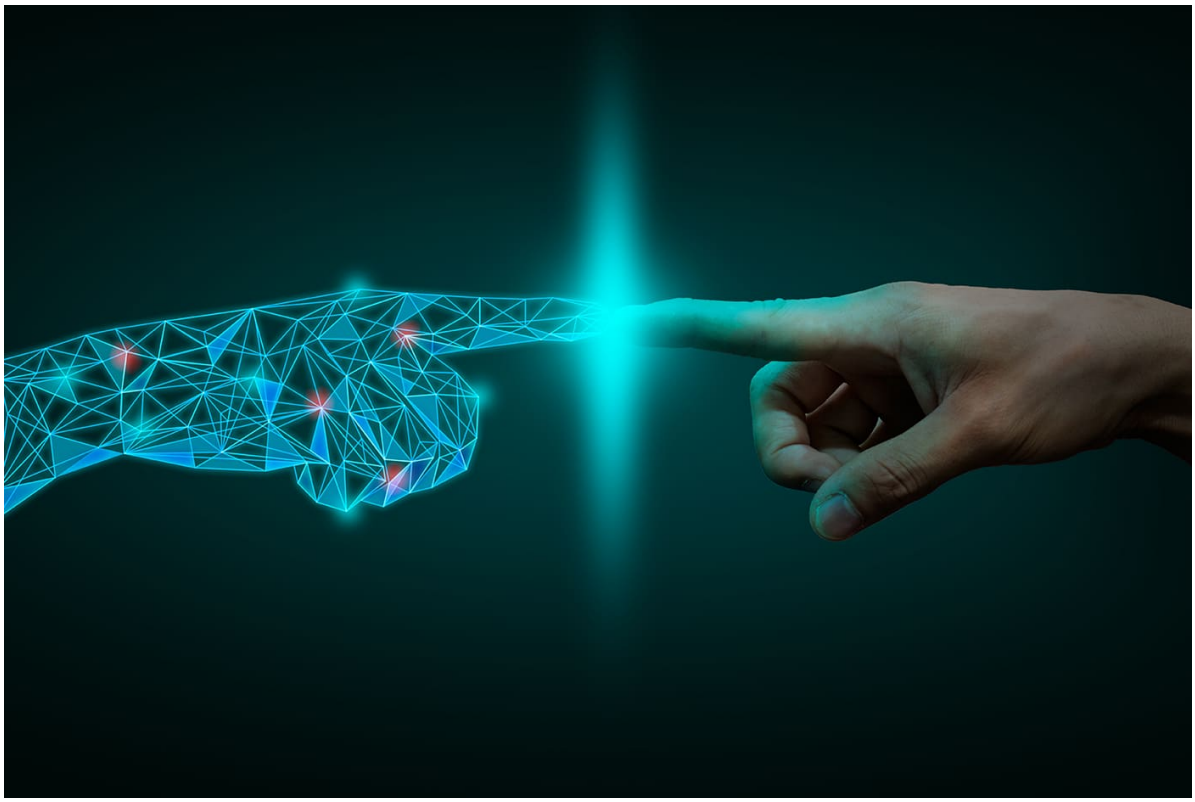
Introduction to Deep Neural Networks

A. Sanchez, F. Reverter and E. Vegas

Overview of Deep Learning

Historical Background (1)

- Today, in April 2023, our world is convulsed by the explosion of Artificial Intelligence.
- It has probably been in the last months (weeks), since ChatGPT has arrived, that everybody has an opinion, or a fear on the topic.



Historical Background (2)

- We don't discuss ethical, technological or sociological aspects, but is "*most of this is about prediction*".
- Most AI engines rely on powerful prediction systems that use statistical learning methods.

Deep learning

- Deep learning is a successful AI model which has powered many application such as *self-driving cars, voice assistants, and medical diagnosis systems.*
- Essentially, deep learning extends the basic principles of artificial neural networks by
  - Adding complex architectures and algorithms and
  - At the same time becoming more automatical
- We won't delve into the history of ANN, but a quick look at it may help fully grasp its current capabilities.

The early history of AI (1)

- The origins of AI, and as such of DL can be traced almost one century backwards;
- A Quick History of AI, ML and DL

Milestones in the history of DL

We can see several hints worth to account for:

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

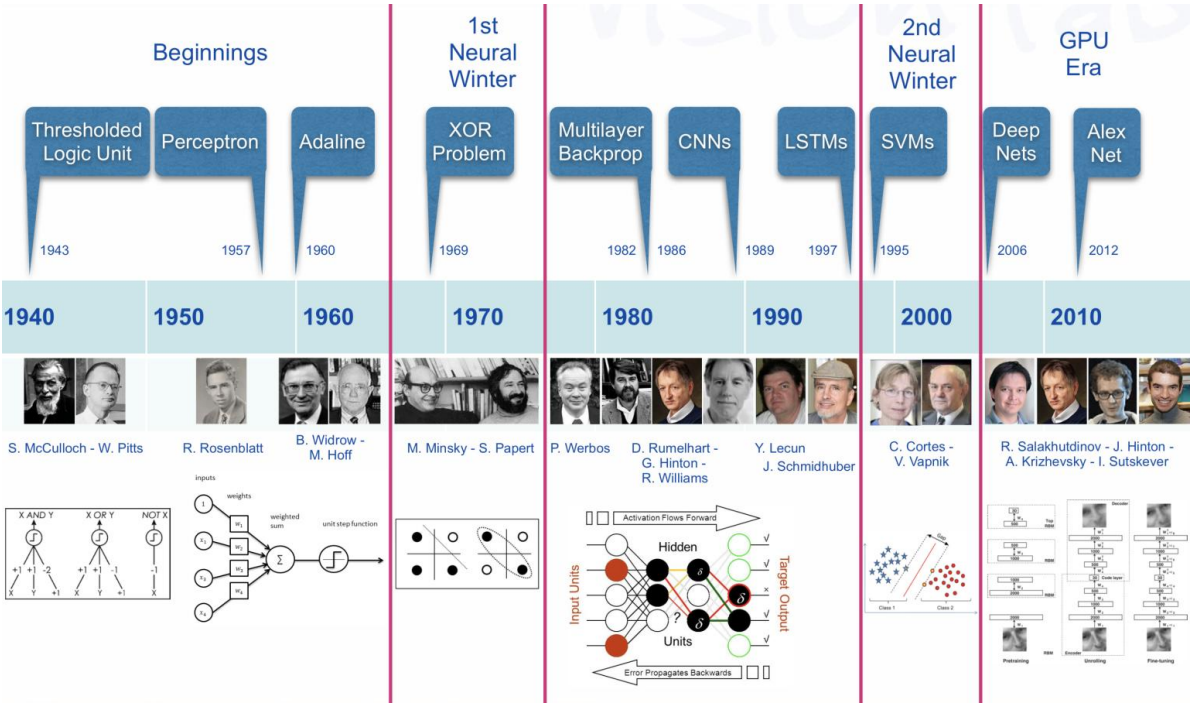
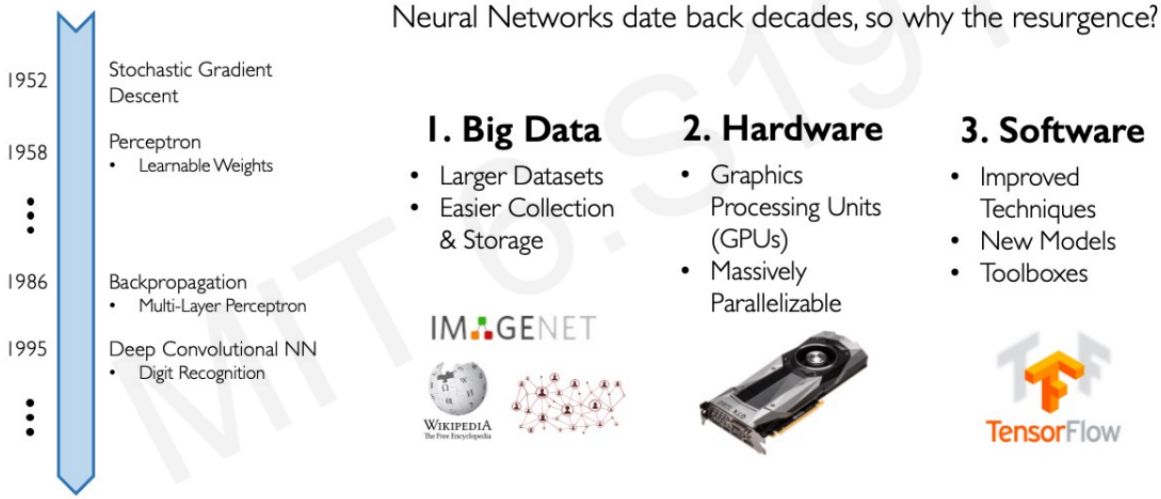


Figure 1: A Brief History of AI from 1940s till Today

From Artificial Neural Networks to Deep learning



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

Success stories

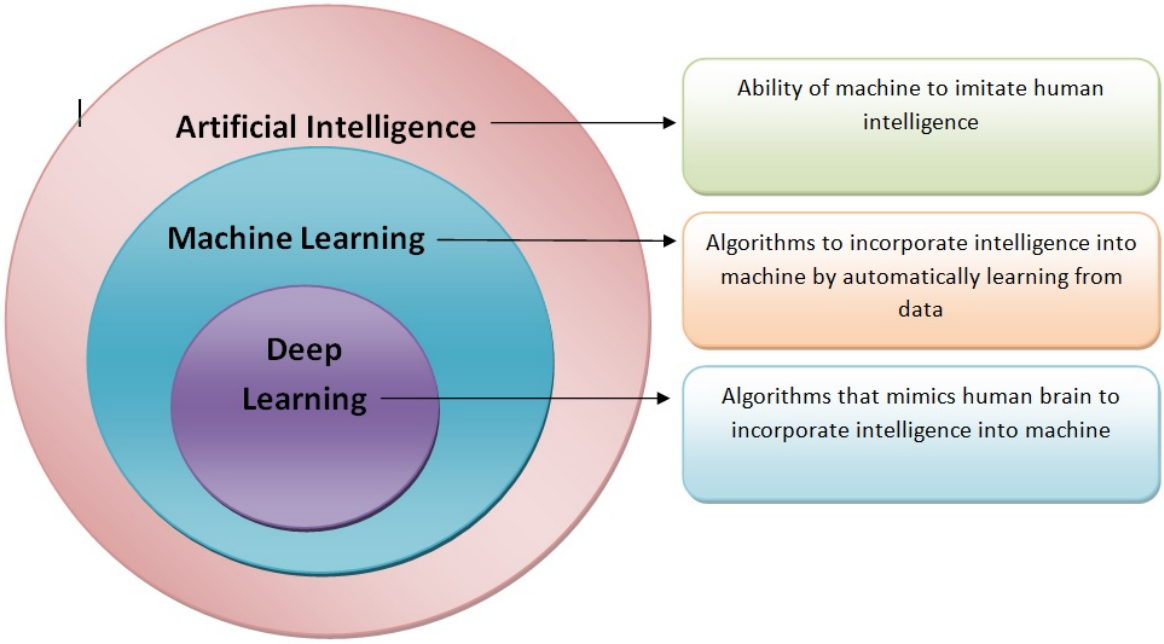
Success stories such as

- 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.

AI, ML, DL ...

```
knitr::include_graphics("images/AI-ML-DL-1.jpg")
```



AI, ML, DL ...

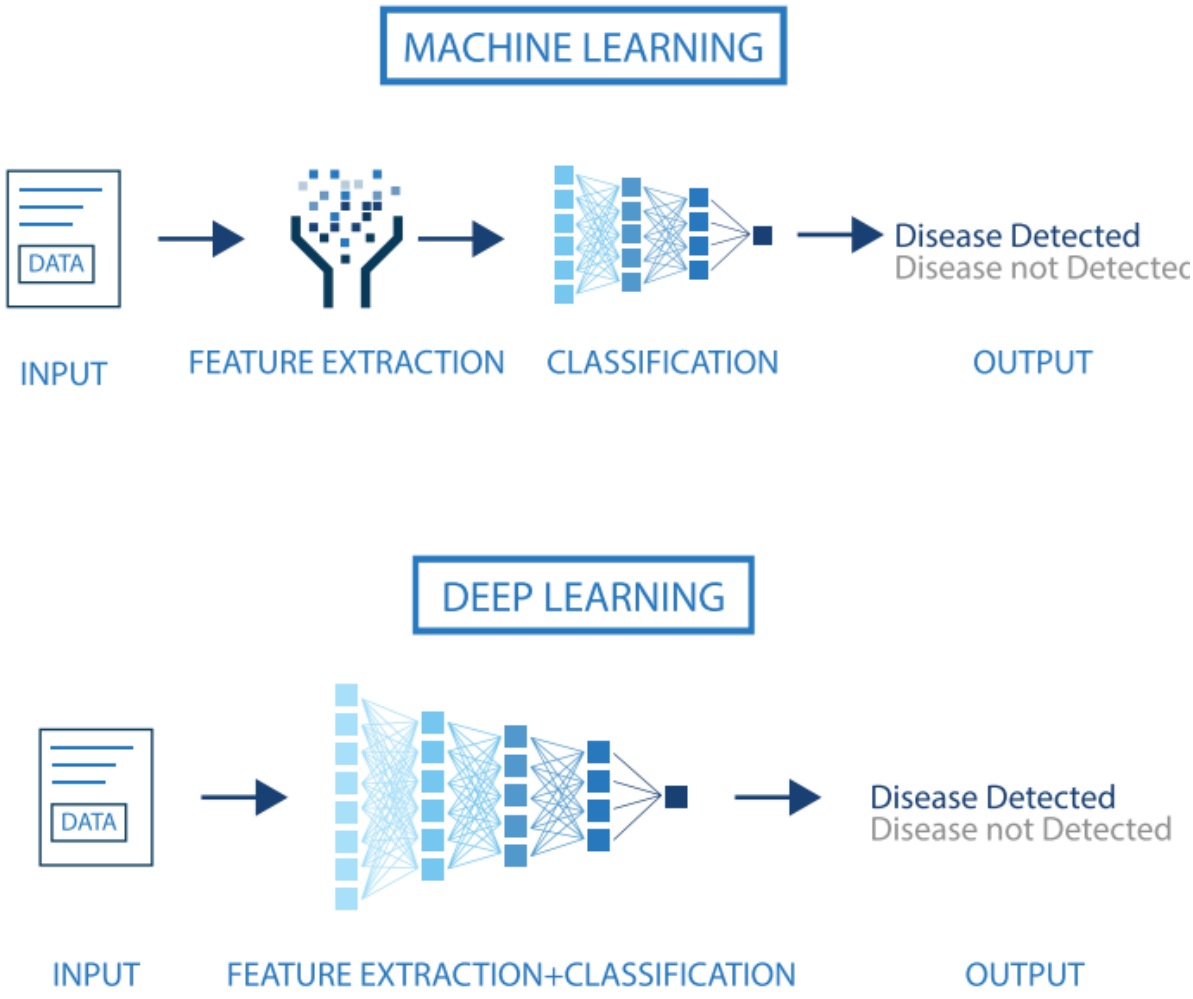
- Artificial intelligence: Ability of a computer to perform tasks commonly associated with intelligent beings.

- Machine learning: study of algorithms that learn from examples and experience instead of relying on hard-coded rules and make predictions on new data
- Deep learning: subfield of ML focusing on learning data representations as successive successive layers of increasingly meaningful representations.

Machine vs Deep Learning

- DNN: feature extraction and classification without (or with much les) human intervention.
- DNN improves with data availability, without seemingly reaching plateaus.

```
knitr::include_graphics("images/ML_vs_DL-2.png")
```



Size is important!

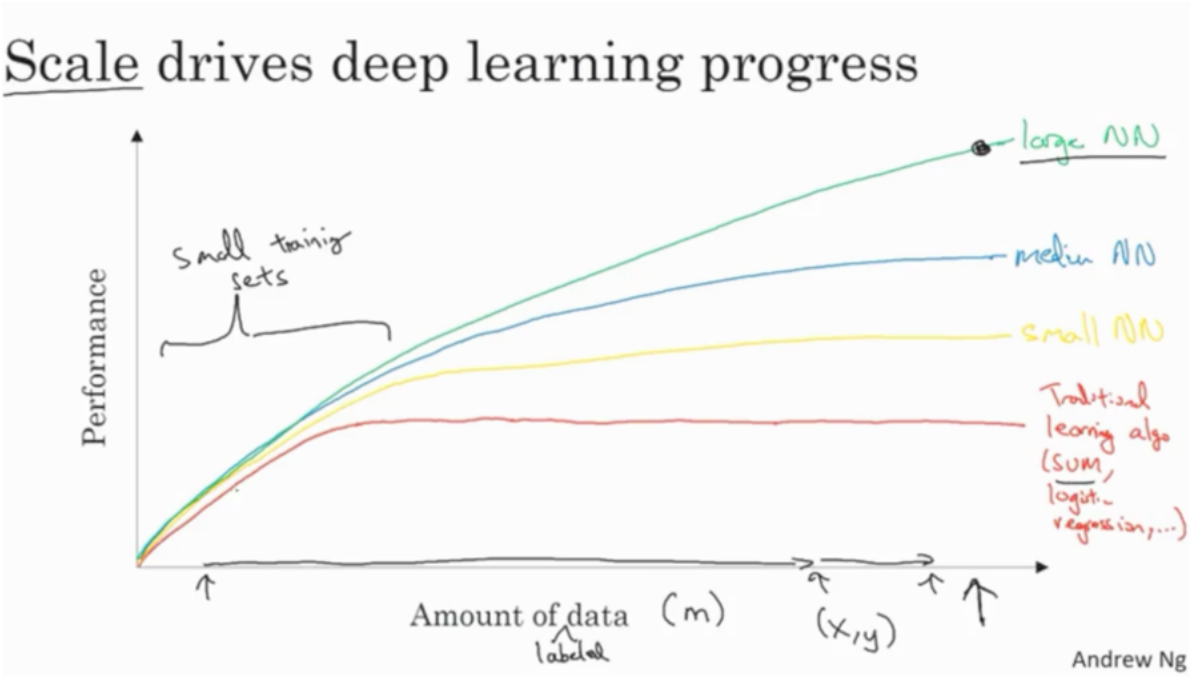


Figure 2: An illustration of the performance comparison between deep learning (DL) and other machine learning (ML) algorithms, where DL modeling from large amounts of data can increase the performance

The impact of Deep learning

- 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

Not all that glitters is gold ...

- According to F. Chollet, the developer of Keras,
  - "*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*".

Artificial Neural Networks

The perceptron, the building block

- The perceptron, was introduced in the 50's (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*?

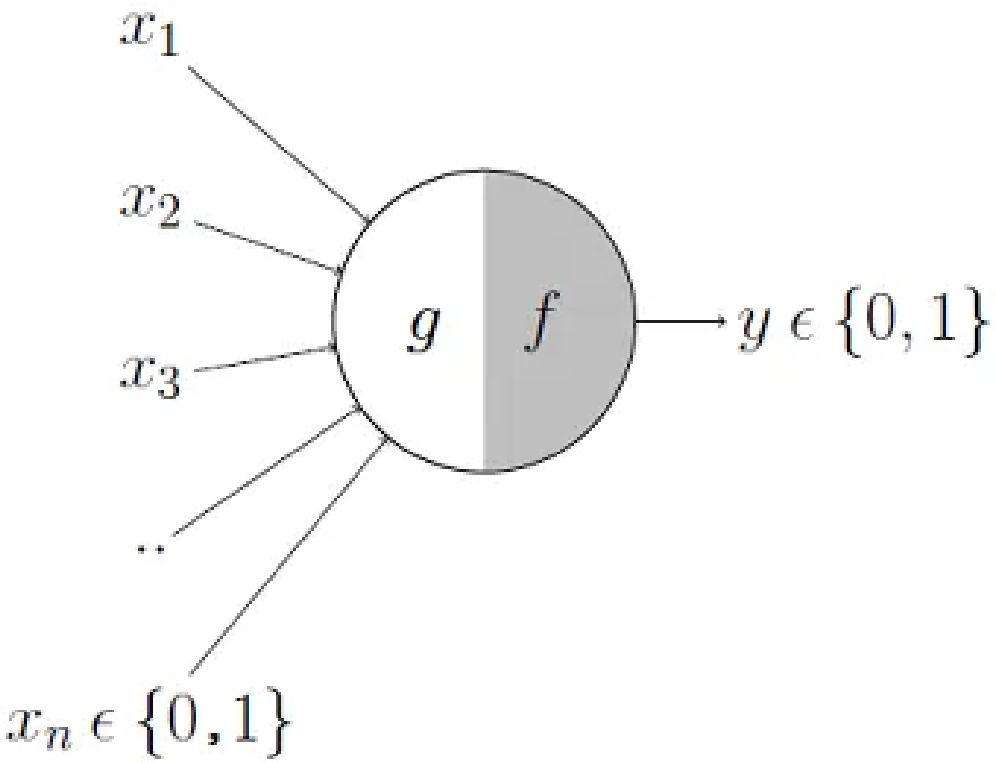
Mc Cullough's neurone

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

Mc Cullough's neurone

- It may be divided into 2 parts.
  - The first part, g,takes an input (ahem dendrite ahem),
  - It 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.


x_1
x_2
x_3
..
$x_n \in \{0,1\}$
g
f
$y \in \{0,1\}$

Limitations

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

Overcoming the limitations

- To overcome these limitations Frank Rosenblatt, 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.
- Rosenblatt's perceptron is very similar to an M-P neuron but
  - It takes a weighted sum of the inputs and
  - It sets the output as one only when the sum is more than an arbitrary threshold (***theta***).

Rosenblatt's perceptron

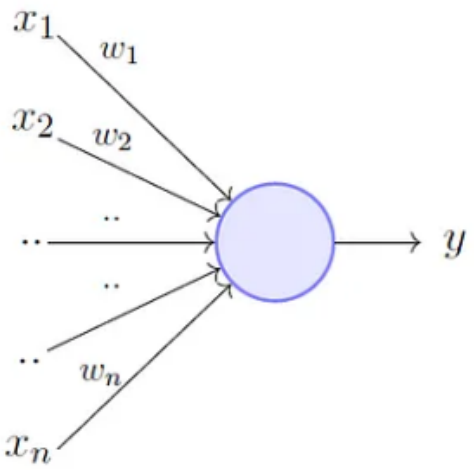
Rosenblatt's perceptron (1)

- Instead of hand coding the thresholding parameter θ,
- It is added as one of the inputs, with the weight $w_0 = -\theta$.

Comparison between the two

Comparison between the two

- This is an improvement because
  - both, weights and threshold, can be learned and
  - the inputs can be real values
- But there is still a drawback in that a single perceptron can only be used to implement linearly separable functions.



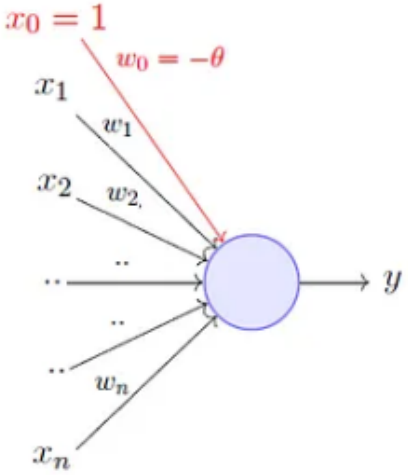
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

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

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

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

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

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

Activation in biological neurons

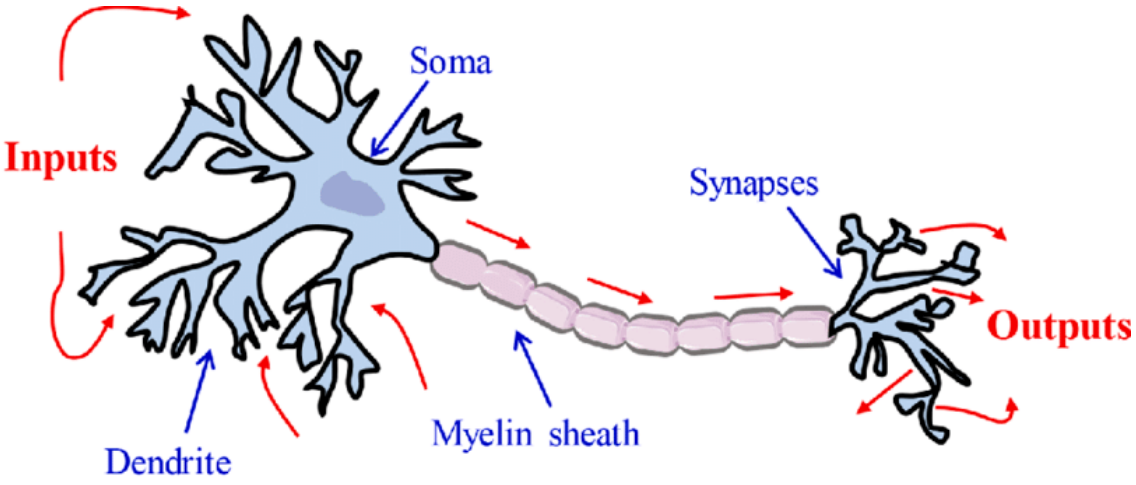
- Biological neurons are specialized cells in the CNS that transmit electrical and chemical signals to communicate with each other.
- The neuron's activation is based on the release of neurotransmitters, chemical substances that transmit signals between nerve cells.
  - When the signal reaching the neuron exceeds a certain threshold, the neuron releases neurotransmitters to continue the communication process.

Activation functions in AN

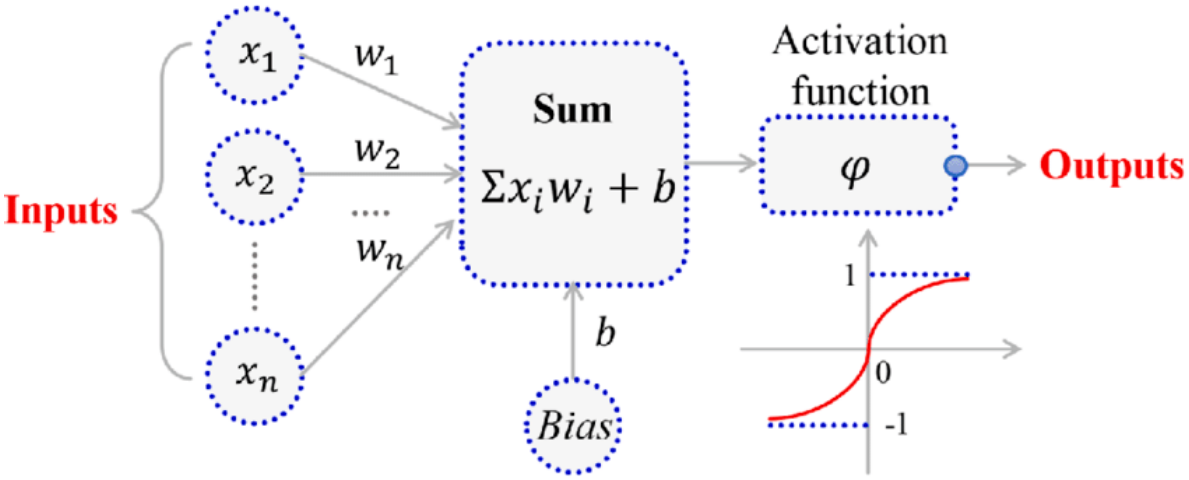
- Analogously, *activation functions* in AN are functions to decide if the AN it is activated or not.
- In AN, the activation function is a mathematical function applied to the neuron's input to produce an output.
  - In practice it extends to complicated functions that can learn complex patterns in the data.
  - Activation functions can incorporate non-linearity, improving over linear classifiers.

Activation function

```
knitr::include_graphics("images/ActivationFunction0.png")
```



(a) Biological neuron



(b) Artificial neuron

Artificial Neuron

With all these ideas 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),
- 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**.

Activation functions

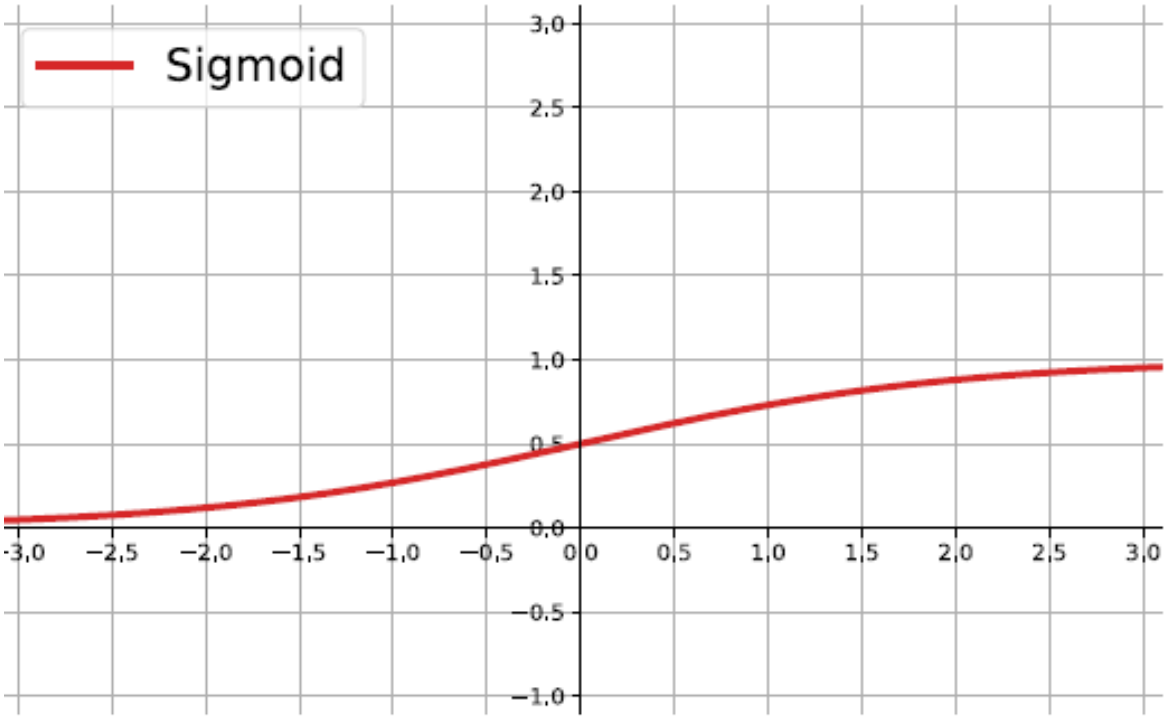
- Goal of activation function is to provide the neuron with *the capability of producing the required outputs*.
- Flexible enough to produce
  - Either linear or non-linear transformations.
  - Output in the desired range ([0,1], {-1,1}, $\mathbb{R}^+$…)
- Usually chosen from a (small) set of possibilities.
  - Sigmoid function:
  - Hyperbolic tangent, or `tanh`, function
  - ReLU

The sigmoid function

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

- Output real values $\in (0, 1)$.
- Natural interpretations as *probabilit* of an event
- *vanishing gradient* problem
- Its derivative is: $f'(z) = f(z)(1 - f(z))$.

```
knitr::include_graphics("images/SigmoidFunction.png")
```



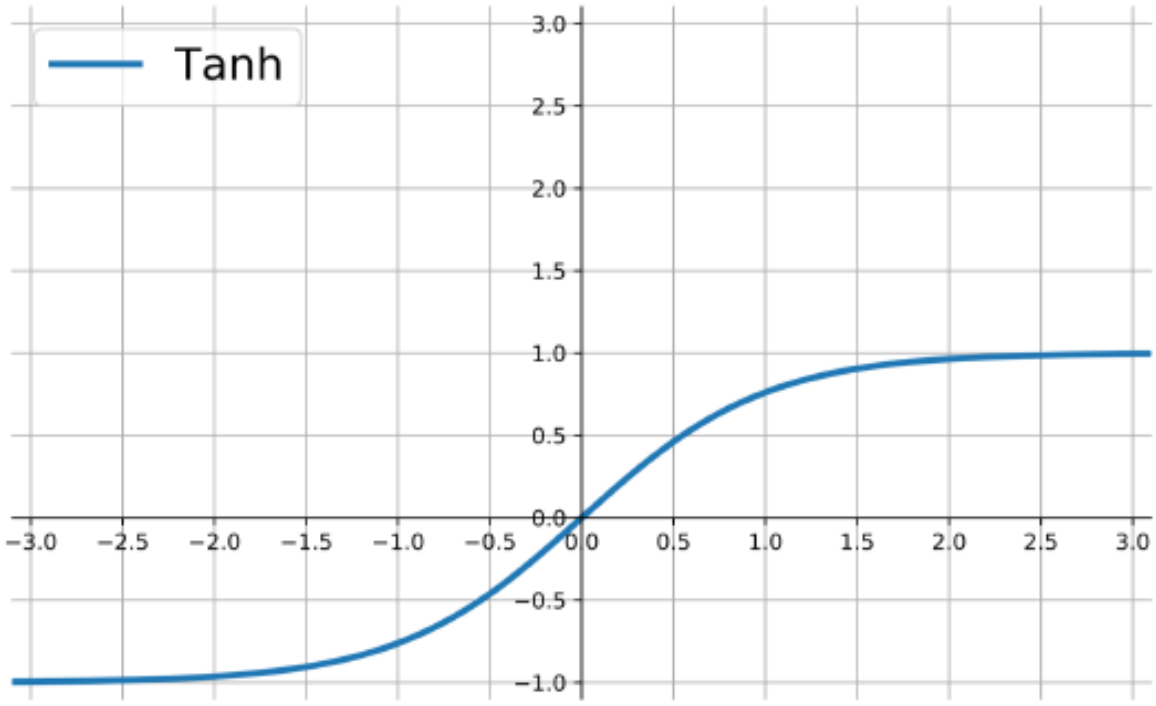
the hyperbolic tangent

Also called `tanh`, function:

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

- outputs are zero-centered and bounded in $-1{,}1$
- scaled and shifted Sigmoid
- stronger gradient but still has vanishing gradient problem
- Its derivative is $f'(z) = 1 - (f(z))^2$.

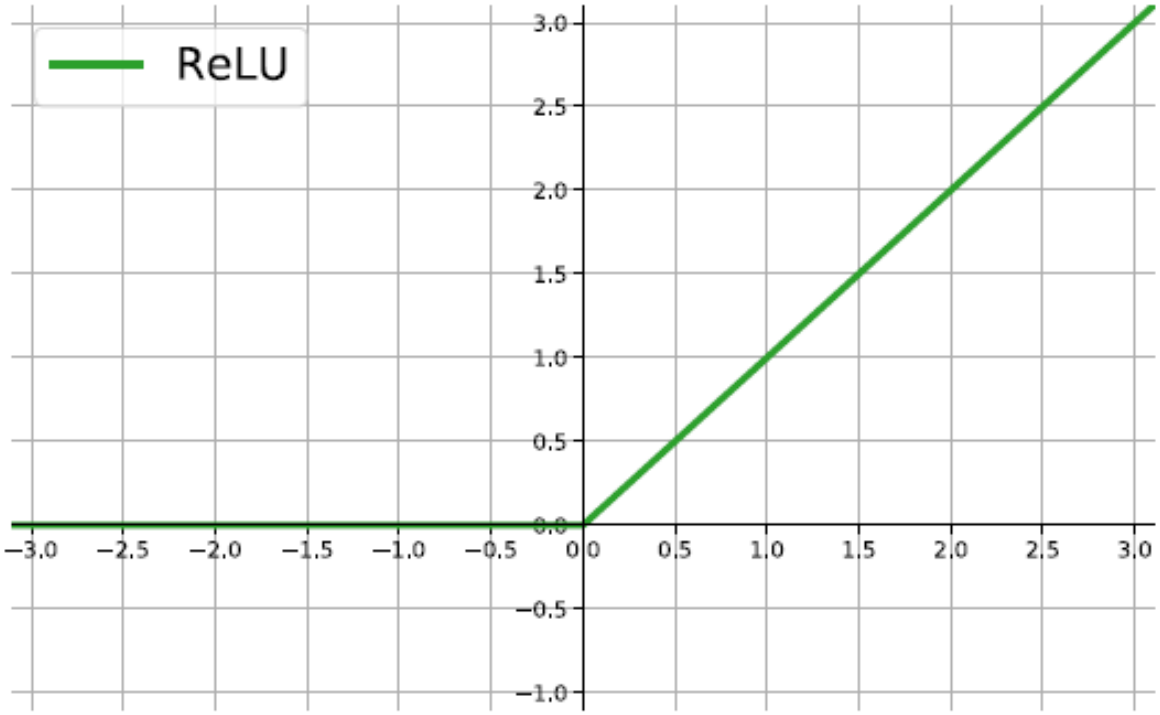
```
knitr::include_graphics("images/TanhFunction.png")
```



The ReLU

- *rectified linear unit*: $f(z) = \max\{0, z\}$.
- Close to a linear: *piecewise linear* function with two linear pieces.
- Outputs are in %(0,∞)$, thus not bounded
- Half rectified: activation threshold at 0
- No vanishing gradient problem

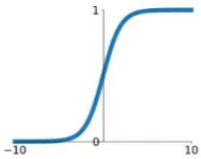
```
knitr::include_graphics("images/ReLUFunction.png")
```



More activation functions

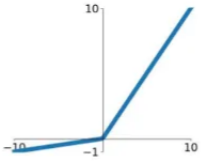
Sigmoid

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



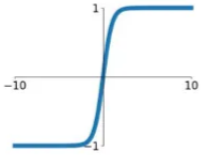
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

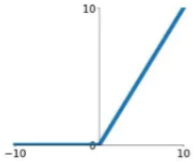


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

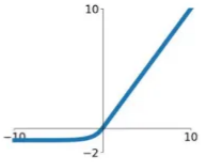
ReLU

$\max(0, x)$



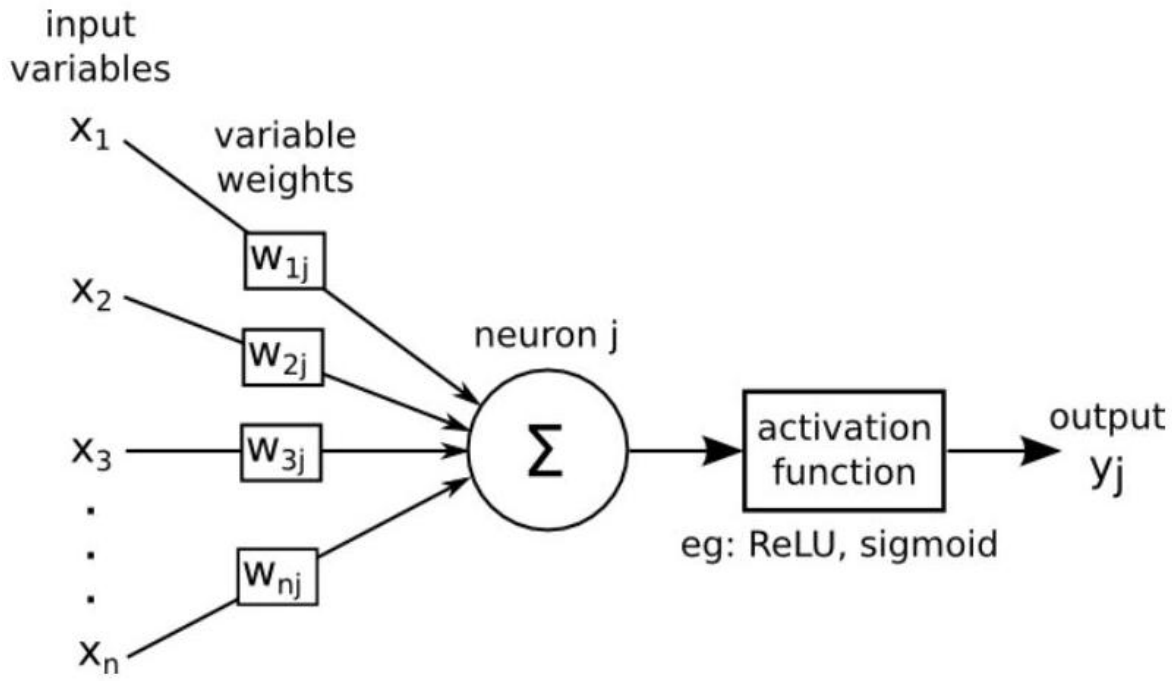
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Putting it all together

```
knitr::include_graphics("images/ArtificialNeuron.png")
```



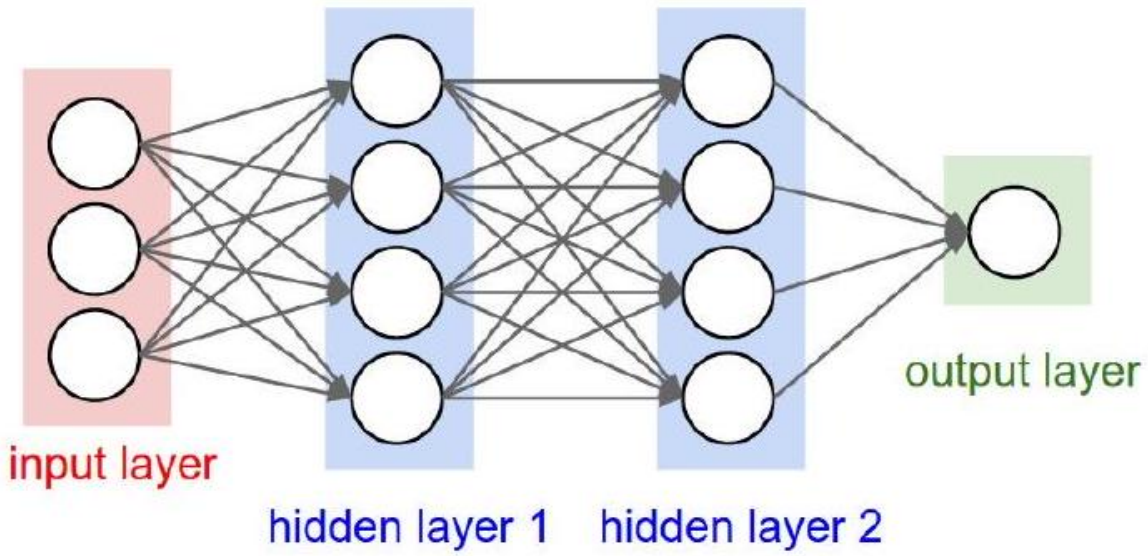
$\Sigma = \langle w_j, x \rangle + b_j$

Multilayer perceptrons

- A Multilayer Perceptron or Artificial Neural Network (ANN) is a computational model that mimics the structure and function of the human brain.
- It is composed of interconnected nodes, called neurons, that are organized into layers.
- Neurons in each layer are connected to neurons in the next layer, forming a directed graph-like structure.
- The first layer (input layer) receives input data.
- Last layer produces the final prediction
- Intermediate (hidden) layers perform intermediate calculatuions.

An Artificial Neural network

```
knitr::include_graphics("images/MultiLayer1.png")
```



The architechture of ANN

- 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

Activation functions for ANN (1)

- For the output layer, the activation function is generally different from the one used on the hidden layers.
  - For **regression**, we apply no activation function on the output layer.
  - For **binary** classification, where output is a prediction of $\mathbb{P}(Y = 1/X) \in [0, 1]$
    - The *sigmoid* activation function is used.

- For **multi-class classification**, where ther is one neuron per class giving a prediction of $\mathbb{P}(Y = i/X)$.
  - The *softmax* activation function is common

The softmax activation function

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

An example

A predictive ANN

We use the `neuralnet` package 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)
```

```
Loading required package: neuralnet
```

```
Warning: package 'neuralnet' was built under R version 4.2.3
```

Data for the example

And use the `dividendinfo.csv` dataset from https://github.com/MGCodesandStats/datasets

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

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
normData <- as.data.frame(lapply(mydata, normalize))
```

Test and training sets

Finally we break our data in a test and a training set:

```
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,]
```

Training a neural network

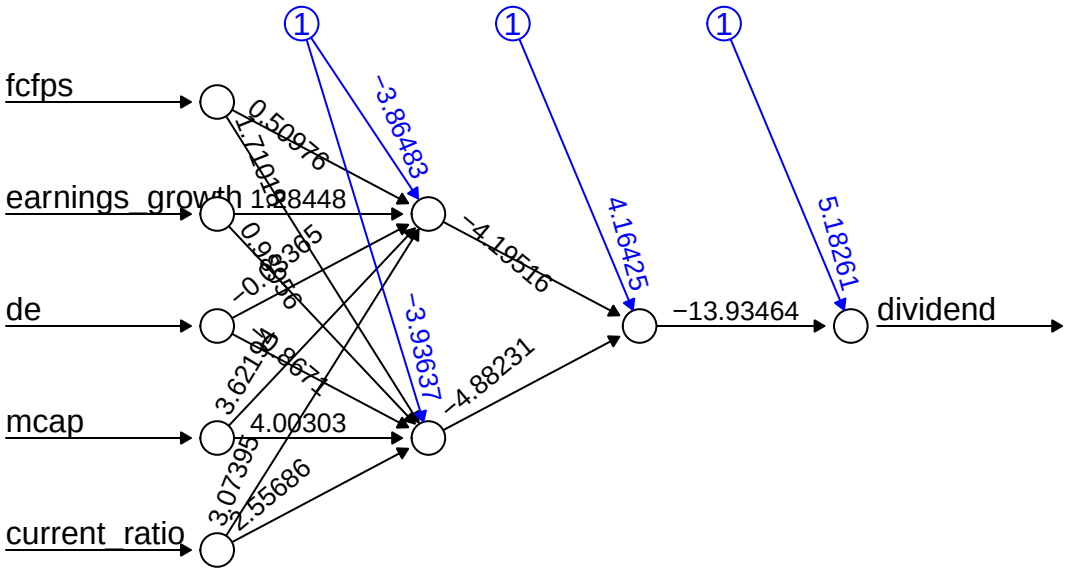
We train a simple NN with two hidden layers, with 4 and 2 neurons respectively.

```
#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)
```

Network plot

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

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



Error: 1.650086 Steps: 275

Predictions

```
temp_test <- subset(testset, select =
                      c("fcfps","earnings_growth",
                        "de", "mcap", "current_ratio"))
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
```

Model evaluation

```
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
```

Some mathematics behind ANN

Mathematical formulation

- An ANN is a predictive model whose functioning and properties can be mathematically characterized.
- In practice this means describing
  - The ANN as a (non linear) function.
  - The (optimization) process for estimating the weights.

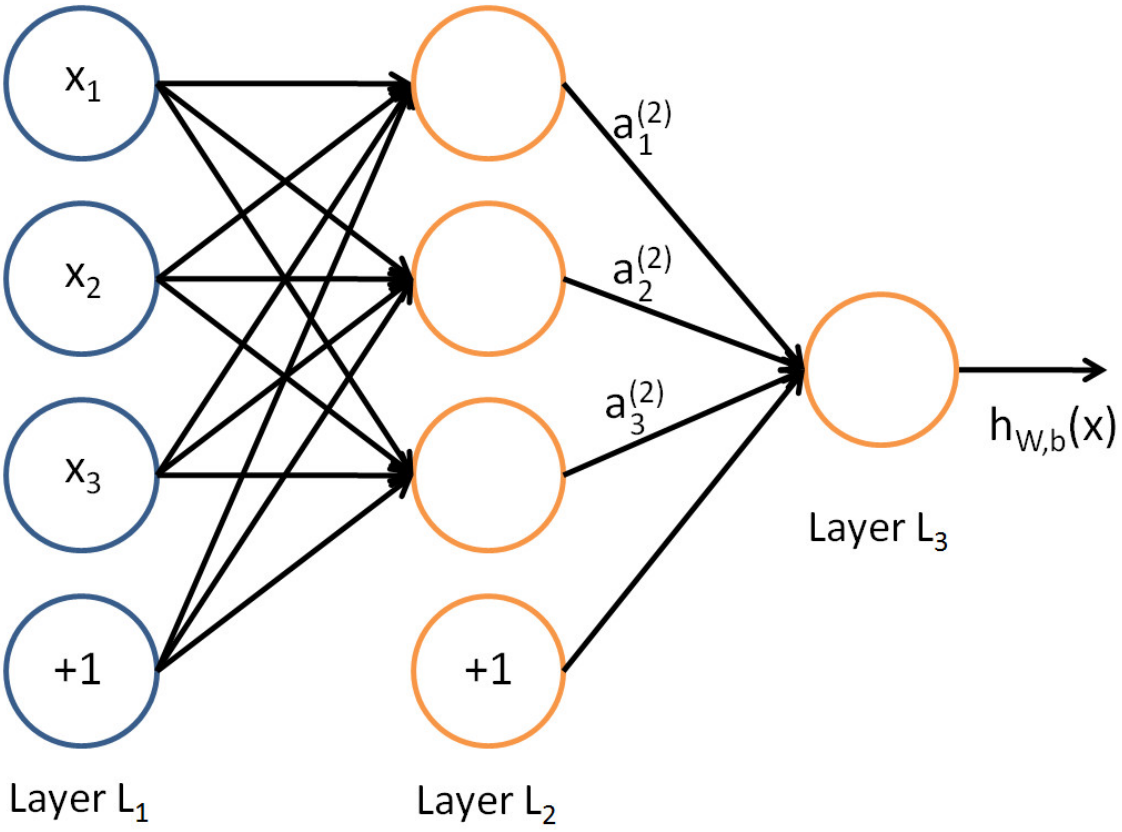
Estimating the weights

- In order to use the ANN for prediction, weights need to be estimated.
- This is done by minimizing an appropriate loss function which requires
  - An appropriate procedure: *gradient based optimization*
  - A method for computing the required derivatives: *back-propagation*.

A guiding example

- Input layer with 3 input units (plus bias unit),
- 1 hidden layer with 3 hidden units,
- Output layer with 1 output unit.

```
knitr::include_graphics("images/nn.jpg")
```



A logistic regression ANN

This ANN can be seen as a logistic regression model:

- From input layer to layer 2: non-linear transformation –> new set of complex features.
- From layer 2 to output layer use a sigmoid activation function to produce the following output from the set of *complex features*.

$$\text{The output is: } h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

Logistic regression (1)

Recall that, the logistic regression model is:

$$\log \frac{p(Y=1|x)}{1-p(Y=1|x)} = \theta^\top x$$

Isolating $p(Y=1|x)$ and taking logs in both sides, we have:

$$\frac{p(Y=1|x)}{1-p(Y=1|x)} = e^{\theta^\top x}$$

Logistic regression (2)

$$p(Y=1|x) = \frac{e^{\theta^\top x}}{1+e^{\theta^\top x}} = \frac{1}{1+e^{-\theta^\top x}}$$

- That is: *when the activation function of the output node is the sigmoid activation function, the output coincides with a logistic regression on complex features*
- And, with $h_\theta(x)$, the output of the NN, we are estimating $p(Y=1|x)$.

An ANN has weights = *parameters*

- Let n_l denote the number of layers in our network, thus $n_l = 3$ in our example.
- Label layer l as L_l, so layer L_1 is the input layer, and layer $L_{n_l} = L_3$ the output layer.
- Our neural network has parameters: $\Theta = (\Theta^{(1)}, \Theta^{(2)})$, where we will write $\theta_{ij}^{(l)}$ to denote the parameter (or weight) associated with the connection between unit j in layer l, and unit i in layer $l+1$.

Back to the example:

- Thus, in our example, we have:
  - $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$, and
  - $\Theta^{(2)} \in \mathbb{R}^{1\times 4}$.

Note that bias units don't have inputs or connections going into them, since they always output the value +1.

The ANN is defined by its weights

- We also let s_l denote the number of nodes in layer l (not counting the bias unit).
- Now, write $a_i^{(l)}$ to denote the activation (meaning output value) of unit i in layer l.
  - For $l = 1$, we also use $a_i^{(1)} = x_i$ to denote the i-th input.
- Given a fixed setting of the parameters Θ, our neural network defines a model $h_\Theta(x)$ that outputs a real number.

Combining everything to compute

- We can now see *how these weights are used to produce the output*:

$$\begin{aligned} a_1^{(2)} &= f(\theta_{10}^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3) && (1) \\ a_2^{(2)} &= f(\theta_{20}^{(1)} + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3) && (2) \\ a_3^{(2)} &= f(\theta_{30}^{(1)} + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3) && (3) \\ h_\Theta(x) &= a_1^{(3)} = f(\theta_{10}^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}) && (4) \end{aligned}$$

- Now, letting $z_i^{(l)}$ denote the total weighted sum of inputs to unit i in layer l, including the bias term

$$z_i^{(2)} = \theta_{i0}^{(1)} + \theta_{i1}^{(1)} x_1 + \theta_{i2}^{(1)} x_2 + \theta_{i3}^{(1)} x_3,$$

  the output becomes: $a_i^{(l)} = f(z_i^{(l)})$.

Compacting notation

- Note that this easily lends itself to a more compact notation.
- Extending the activation function $f(\cdot)$ to apply to vectors in an elementwise fashion:

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)],$$

then we can write the previous equations more compactly as:

$$\begin{aligned} z^{(2)} &= \Theta^{(1)} x \\ a^{(2)} &= f(z^{(2)}) \\ z^{(3)} &= \Theta^{(2)} a^{(2)} \\ h_\Theta(x) &= a^{(3)} = f(z^{(3)}) \end{aligned}$$

Compacting notation (II)

- More generally, recalling that we also use $a^{(1)} = x$ to also denote the values from the input layer,
- then given layer l's activations $a^{(l)}$, we can compute layer $l+1$'s activations $a^{(l+1)}$ as:

$$z^{(l+1)} = \Theta^{(l)} a^{(l)} \tag{5}$$
$$a^{(l+1)} = f(z^{(l+1)}) \tag{6}$$

Matricial representation (I)

$$z^{(l+1)} = \begin{bmatrix} z_1^{(l+1)} \\ z_2^{(l+1)} \\ \vdots \\ z_{s_{l+1}}^{(l)} \end{bmatrix} = \begin{bmatrix} \theta_{10}^{(l)} & \theta_{11}^{(l)} & \theta_{12}^{(l)} & \dots & \theta_{1s_l}^{(l)} \\ \theta_{20}^{(l)} & \theta_{21}^{(l)} & \theta_{22}^{(l)} & \dots & \theta_{2s_l}^{(l)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \theta_{s_{l+1}0}^{(l)} & \theta_{s_{l+1}1}^{(l)} & \theta_{s_{l+1}2}^{(l)} & \dots & \theta_{s_{l+1}s_l}^{(l)} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ a_1^{(l)} \\ a_2^{(l)} \\ \vdots \\ a_{s_l}^{(l)} \end{bmatrix}$$

Matricial representation (II)

The activation is then:

$$a^{(l+1)} = \begin{bmatrix} a_1^{(l+1)} \\ a_2^{(l+1)} \\ \vdots \\ a_{s_{l+1}}^{(l)} \end{bmatrix} = f(z^{(l+1)}) = \begin{bmatrix} f(z_1^{(l+1)}) \\ f(z_2^{(l+1)}) \\ \vdots \\ f(z_{s_{l+1}}^{(l)}) \end{bmatrix}$$

References and Resources