Introduction to Deep Neural Networks

Esteban Vegas Ferran Reverter
Alex Sanchez

2023-05-01

Table of contents


```r
options(width=100)
if(!require("knitr")) install.packages("knitr")
library("knitr")
#getOption("width")
knitr::opts_chunk$set(comment=NA,echo = TRUE, cache=TRUE)
```

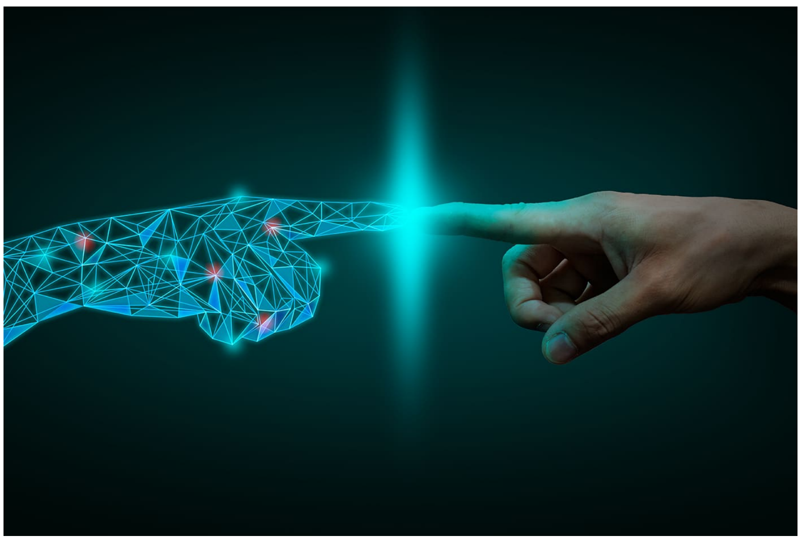
Introduction to Deep Neural Networks

Overview of Deep Learning

Historical Background and Key Milestones

Today, in April 2023, our world is convulsed by the explosion of Artificial Intelligence.

Although it has been growing steadily, it has probably been in the last months (weeks), since ChatGPT has arrived, that everybody has an opinion, or a fear on the topic.



While we are not going to discuss ethical, technological or sociological aspects, what seems clear to the data scientist's eye is that "*most of this is about prediction*".

Most AI engines rely on powerful prediction systems that use statistical learning methods.

Deep learning is a highly successful model in the field of AI, which has powered numerous applications in various domains. It has shown remarkable performance in tasks such as image recognition, natural language processing, and speech recognition.

Deep learning extends the basic principles of artificial neural networks by introducing more complex architectures and algorithms and, at the same time, by enabling machines to learn from large datasets by automatically identifying relevant patterns and features without explicit programming.

One key advantage of deep learning over traditional machine learning algorithms is its ability to handle high-dimensional and unstructured data such as images, videos, and audio.

The early history of artificial [neural networks]/intelligence

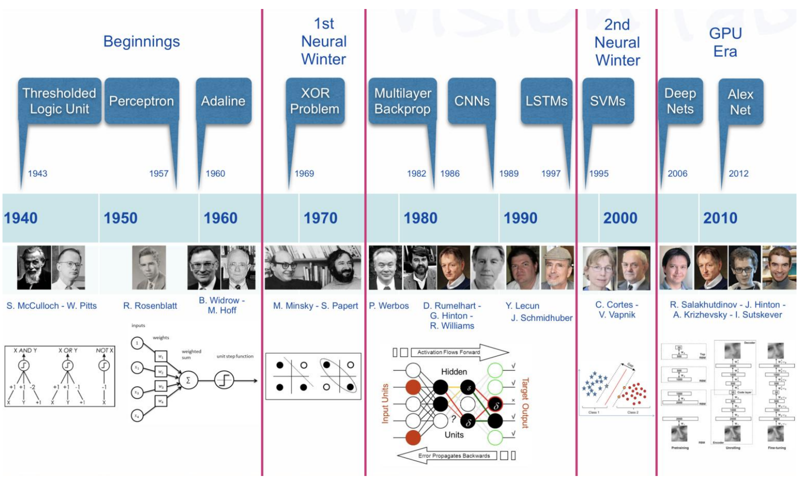


Figure 1: A Brief History of AI from 1940s till Today

The origins of AI, and as such of DL can be traced almost one century backwards. While it is an interesting, or even fascinating, history wee don't go into it (see a summary in A Quick History of AI, ML and DL

We can see there however, several hints worth to account for because we will go through them to understand how a deep neural network works. These are:

- The **Perceptron** and the first **Artificial Neural Network** where the basic building block was introduced.

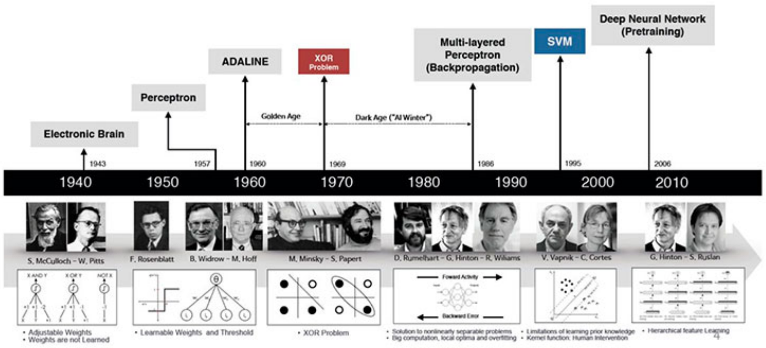


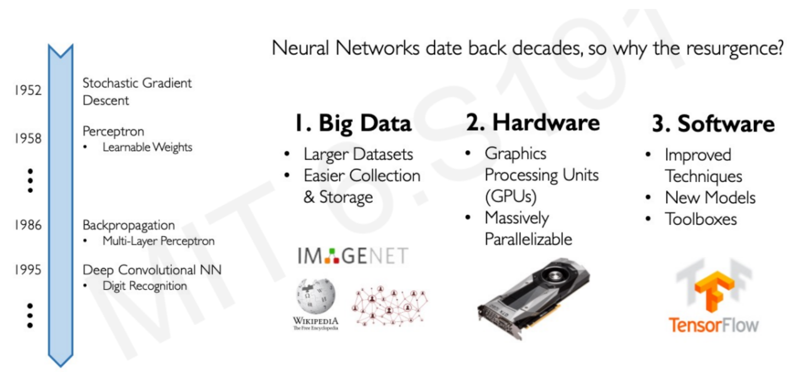
Figure 2: The origins of Deep learning and modern Artificial Intelligence can be traced back to the per4ceptron. Source: "A Quick History of AI, ML and DL"

- The **Multilayered perceptron** and **backpropagation** where complex architechtures were suggested to improve the capabilities.
- **Deep Neural Networks**, with many hidden layers, and auto-tunability capabilities.

In short, there has been an mathematical and a technological evolution that at some point has allowed to meet with

- The required theoretical background (DNN)
- The required computational capabilities (GPU, HPC)
- The required quantity of data (Big Data, Images, Social Networks)

This has resulted in making artificial intelligence widely accessible to businesses, researchers, and the general public.



Source: Alex Amini's 'MIT Introduction to Deep Learning' course (introtodeeplearning.com)

Success stories such as

- the development of self-driving cars,
- the use of AI in medical diagnosis, and
- the creation of personalized recommendations in online shopping

have also contributed to the widespread adoption of AI.

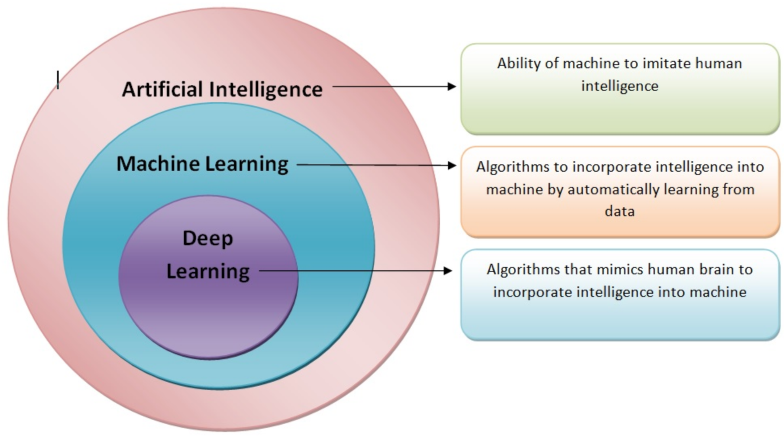
Comparison with Traditional Machine Learning

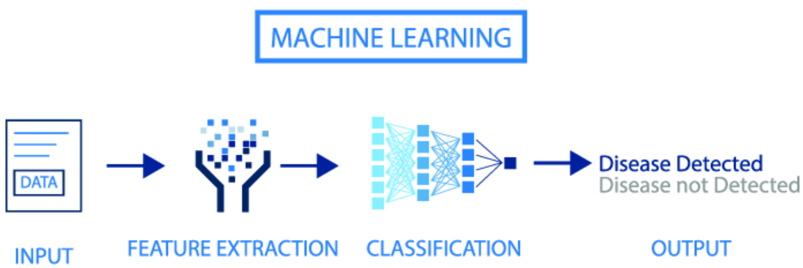
A reasonable question is "*how are ArtificiaI Intelligence, Machine Learning and Deep learning related*"?

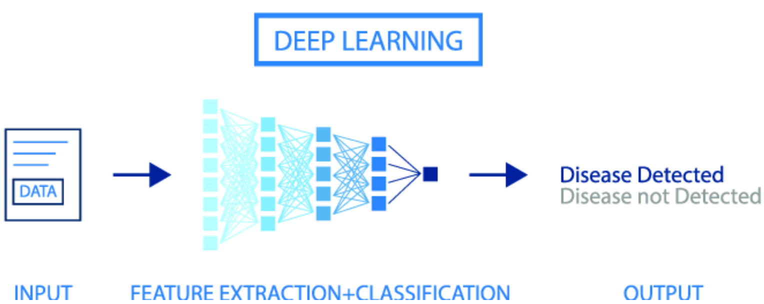
A standard answer can be found in the image below that has a myriad variations:

We can keep three definitions:

- Artificial intelligence is the ability of a computer to perform tasks commonly associated with intelligent beings.
- Machine learning is the study of algorithms that learn from examples and experience instead of relying on hard-coded rules and make predictions on new data
- Deep learning is a subfield of machine learning focusing on learning data representations as successive successive layers of increasingly meaningful representations.


Artificial Intelligence
Machine Learning
Deep Learning
Ability of machine to imitate human intelligence
Algorithms to incorporate intelligence into machine by automatically learning from data
Algorithms that mimics human brain to incorporate intelligence into machine


MACHINE LEARNING
DATA
Disease Detected
Disease not Detected
INPUT
FEATURE EXTRACTION
CLASSIFICATION
OUTPUT


DEEP LEARNING
DATA
Disease Detected
Disease not Detected
INPUT
FEATURE EXTRACTION+CLASSIFICATION
OUTPUT

We will be coming back to the difference between ML and DL, but two strengths of DL that differentiate it from ML should be clear from the beginning:

- DNN combine feature extraction and classification in a way that does not require (or dramatically decreases) human intervention.
- The power of DNN requires in its ability to improve with data availability, without seemingly reaching plateaus as ML.

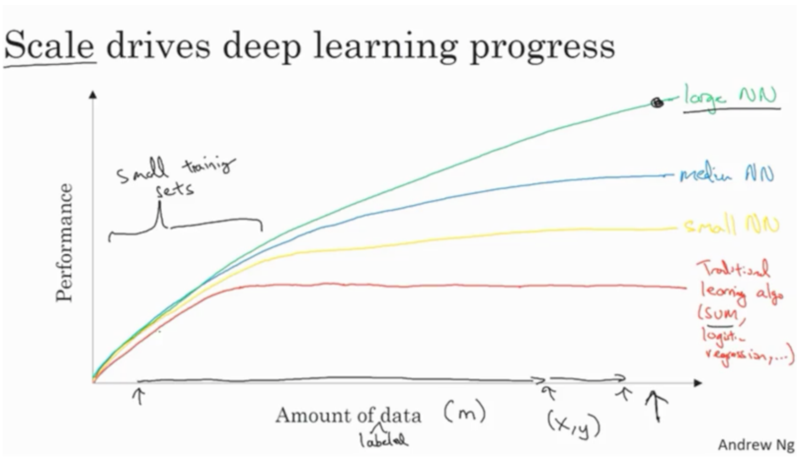


Figure 3: An illustration of the performance comparison between deep learning (DL) and other machine learning (ML) algorithms, where DL modeling from large amounts of data can increase the performance

Deep learning is having a strong impact

- Near-human-level image classification
- Near-human-level speech transcription
- Near-human-level handwriting transcription
- Dramatically improved machine translation
- Dramatically improved text-to-speech conversion
- Digital assistants such as Google Assistant and Amazon Alexa

- Near-human-level autonomous driving
- Improved ad targeting, as used by Google, Baidu, or Bing
- Improved search results on the web
- Ability to answer natural language questions
- Superhuman Go playing

According to [@chollet2022] … "*we shouldn't believe the short-term hype, but should believe in the long-term vision. It may take a while for AI to be deployed to its true potential—a potential the full extent of which no one has yet dared to dream—but AI is coming, and it will transform our world in a fantastic way*".

Once the introduction is ready we con move onto the building blocks of neural networks, perceptrons.

Artificial Neural Networks

The perceptron, the building block

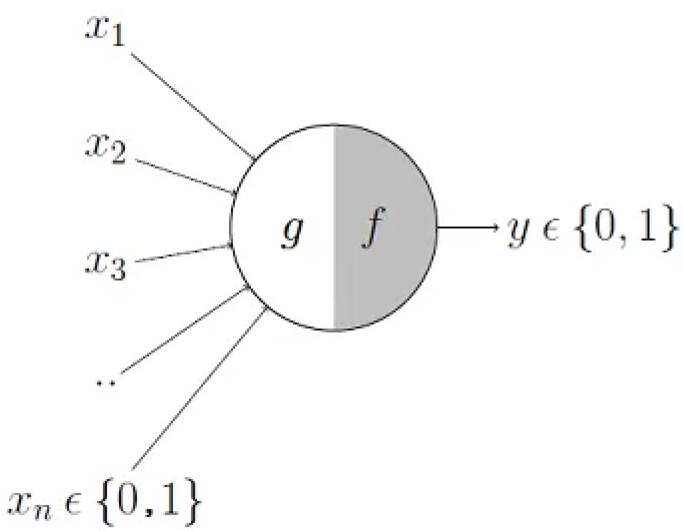
The perceptron, was introduced by Rosenblatt (one version of the perceptron at least), as a mathematical model that might emulate a neuron.

The idea is: *how can we produce a model that given some inputs, and an appropriate set of examples, learn to produce the desired output*?

The first computational model of a neuron was proposed by Warren MuCulloch (neuroscientist) and Walter Pitts (logician) in 1943.

It may be divided into 2 parts. The first part, g,takes an input (ahem dendrite ahem), performs an aggregation and based on the aggregated value the second part, f, makes a decision. See the source of this picture for an illustration on how this can be used to emulate logical operations such as AND, OR or NOT, but not XOR.

This first attempt to emulate neurons succeeded but with limitations:



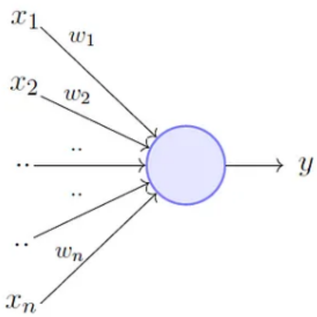
- What about non-boolean (say, real) inputs?
- What if all inputs are not equal?
- What if we want to assign more importance to some inputs?
- What about functions which are not linearly separable? Say XOR function

To overcome these limitations Frank Rosenblatt, an American psychologist, proposed the classical perception model, the *artificial neuron*, in 1958. It is more generalized computational model than the McCulloch-Pitts neuron where weights and thresholds can be learnt over time.

The perceptron proposed by Rosenblatt this is very similar to an M-P neuron but we take a weighted sum of the inputs and set the output as one only when the sum is more than an arbitrary threshold (***theta***).

Additionally, instead of hand coding the thresholding parameter θ, we add it as one of the inputs, with the weight $w_0 = -\theta$ like shown below, which makes it learnable.

Now, while this is an improvement (because both the weights and the threshold can be learned and the inputs can be real



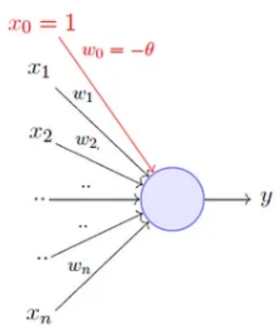
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$$

McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} x_i < 0$$

Perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

values) there is still a drawback in that a single perceptron can only be used to implement linearly separable functions.

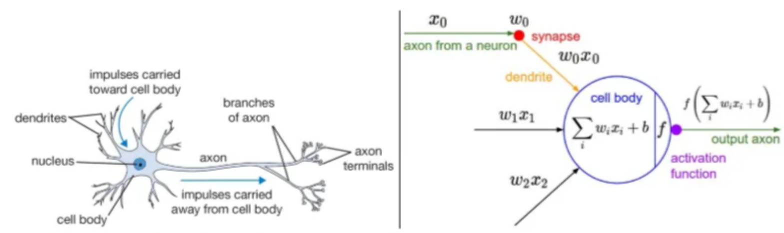
Artificial Neural Networks improve on this by introducing *Activation Functions*

Neurons and Activation Functions

An activation function is a function that is added into an artificial neurone in order to help it learn complex patterns in the data.

When comparing with a neuron-based model that is in our brains (when neurons are connected), the activation function is at the end deciding *what is to be fired to the next neuron.*

That is exactly what an activation function does in an ANN as well. It takes in the output signal from the previous cell and converts it into some form that can be taken as input to the next cell. The comparison can be summarized in the figure below.



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

.

With all these inputs in mind we can now define an Artificial Neuron as a *computational unit* that - takes as input $x = (x_0, x_1, x_2, x_3)$ ($x_0 = +1$, called bias), and - outputs $h_\theta(x) = f(\theta^\top x) = f(\sum_i \theta_i x_i)$, - where $f : \mathbb{R} \mapsto \mathbb{R}$ is called the **activation function**.

The goal of the activation function is to provide the Neuron with *the capability of producing the required outputs.*

For instance, if the output has to be a probability, the activation function will only produce values between 0 and 1.

With this idea in mind activation functions are chosen from a set of pre-defined functions:

- the sigmoid function:

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

- the hyperbolic tangent, or `tanh`, function:

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

The `tanh(z)` function is a rescaled version of the sigmoid, and its output range is $[-1, 1]$ instead of $[0, 1]$.

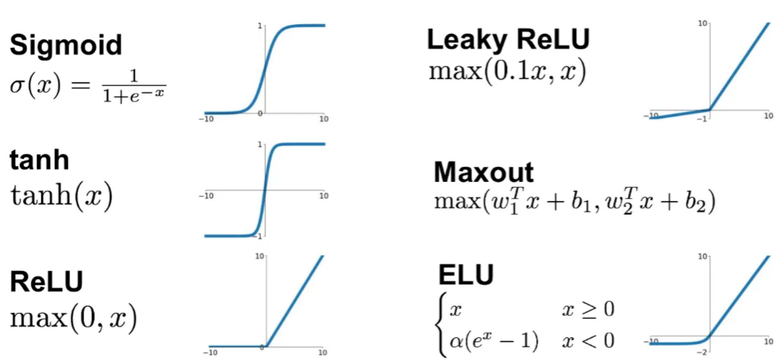
Two useful properties to recall are that: - *If* $f(z) = 1/(1 + e^{z})$ is the sigmoid function, then its derivative is given by $f'(z) = f(z)(1 - f(z))$.

- *Similarly, if* f is the `tanh` function, then its derivative is given by $f'(z) = 1 - (f(z))^2$.
- In modern neural networks, the default recommendation is to use the *rectified linear unit* or ReLU defined by the activation function $f(z) = \max\{0, z\}$.

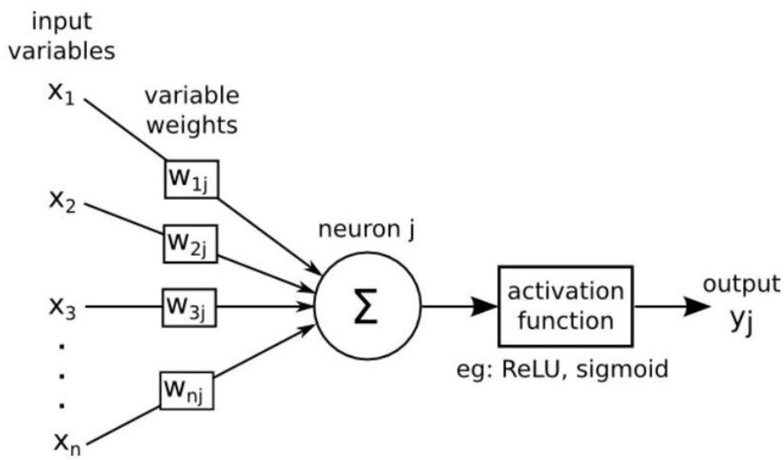
This function remains very close to a linear one, in the sense that is a piecewise linear function with two linear pieces.

Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimize with gradient based methods.

They also preserve many of the properties that make linear models generalize well.



Putting alltogether we have the following schematic representation of an artificial neuron where $\Sigma = \langle w_j, x \rangle + b_j$ and $\langle w_j, x \rangle$ represents the dot product between vectors w and x.

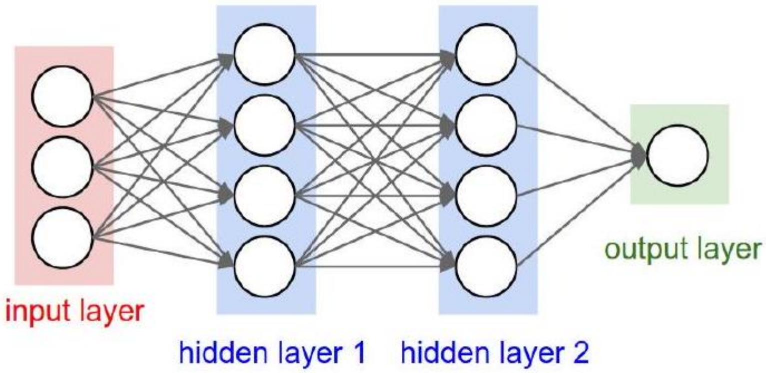


Multilayer perceptrons

A multilayer perceptron (or Artificial neural network) is a structure composed by *several hidden layers of neurons* where the output of a neuron of a layer becomes the input of a neuron of the next layer.

Moreover, the output of a neuron can also be the input of a neuron of the same layer or of neuron of previous layers (this is the case for recurrent neural networks). On last layer, called output layer, we may apply a different activation function as for the hidden layers depending on the type of problems we have at hand : regression or classification.

The Figure below represents a neural network with three input variables, one output variable, and two hidden layers.



Multilayers perceptrons have a basic architecture since each unit (or neuron) of a layer is linked to all the units of the next layer but has no link with the neurons of the same layer.

The parameters of the architecture are:

- the number of hidden layers and
- the number of neurons in each layer.

The activation functions are also to choose by the user. For the output layer, as mentioned previously, the activation function is generally different from the one used on the hidden layers. For example:.

- In the case of regression, we apply no activation function on the output layer.
- For binary classification, the output gives a prediction of $\mathbb{P}(Y = 1/X)$ since this value is in $[0,1]$ and the sigmoid activation function is generally considered.
- For multi-class classification, the output layer contains one neuron per class (i), giving a prediction of $\mathbb{P}(Y = i/X)$. The sum of all these values has to be equal to 1 . The sum of all these values has to be equal to 1 . The multidimensional function *softmax* is generally used

$$\operatorname{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Mathematical formulation of the ANN

We can summarize the mathematical formulation of a multi-layer perceptron with (L) hidden layers as follows:

- Set $h^{(0)}(x) = x$ For $k = 1, \dots, L$ (hidden layers),

$$
\begin{aligned}
a^{(k)}(x) &= b^{(k)} + W^{(k)} h^{(k-1)}(x) \\
h^{(k)}(x) &= \phi\left(a^{(k)}(x)\right)
\end{aligned}
$$

For $k = L + 1$ (output layer)

$$
\begin{aligned}
a^{(L+1)}(x) &= b^{(L+1)} + W^{(L+1)} h^{(L)}(x) \\
h^{(L+1)}(x) &= \psi\left(a^{(L+1)}(x)\right) := f(x, \theta).
\end{aligned}
$$

where ϕ is the activation function and ψ is the output layer activation function (for example softmax for multiclass classification).

At each step, $W^{(k)}$ is a matrix with number of rows equal to the number of neurons in the layer k and number of columns the number of neurons in the layer $k - 1$.

By organizing our parameters in matrices and using matrix-vector operations, we can take advantage of fast linear algebra routines to quickly perform calculations in our network.

An example

In this example we train and use a "shallow neural network", called this way in contrast with "deep neural networks".

We will use the `neuralnet` R package, which is not intended to work with deep neural networks, to build a simple neural network to predict if a type of stock pays dividends or not.

```
if (!require(neuralnet))
  install.packages("neuralnet", dep=TRUE)
```

The data for the example are the `dividendinfo.csv` dataset, available from: https://github.com/MGCodesandStats/datasets

```
mydata <- read.csv("https://raw.githubusercontent.com/MGCodesandStats/datasets/master/divide
str(mydata)
```

```
'data.frame':	200 obs. of  6 variables:
 $ dividend       : int  0 1 1 0 1 1 1 0 1 1 ...
 $ fcfps          : num  2.75 4.96 2.78 0.43 2.94 3.9 1.09 2.32 2.5 4.46 ...
 $ earnings_growth: num  -19.25 0.83 1.09 12.97 2.44 ...
 $ de             : num  1.11 1.09 0.19 1.7 1.83 0.46 2.32 3.34 3.15 3.33 ...
 $ mcap           : int  545 630 562 388 684 621 656 351 658 330 ...
 $ current_ratio  : num  0.924 1.469 1.976 1.942 2.487 ...
```

Data preprocessing

One of the most important procedures when forming a neural network is data normalization. This involves adjusting the data to a common scale so as to accurately compare predicted and actual values. Failure to normalize the data will typically result in the prediction value remaining the same across all observations, regardless of the input values.

We can do this in two ways in R:

- Scale the data frame automatically using the scale function in R
- Transform the data using a max-min normalization technique

In this example We implement the max-min normalization technique.

See this link for further details on how to use the normalization function.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

```
normData <- as.data.frame(lapply(mydata, normalize))
```

As usually, the dataset is separated in a training and a test set. The training set contains a random selection with and (arbitrary) 66% of the observations.

```
perc2Train <- 2/3
ssize <- nrow(normData)
set.seed(12345)
data_rows <- floor(perc2Train *ssize)
train_indices <- sample(c(1:ssize), data_rows)
trainset <- normData[train_indices,]
testset <- normData[-train_indices,]
```

The `trainset` set will be used to train the network and the `testset` set one will be used to evaluate it.

Training a neural network

Setting the parameters of a neural network requires experience and understanding of their meaning, and even so, canges in the parameters can lead to similar results.

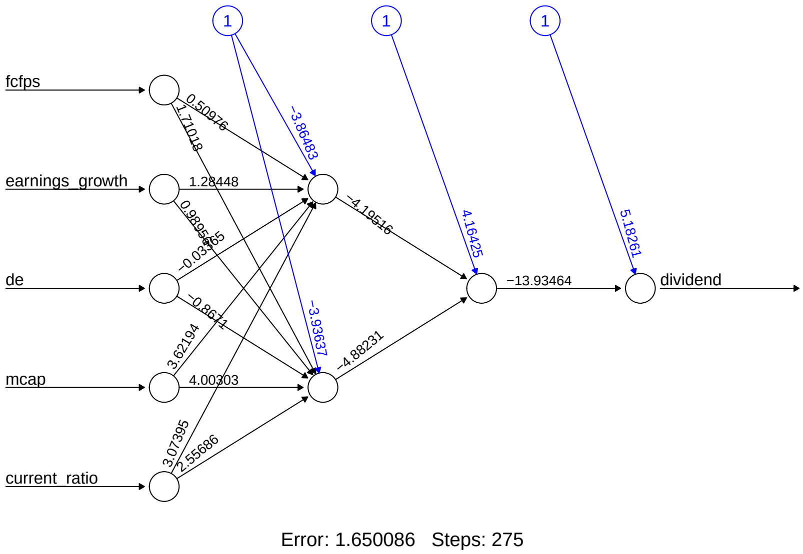
We create a simple NN with two hidden layers, with 4 and 2 neurons respectively. This is specified in the `hidden` parameter. For other parameters see the package help.

```
# Neural Network
library(neuralnet)
nn <- neuralnet(dividend ~ fcfps + earnings_growth + de + mcap + current_ratio,
                data=trainset,
                hidden=c(2,1),
                linear.output=FALSE,
                threshold=0.01)
```

The output of the procedure is a neural network with estimated weights.

This can be seen with a `plot` function (including the `rep` argument).

```r
plot(nn, rep = "best")
```



The object **nn**contains information the weights and the results although it is not particularly clear or useful.

```r
summary(nn)
```

```
                    Length Class      Mode
call                  6    -none-     call
response            133    -none-     numeric
covariate           665    -none-     numeric
model.list            2    -none-     list
err.fct               1    -none-     function
act.fct               1    -none-     function
linear.output         1    -none-     logical
data                  6    data.frame list
exclude               0    -none-     NULL
net.result            1    -none-     list
weights               1    -none-     list
generalized.weights   1    -none-     list
startweights          1    -none-     list
```

```
result.matrix        20     -none-      numeric
```

```
nn$result.matrix
```

```
                                       [,1]
error                           1.650085627
reached.threshold               0.008511491
steps                         275.000000000
Intercept.to.1layhid1          -3.864825441
fcfps.to.1layhid1               0.509757479
earnings_growth.to.1layhid1     1.284483123
de.to.1layhid1                 -0.033645969
mcap.to.1layhid1                3.621943288
current_ratio.to.1layhid1       3.073952492
Intercept.to.1layhid2          -3.936370246
fcfps.to.1layhid2               1.710175929
earnings_growth.to.1layhid2     0.989556920
de.to.1layhid2                 -0.867102084
mcap.to.1layhid2                4.003031001
current_ratio.to.1layhid2       2.556863158
Intercept.to.2layhid1           4.164248649
1layhid1.to.2layhid1           -4.195155060
1layhid2.to.2layhid1           -4.882313558
Intercept.to.dividend           5.182610336
2layhid1.to.dividend          -13.934644457
```

Model evaluation

A prediction for each value in the `testset` dataset can be built with the `compute` function.

```
#Test the resulting output
temp_test <- subset(testset, select =
                      c("fcfps","earnings_growth",
                        "de", "mcap", "current_ratio"))
head(temp_test)
```

```
       fcfps earnings_growth        de       mcap current_ratio
```

```
9  0.4929006      0.52417860 0.7862595 0.79741379   0.662994637
19 0.8722110      0.89705139 0.5190840 0.31465517   0.631284474
22 0.0811359      0.68272957 0.4554707 0.05747126   0.000785556
26 0.4077079      0.07649537 0.6310433 0.70977011   0.379642293
27 0.4279919      0.70362258 0.1882952 0.30603448   0.628283435
29 0.3509128      0.74203875 0.6030534 0.53017241   0.543404499
```

```r
nn.results <- compute(nn, temp_test)
results <- data.frame(actual =
                  testset$dividend,
                  prediction = nn.results$net.result)
head(results)
```

```
   actual   prediction
9       1 0.9919213885
19      1 0.9769206123
22      0 0.0002187144
26      0 0.6093330933
27      1 0.7454164893
29      1 0.9515431416
```

A confusion matrix can be built to evaluate the predictive ability of the network:

```r
roundedresults<-sapply(results,round,digits=0)
roundedresultsdf=data.frame(roundedresults)
attach(roundedresultsdf)
table(actual,prediction)
```

```
      prediction
actual  0  1
     0 33  3
     1  4 27
```