 - 0.7513) \times 0.5932 \\
&= 0.7413 \times 0.1868 \times 0.5932 \\
&= 0.0821
\end{aligned}
$$

Backpropagation example

For $w_3^{1,2}$:

$$\frac{\partial E(y^3)}{\partial w_3^{1,2}} = \frac{\partial E(y^3)}{\partial y_3^1} \times \frac{\partial y_3^1}{\partial z_3^1} \times \frac{\partial z_3^1}{\partial w_3^{1,2}}$$

where

$$\frac{\partial E(y^3)}{\partial y_3^1} = 2 \times \frac{1}{2} \times (y_{exp}^1 - y_3^1) \times -1$$

$$\frac{\partial y_3^1}{\partial z_3^1} = \sigma'(z_3^1) = y_3^1 \times (1 - y_3^1)$$

$$\frac{\partial z_3^1}{\partial w_3^{1,2}} = y_2^2$$

Backpropagation example

$$\begin{aligned}
\frac{\partial E(y^3)}{\partial w_3^{1,2}} &= -(y^1_{exp} - y^1_3) \times y^1_3 \times (1 - y^1_3) \times y^2_2 \\
&= -(0.01 - 0.7513) \times 0.7513 \times (1 - 0.7513) \times 0.5968 \\
&= 0.7413 \times 0.1868 \times 0.5968 \\
&= 0.0826
\end{aligned}$$

Backpropagation example

For $w_3^{2,1}$:

$$\frac{\partial E(y^3)}{\partial w_3^{2,1}} = \frac{\partial E(y^3)}{\partial y_3^2} \times \frac{\partial y_3^2}{\partial z_3^2} \times \frac{\partial z_3^2}{\partial w_3^{2,1}}$$

where

$$\frac{\partial E(y^3)}{\partial y_3^2} = 2 \times \frac{1}{2} \times (y_{exp}^2 - y_3^2) \times -1$$

$$\frac{\partial y_3^2}{\partial z_3^2} = \sigma'(z_3^2) = y_3^2 \times (1 - y_3^2)$$

$$\frac{\partial z_3^2}{\partial w_3^{2,1}} = y_2^1$$

Backpropagation example

$$
\begin{aligned}
\frac{\partial E(y^3)}{\partial w_3^{2,1}} &= -(y_{exp}^2 - y_3^2) \times y_3^2 \times (1 - y_3^2) \times y_2^1 \\
&= -(0.99 - 0.7729) \times 0.7729 \times (1 - 0.7729) \times 0.5932 \\
&= -0.2171 \times 0.1755 \times 0.5932 \\
&= -0.02260
\end{aligned}
$$

Backpropagation example

For $w_3^{2,2}$:

$$\frac{\partial E(y^3)}{\partial w_3^{2,2}} = \frac{\partial E(y^3)}{\partial y_3^2} \times \frac{\partial y_3^2}{\partial z_3^2} \times \frac{\partial z_3^2}{\partial w_3^{2,2}}$$

where

$$\frac{\partial E(y^3)}{\partial y_3^2} = 2 \times \frac{1}{2} \times (y_{exp}^2 - y_3^2) \times -1$$

$$\frac{\partial y_3^2}{\partial z_3^2} = \sigma'(z_3^2) = y_3^2 \times (1 - y_3^2)$$

$$\frac{\partial z_3^2}{\partial w_3^{2,2}} = y_2^2$$

$$
\begin{aligned}
\frac{\partial E(y^3)}{\partial w_3^{2,1}} &= -(y_{exp}^2 - y_3^2) \times y_3^2 \times (1 - y_3^2) \times y_2^1 \\
&= -(0.99 - 0.7729) \times 0.7729 \times (1 - 0.7729) \times 0.5968 \\
&= -0.2171 \times 0.1755 \times 0.5968 \\
&= -0.02274
\end{aligned}
$$

We now can update W_3:

$$W_3^\star = W_3 - \eta \begin{bmatrix} 0.0821 & 0.0826 \\ -0.0226 & -0.0227 \end{bmatrix}$$

where η is the learning rate, we set it to 0.5 in this case

$$W_3^\star = \begin{bmatrix} 0.358916479717885 & 0.408666186076233 \\ 0.511301270238737 & 0.561370121107989 \end{bmatrix}$$

Limits of Neural Networks

- does the use of the gradient function gives the minimum ?
- like for Maximum Parsimony: does the minimum represent the best network ?
- number and size of the hidden layers ?

6. Interesting links and applications

to explore in greater depth the course, follow those links:

- a series of very intersting videos about NN: 3Blue1Brown
- Apple Watch Detects Signs of Diabetes
- Solving SpaceNet Road Detection Challenge With Deep Learning
- Deep neural network from scratch from Florian Courtial

There are many tookits for NN available for many languages:

- GPU computing: cuDNN (NVidia)
- Theano (University of Montreal)
- Tensorflow (Google)
- Caffe (Berkeley AI Research)
- MXNet (Microsoft, Nvidia, Intel, ...)
- many more on wikipedia

Synaptic.js

Synaptic.js defines itself as the javascript architecture-free neural network library for node.js and the browser

- you can easily define a NN
- train it efficiently
- integrate the code in a web page

NN

Prediction

0, 0	0.013703341441539466	
0, 1	0.9914248535051694	
1, 0	0.9914204053611276	
1, 1	0.007987048313245976	
training time:	122.69999999989523	ms
	Predict	

Synaptic.js for XOR

Manually

```
var manualTrainingSet = [
  { input: [0,0], output: [0] },
  { input: [0,1], output: [1] },
  { input: [1,0], output: [1] },
  { input: [1,1], output: [0] }
]
```

Generated

```
generatedTrainingSet = [];
for (var i = 0; i < 4; ++i) {
  var op1 = Math.trunc(i/2);
  var op2 = Math.trunc(i & 1);
  input=[op1 , op2];
  generatedTrainingSet.push({ input,
    "output": [Math.trunc(op1 ^ op2)] });
}
```

Training of the neural network

Don't use `myTrain.trainXOR()` which automatically provides a XOR training set, but use `train(..)`

```
// var trainingSet = manualTrainingSet;
var trainingSet = generatedTrainingSet;

myTrainer.train(trainingSet);
```

Neural Network for the IRIS dataset

Modify the example of the XOR network to create a network for the IRIS dataset

- take the IRIS dataset from WEKA and convert it to JSON
- load the JSON data into the web page using JQuery
- train the network and display the results

IRIS Neural Network

Prediction

result

```
149 successfully classified
0 (1.00, 0.00, 0.00) OK
1 (1.00, 0.00, 0.00) OK
2 (1.00, 0.00, 0.00) OK
3 (1.00, 0.00, 0.00) OK
4 (1.00, 0.00, 0.00) OK
5 (1.00, 0.00, 0.00) OK
6 (1.00, 0.00, 0.00) OK
7 (1.00, 0.00, 0.00) OK
8 (1.00, 0.00, 0.00) OK
```

training time: 1313.9999999984866 ms

Predict

FACULTÉ DES SCIENCES
Unité de formation et de recherche
DÉPARTEMENT INFORMATIQUE
université angers

JQuery

```
<script type="text/javascript"
  src="https://ajax.googleapis.com/ajax/libs/
    jquery/2.1.3/jquery.min.js">
</script>
<script type="text/javascript">
var trainingSet = [];

$(document).ready(function(){
   $.getJSON("iris.json", function(result){
        for (i in result) {
  ...
        }
...
  });
});
</script>
```

Normalization

To get the best results you need to normalize the data, for example:

- feature scaling:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- standard score:

$$x' = \frac{x - \mu}{\sigma}$$

  where μ and σ are respectively the mean and standard deviation of the data

6. End

université angers
FACULTÉ DES SCIENCES
Unité de formation et de recherche
DÉPARTEMENT INFORMATIQUE

UA - Angers
2 Boulevard Lavoisier 49045 Angers
Cedex 01
Tel: (+33) (0)2-41-73-50-72