Mondragon Unibertsitatea
Faculty of Engineering

Advanced Machine Learning

Lecture 1: Deep Learning basics

Artificial Neural Networks

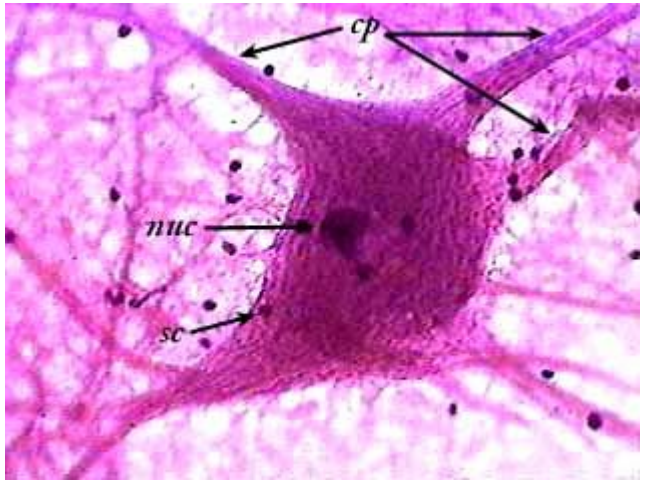
Syllabus

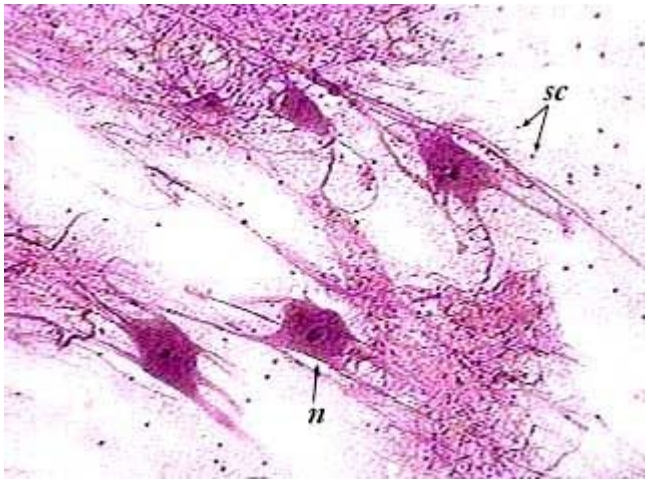
Mondragon Unibertsitatea
Faculty of Engineering

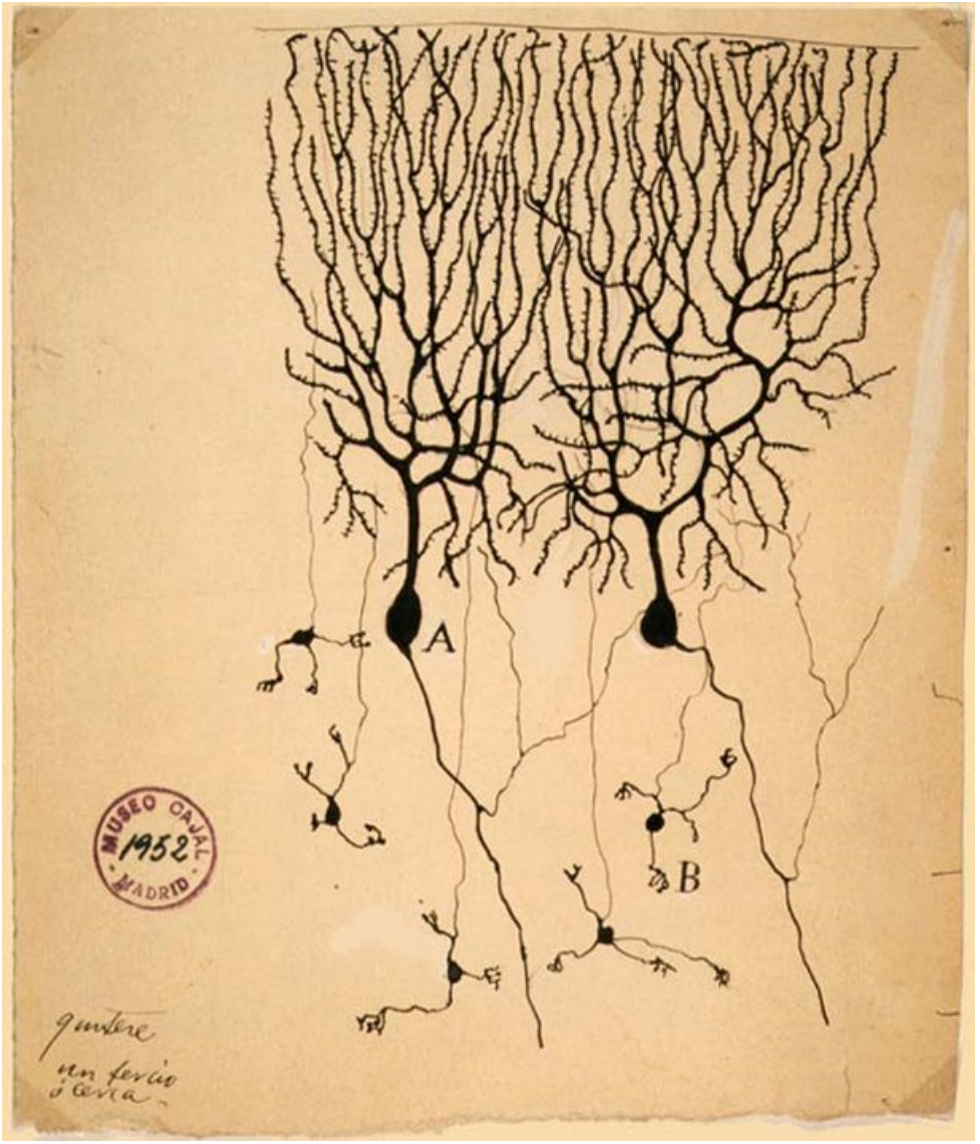
- Parts of this subsection:
  - The neuron
  - Activation Functions
  - How does a NN work (intuition)
  - Loss Functions
  - BackPropagation
  - Optimization - Gradient Descent

Neuron (or Perceptron)

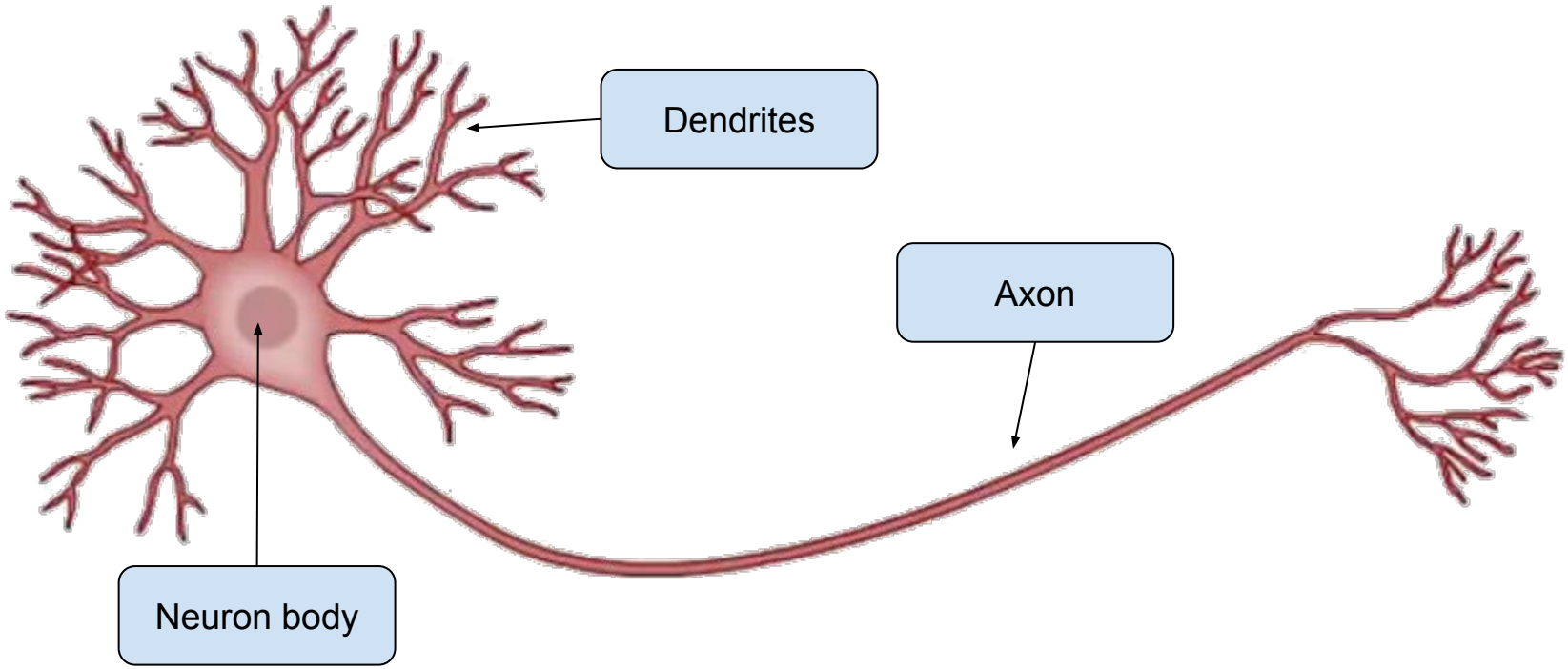
Neuron



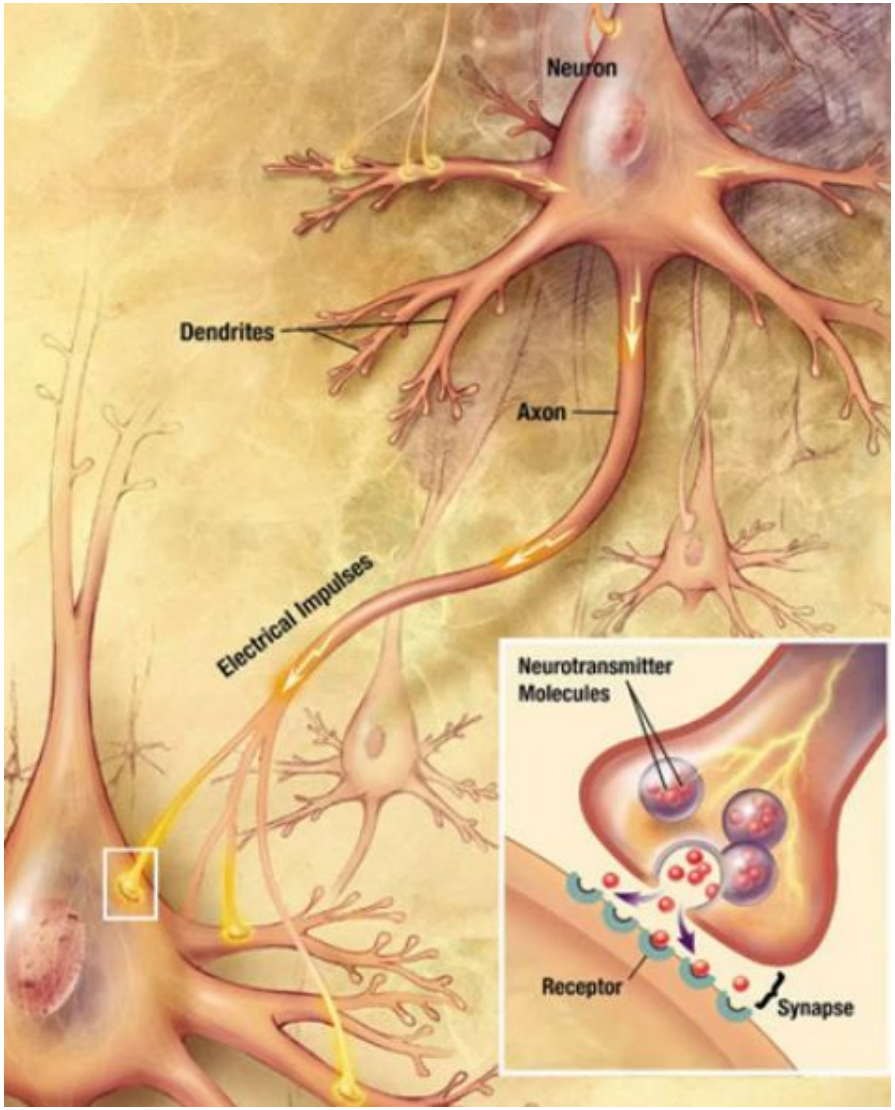




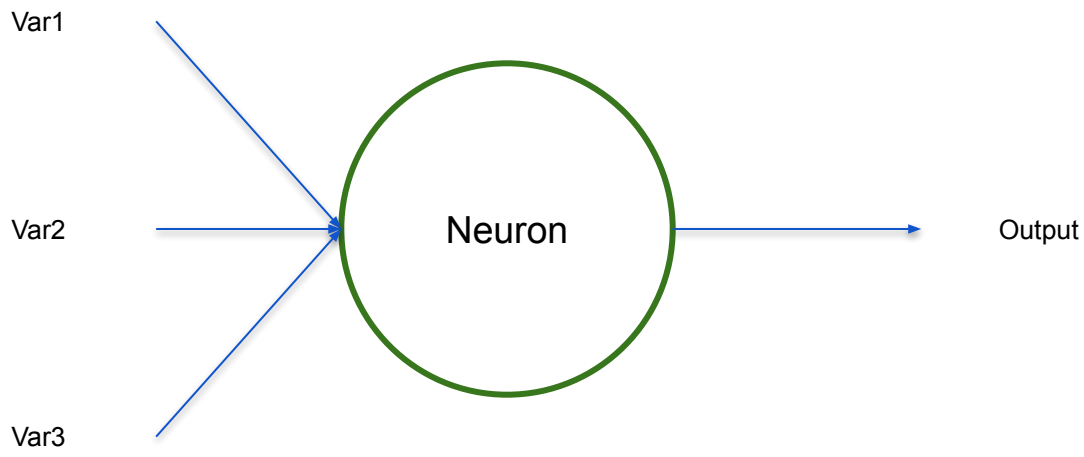
Neuron



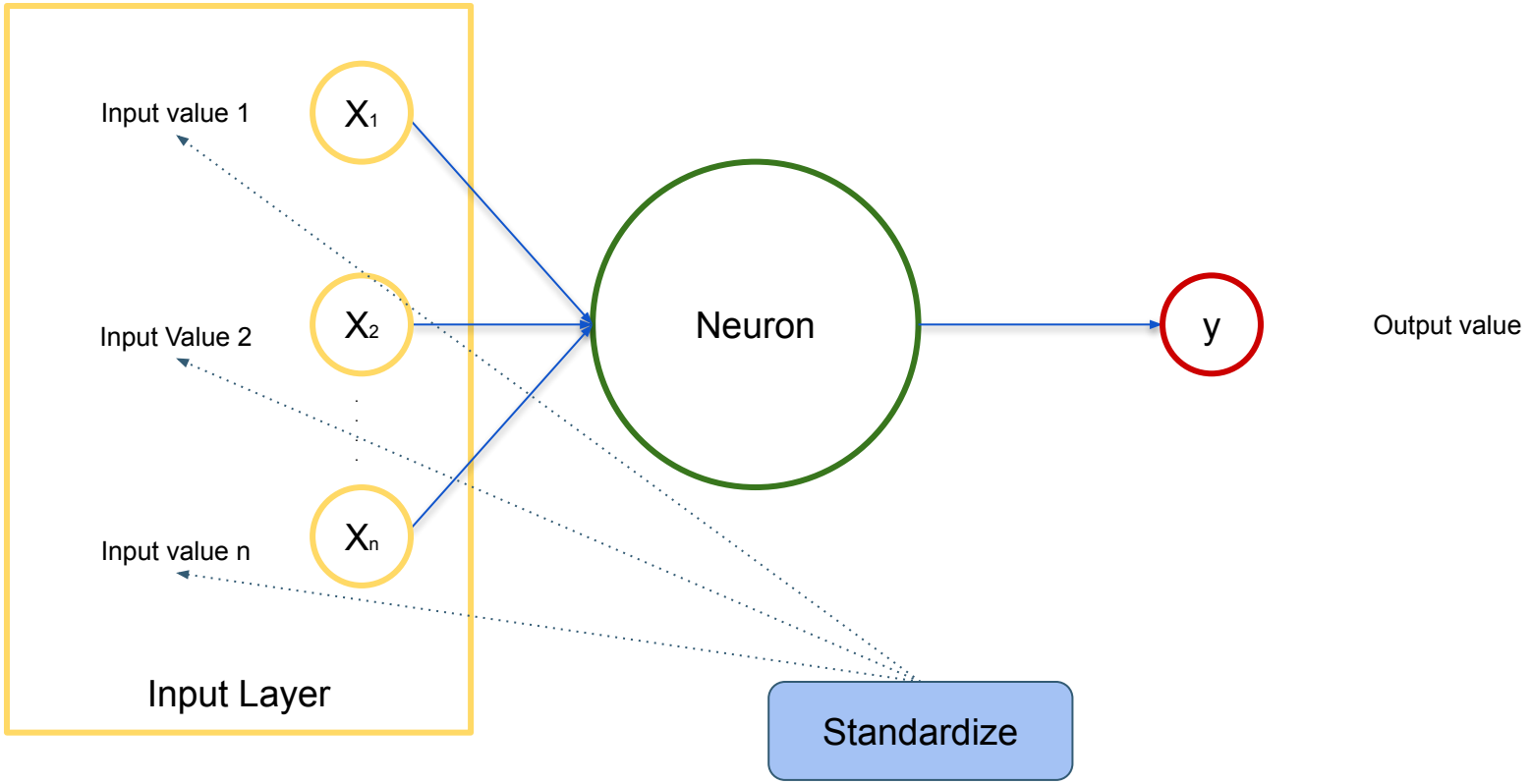
Neuron


Neuron
Dendrites
Axon
Electrical Impulses
Neurotransmitter Molecules
Receptor
Synapse

Neuron

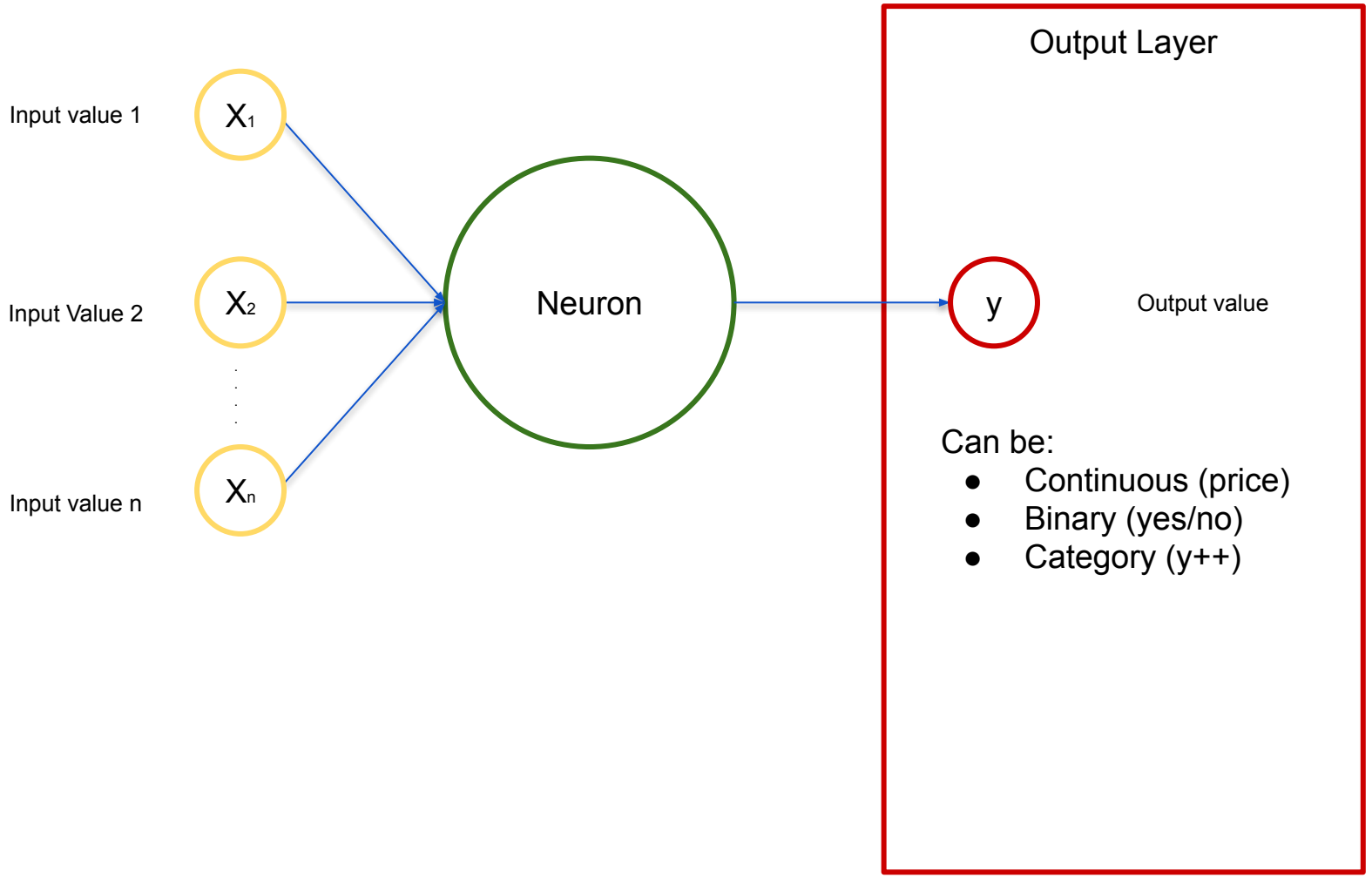


Neuron - Input Layer

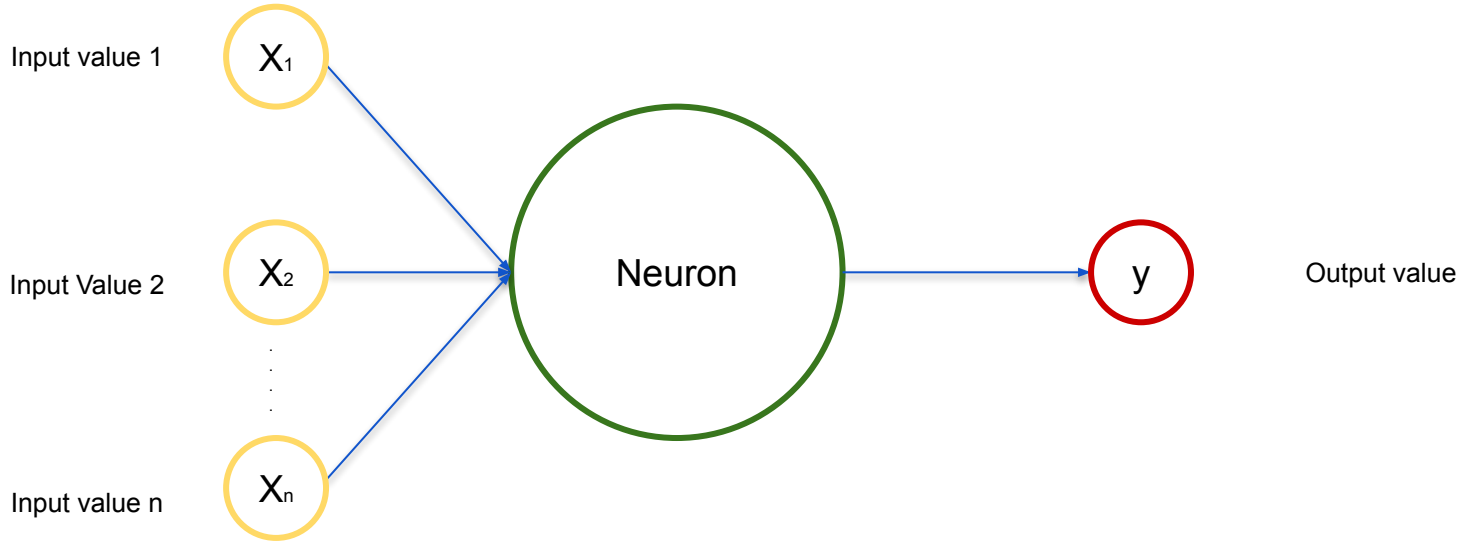


Person	€ in bank	Age	work	salary	shampoo
1	15000	22	NO	NO	Eroski
2	250000	62	arquitect	120000	H&S

Neuron - Output Layer



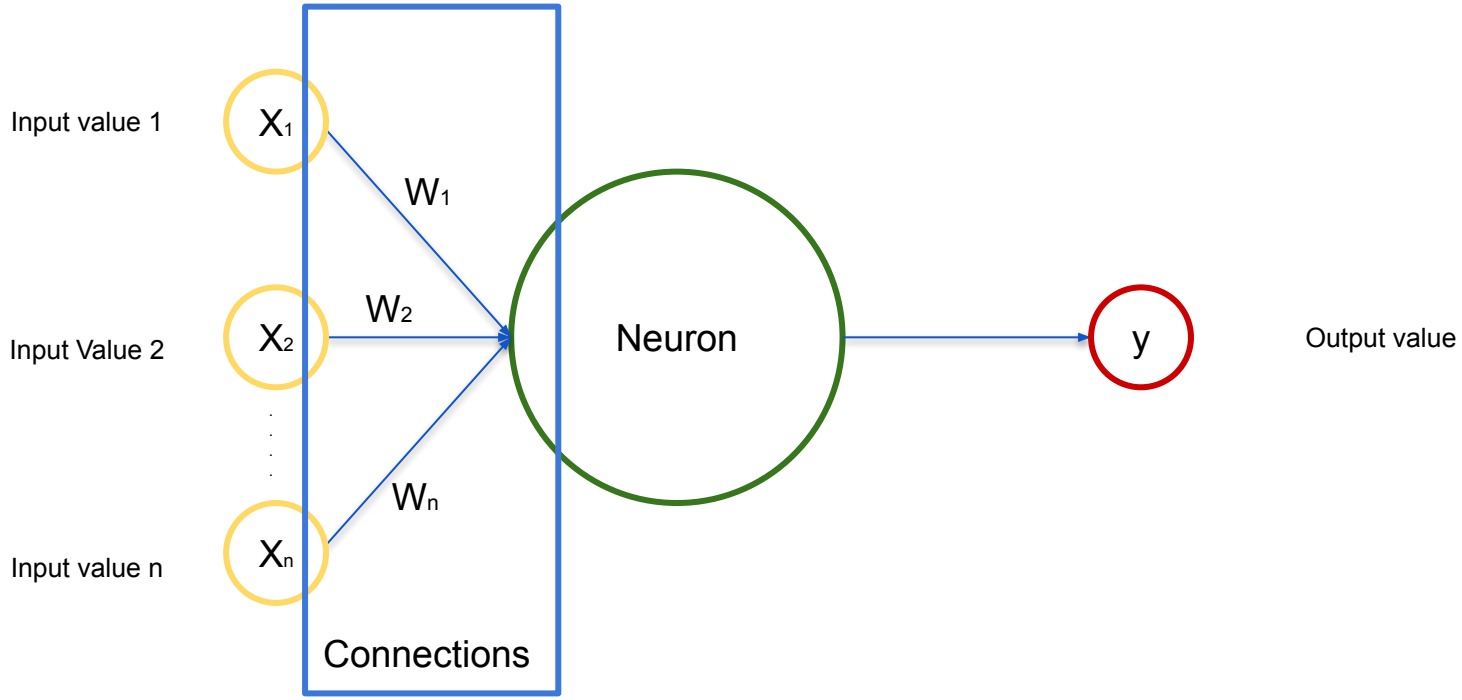
Neuron - Input Output Relation



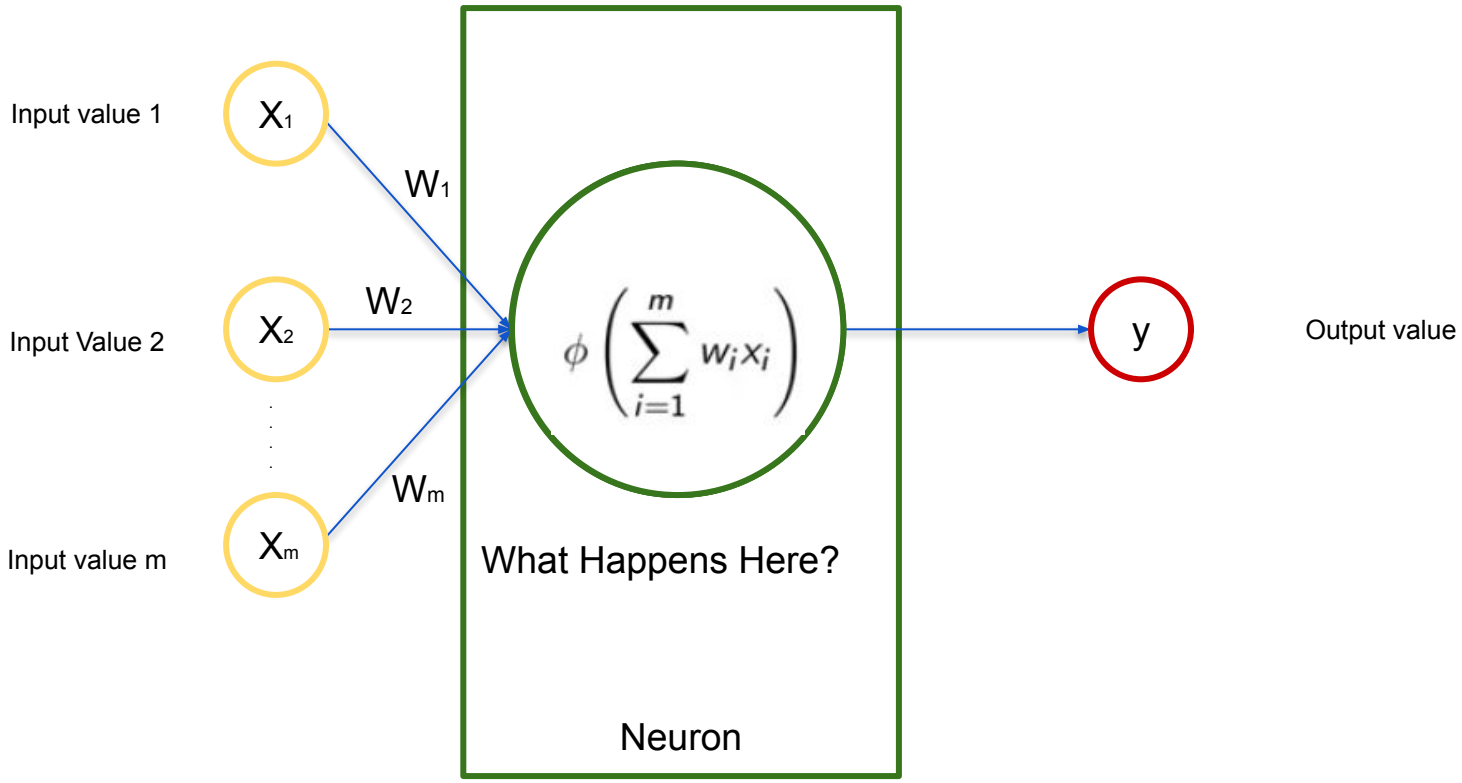
Person	€ in bank	Age	work	salary	shampoo
2	250000	62	Arqu.	120000	H&S

Can we give him a mortgage?
1

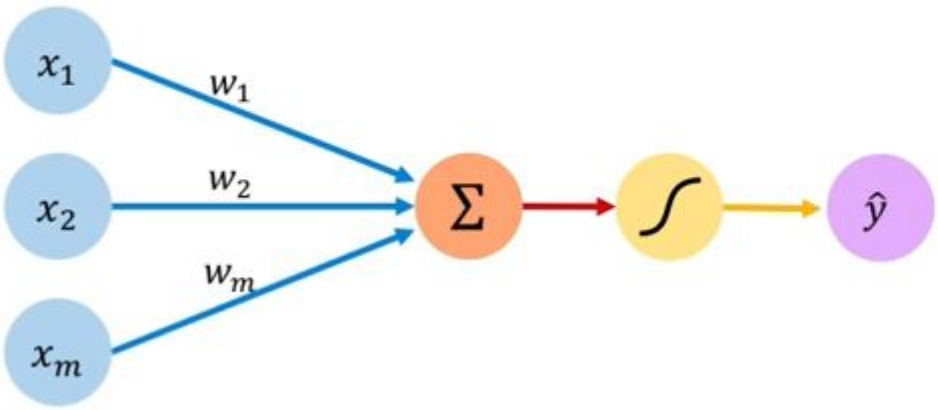
Neurona - Weights and Connections

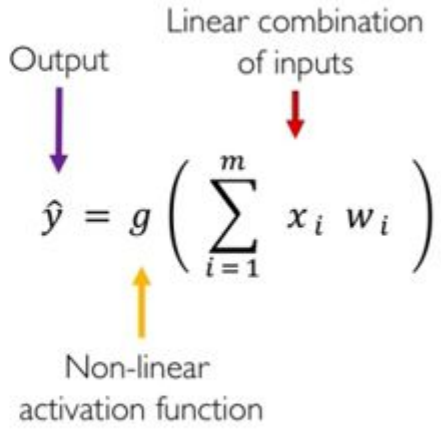


Neurona - Nucleus

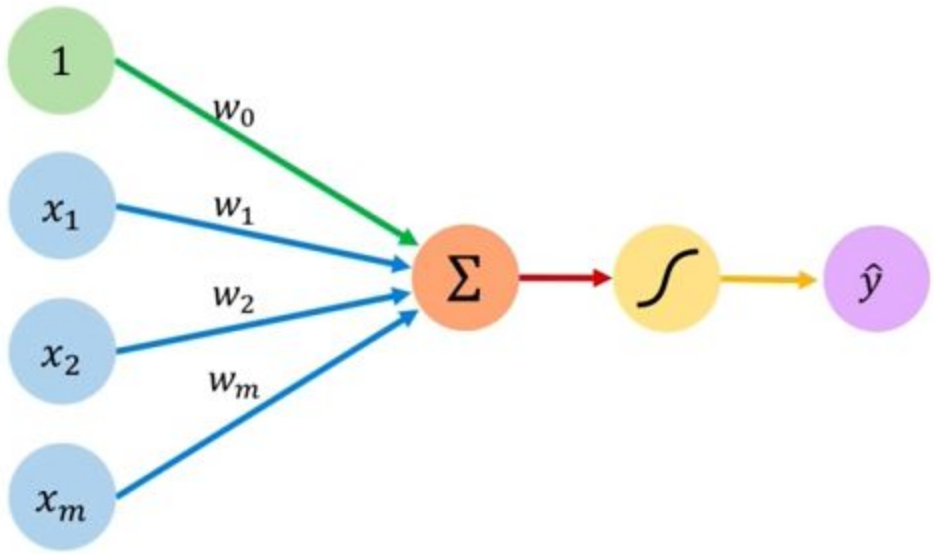


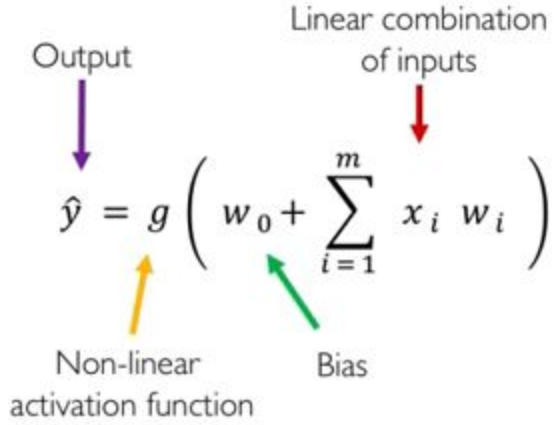
Neuron


x_1
x_2
x_m
w_1
w_2
w_m
Σ
$\int$
$\hat{y}$

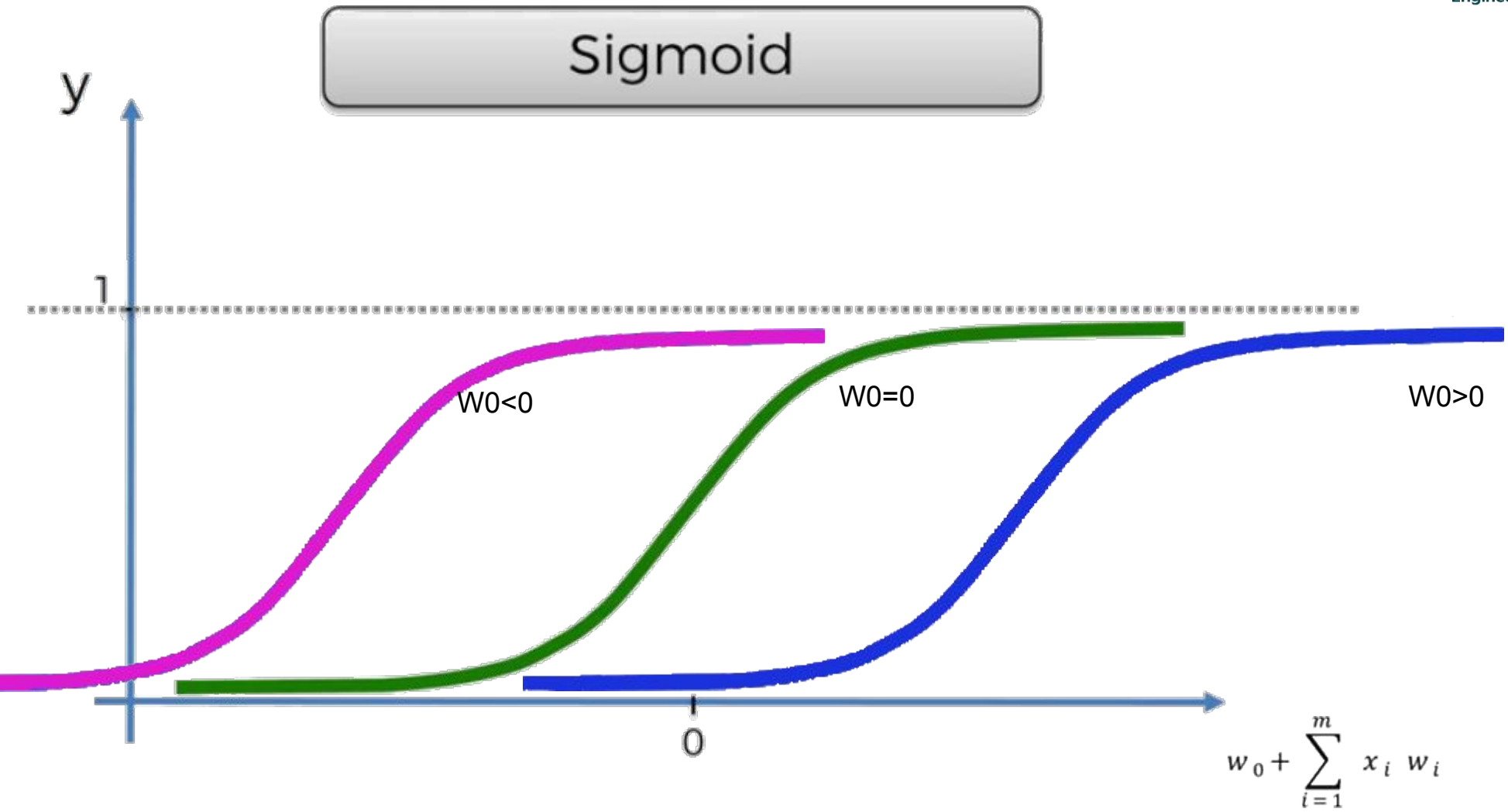

Output
Linear combination of inputs
$\hat{y} = g\left(\sum_{i=1}^{m} x_i \; w_i\right)$
Non-linear activation function

Neuron


1
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
$\int$
$\hat{y}$

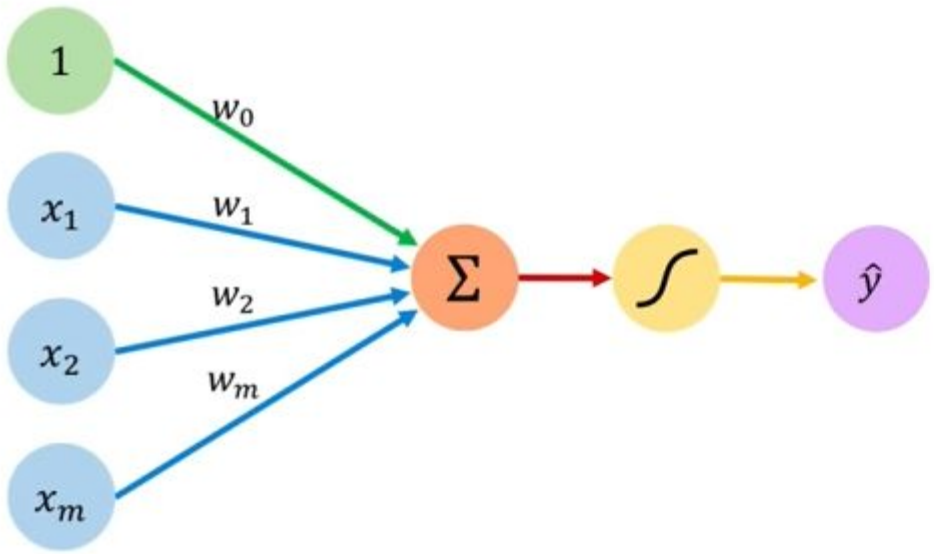

Output
Linear combination of inputs
$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$
Non-linear activation function
Bias

biass in sigmoid



source: superdatascience.com

Neuron



Inputs Weights Sum Non-Linearity Output

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

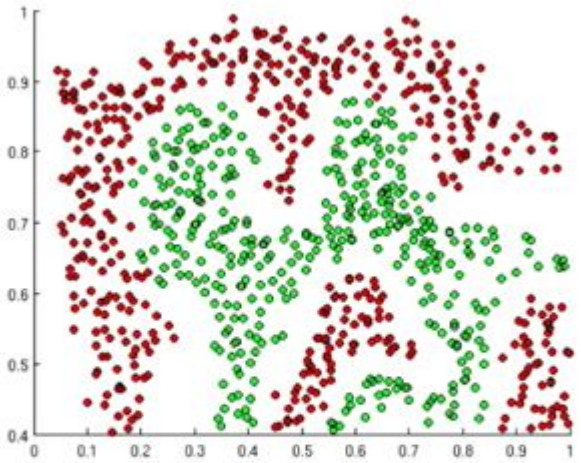
where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Activation Functions

Importance of Activation Functions

Mondragon Unibertsitatea
Faculty of Engineering

The purpose of Activation Functions is to **induce non-linearities** into the network

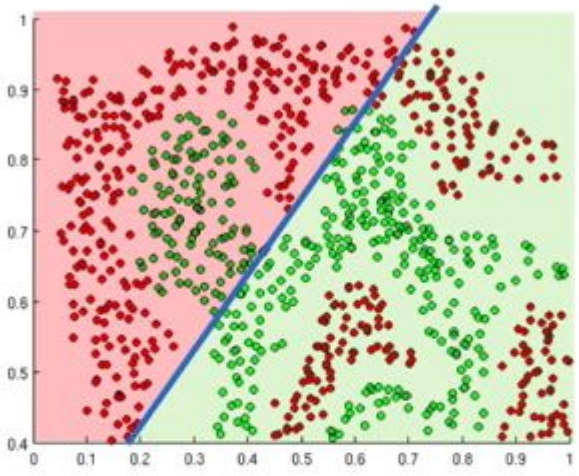


What if we wanted to build a neural network to distinguish green vs red points?

Importance of Activation Functions

Mondragon Unibertsitatea
Faculty of Engineering

The purpose of Activation Functions is to **induce non-linearities** into the network (in the real world almost all data is non-linear)

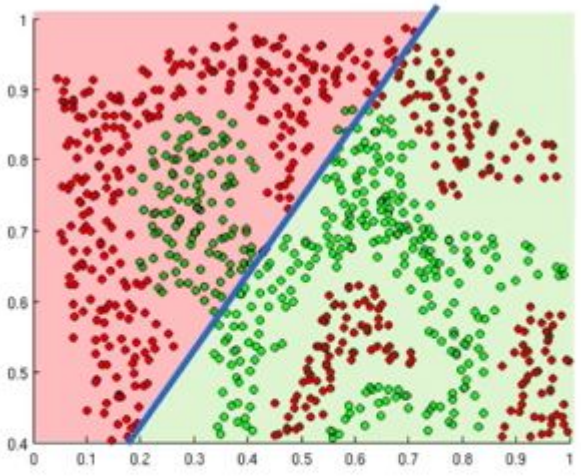


Linear activation functions produce linear decisions no matter the network size

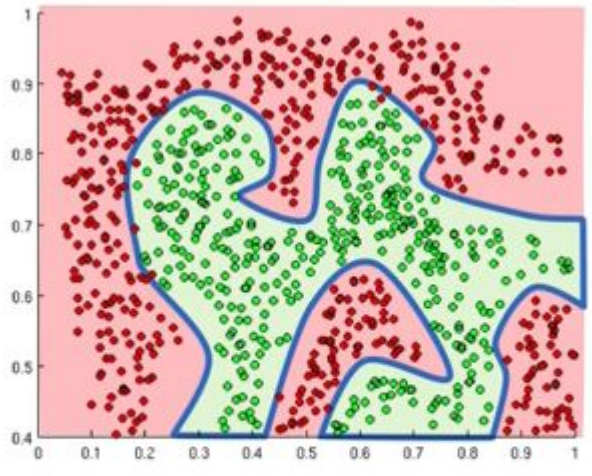
Importance of Activation Functions

Mondragon Unibertsitatea
Faculty of Engineering

The purpose of Activation Functions is to **induce non-linearities** into the network (in the real world almost all data is non-linear)

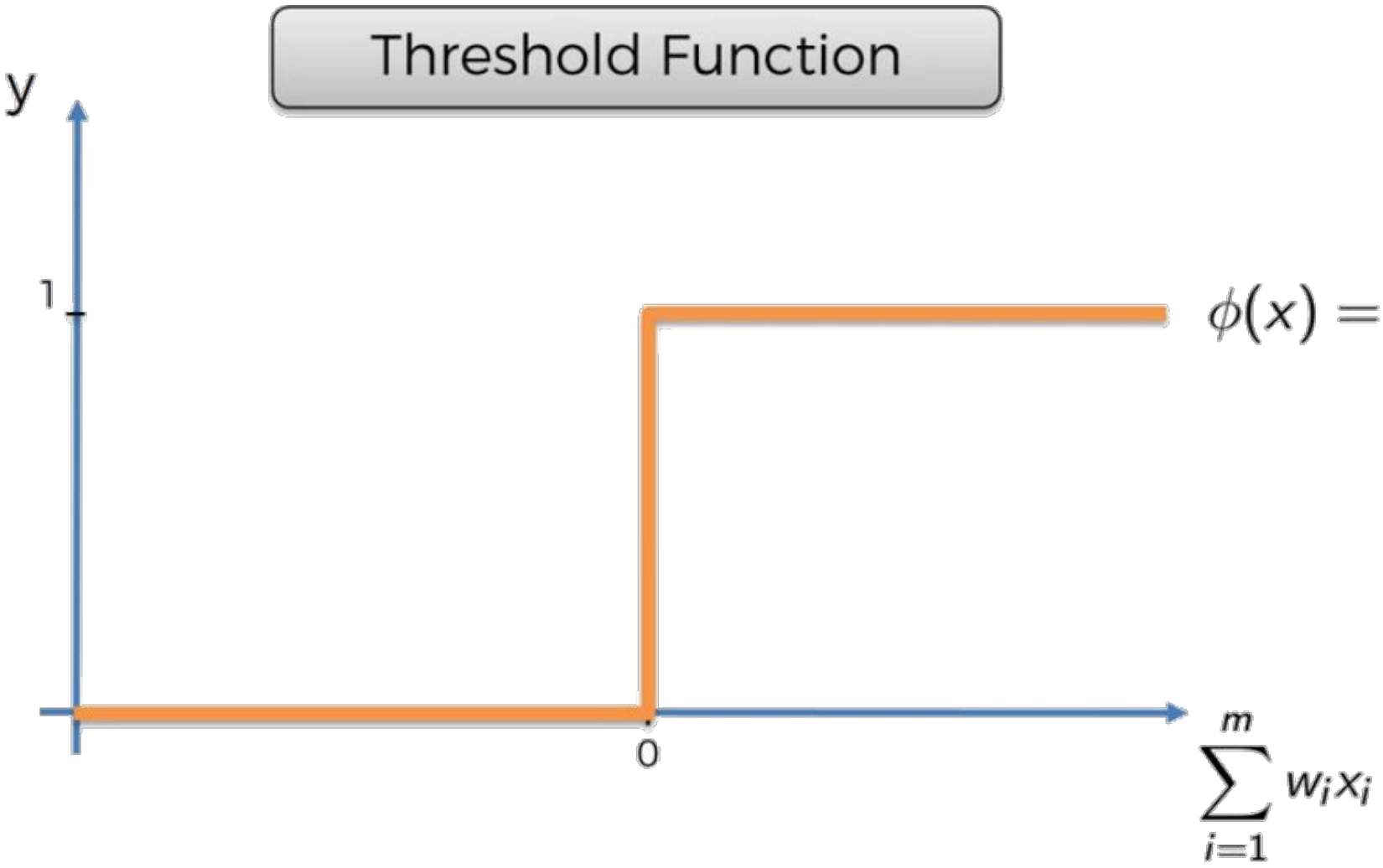


Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

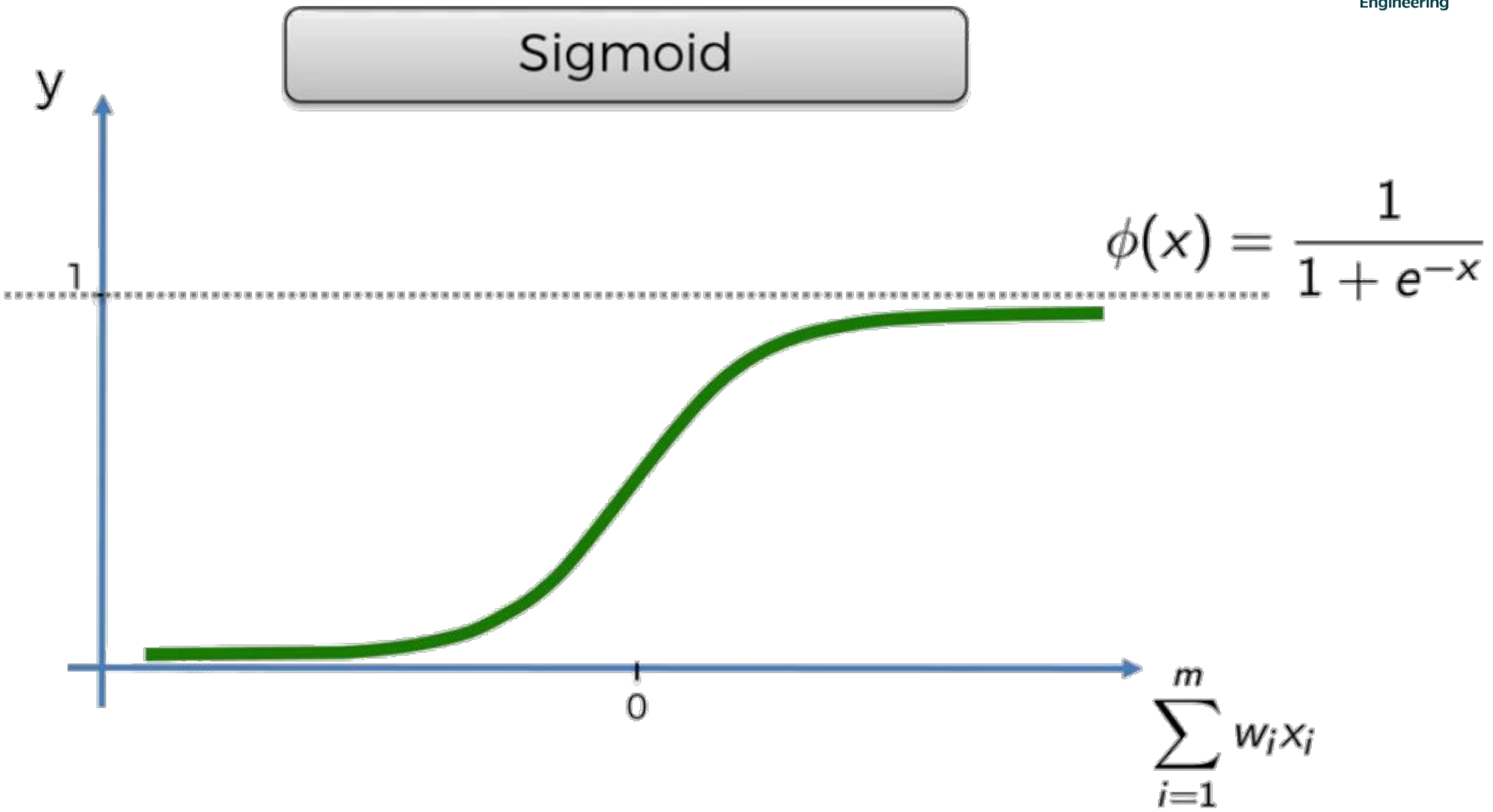
Threshold Function



$$\phi(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

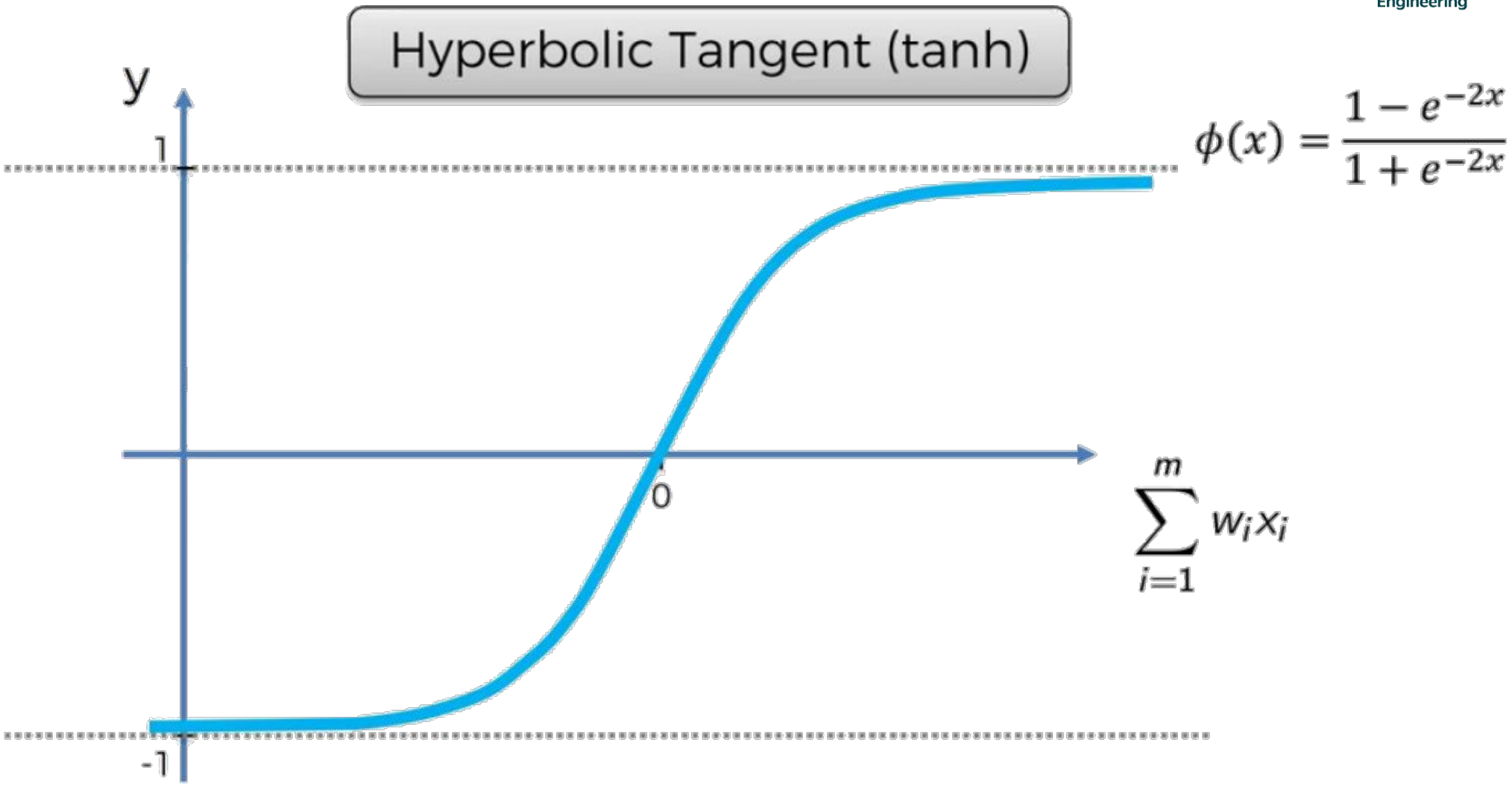
source: superdatascience.com

Sigmoid



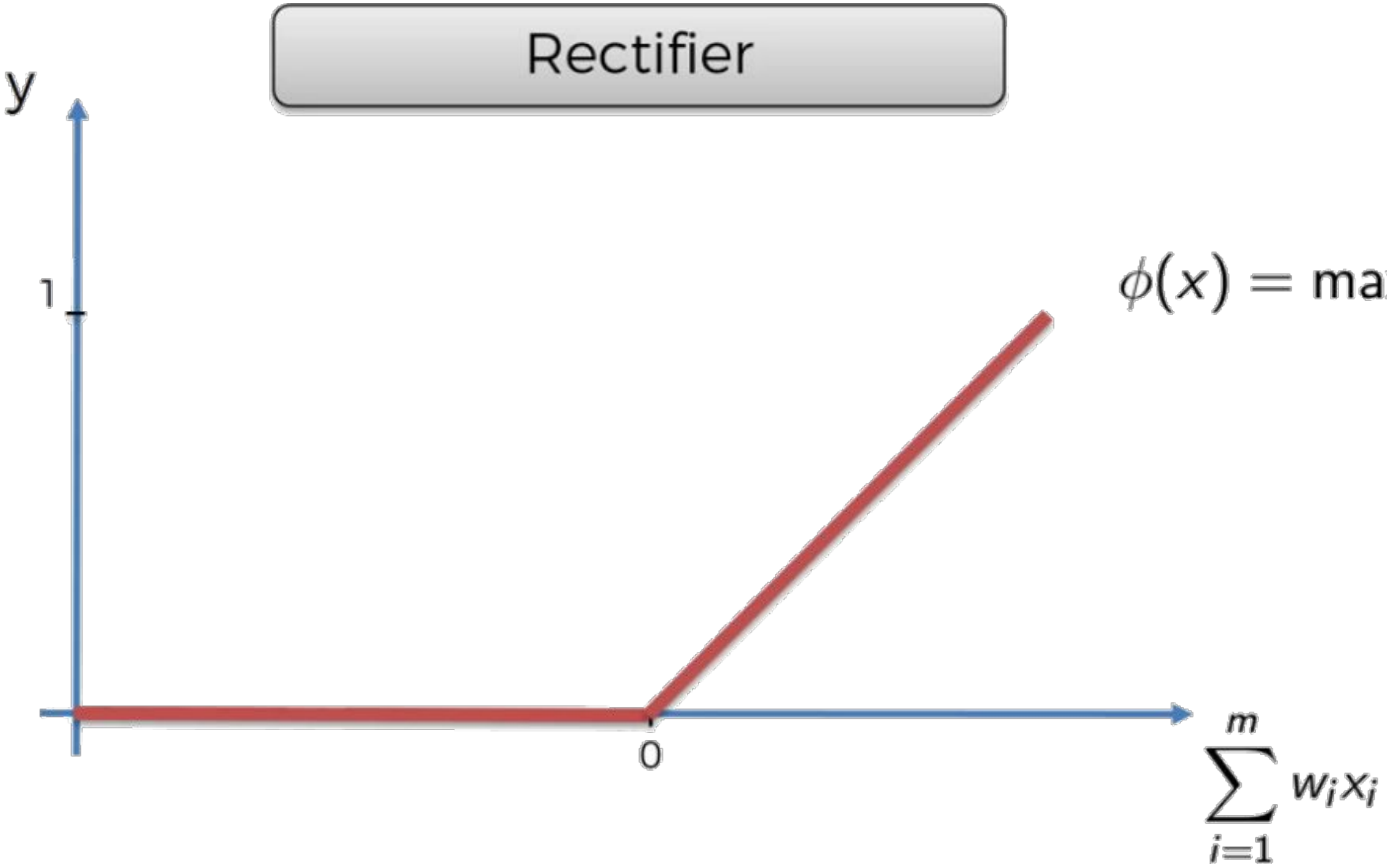
source: superdatascience.com

Hyperbolic Tangent (tanh)



source: superdatascience.com

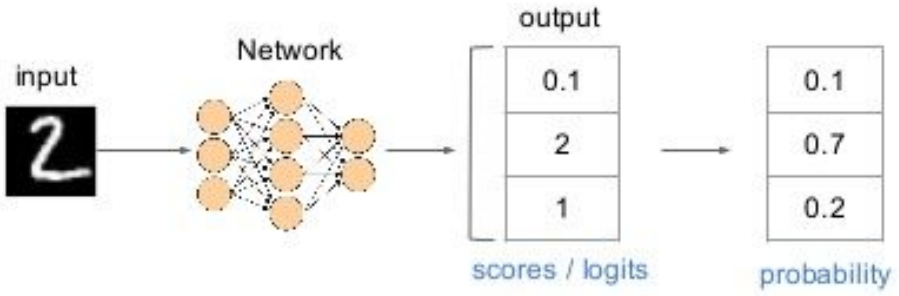
Rectifier (ReLU)



source: superdatascience.com

SOFTMAX

- Convert scores into confidences/probabilities
  - From 0.0 to 1.0
  - Probability for all classes adds to 1.0



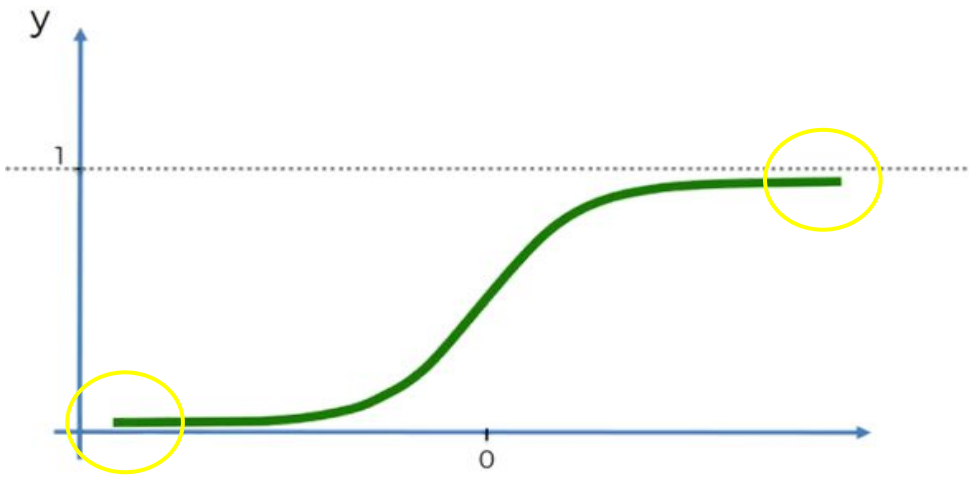
28

source: superdatascience.com

Why ReLu?

Mondragon Unibertsitatea
Faculty of Engineering

- The biggest advantage of ReLu is the non saturation of its gradient
  - This, accelerates the convergence of stochastic gradient descent
  - In those locations, the gradient/derivative value is very small. Consequently, after numerous iterations the weights get updated so slowly because the value of gradient is so much small. This is why we use ReLU which its gradient does not have this problem.
- Tanh / Sigmoid neurons involve expensive operations (exponentials, etc.), the ReLU can be implemented simply
- ReLu combined with modern techniques (Xavier initialization, dropout and (later) batchnorm) give great results.



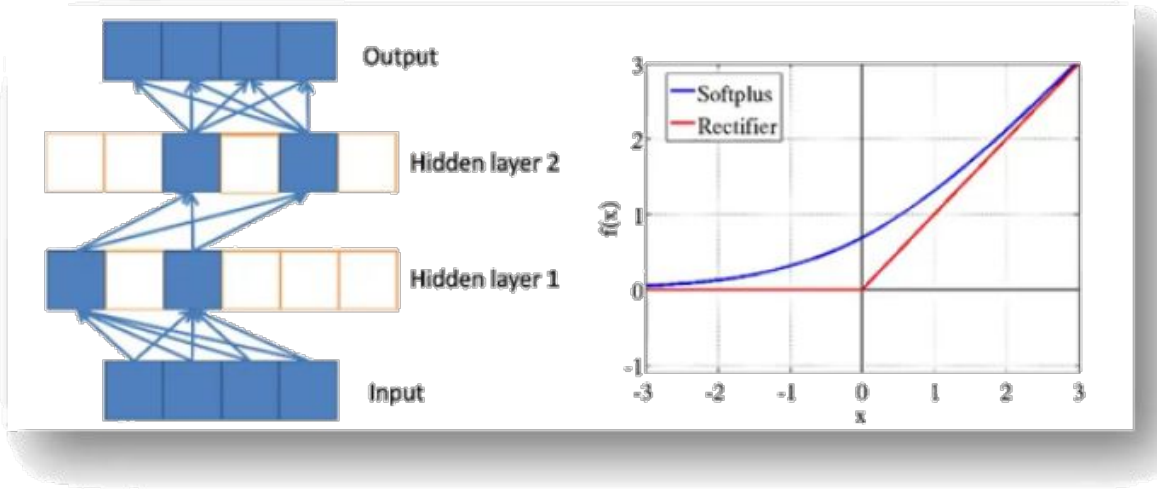
Activation Functions

Mondragon Unibertsitatea
Faculty of Engineering

Additional Reading:

Deep sparse rectifier neural networks

By Xavier Glorot et al. (2011)



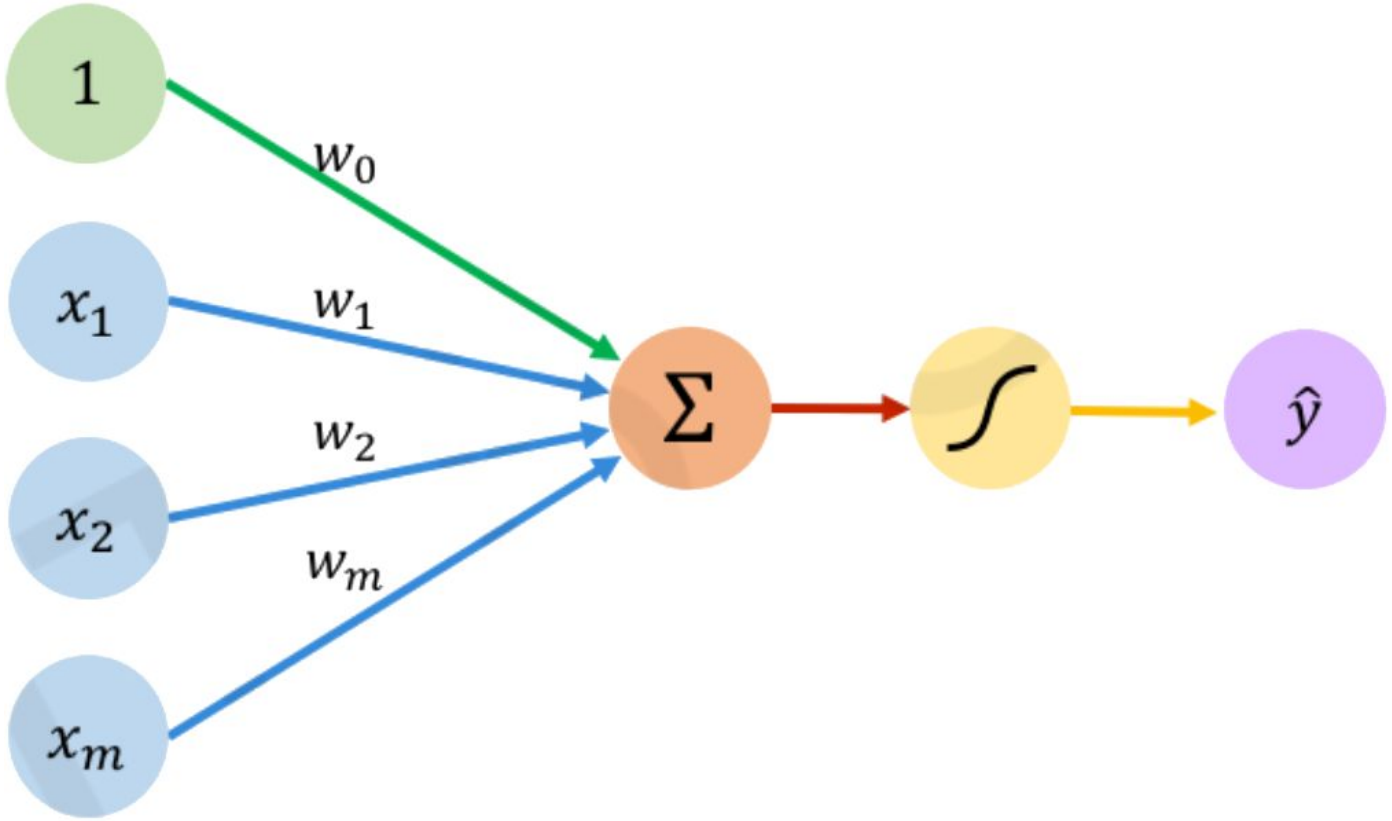
Link:

http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf

source: superdatascience.com

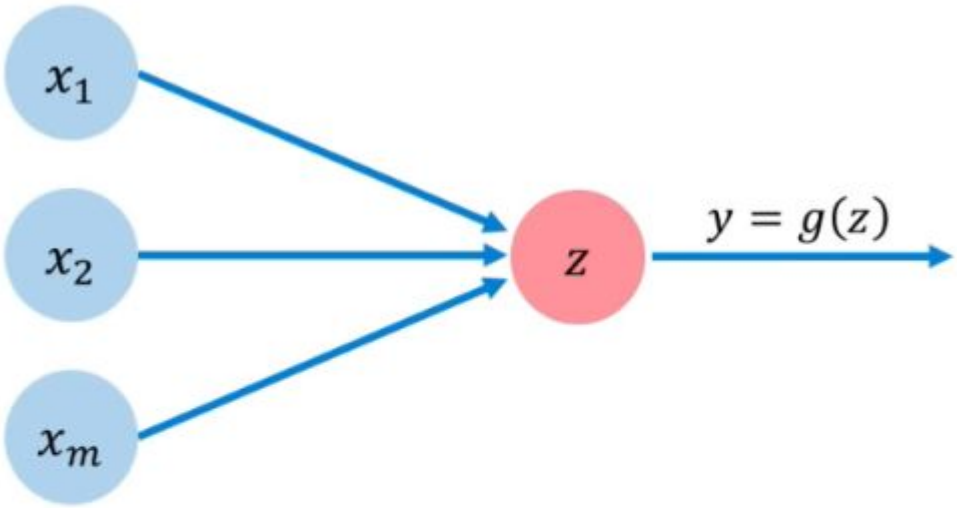
Neuron - RECAP

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$



Inputs Weights Sum Non-Linearity Output

Neuron Simplified

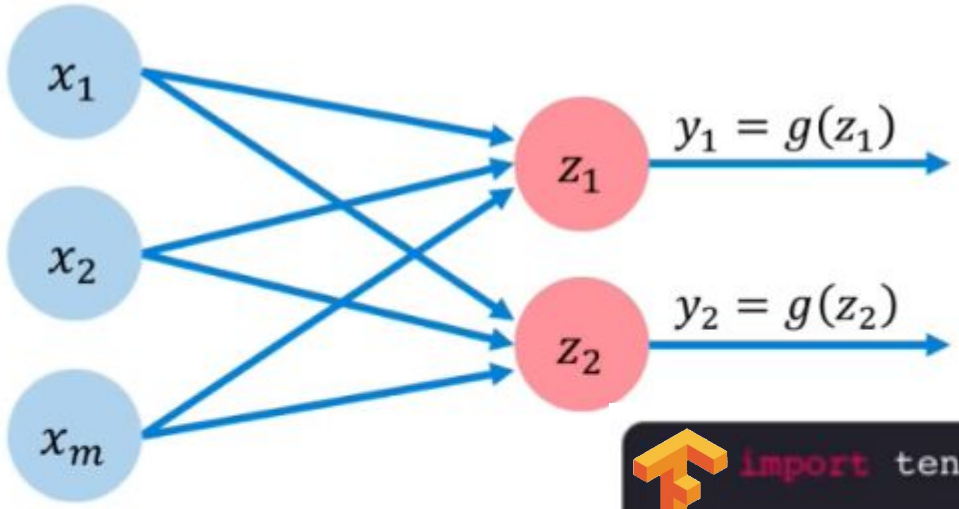


$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

Multi Output Neuron

Mondragon Unibertsitatea
Faculty of Engineering

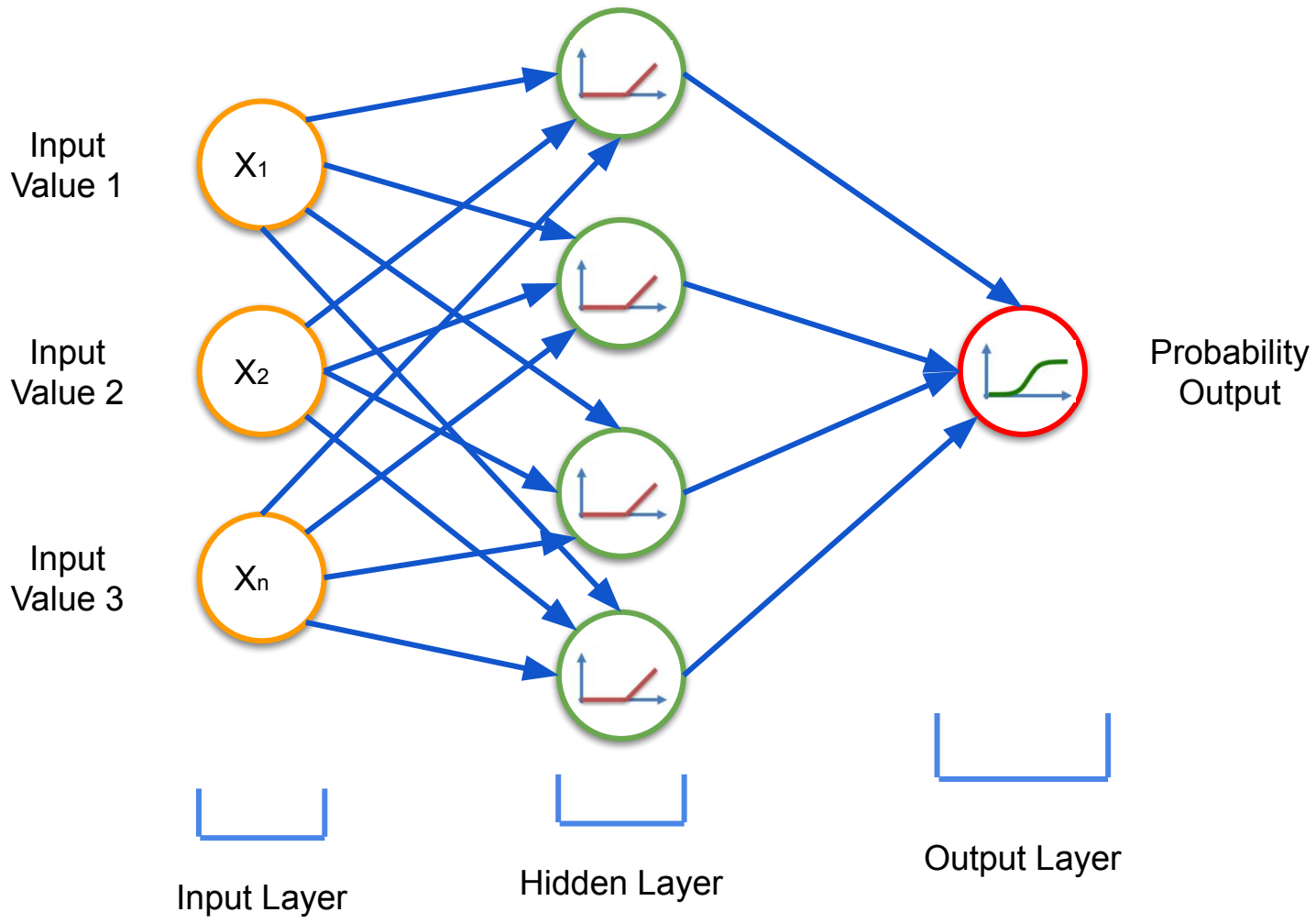
- All Inputs are densely connected to all outputs, these layers are called **DENSE** layers




```
import tensorflow as tf

layer = tf.keras.layers.Dense(
    units=2)
```

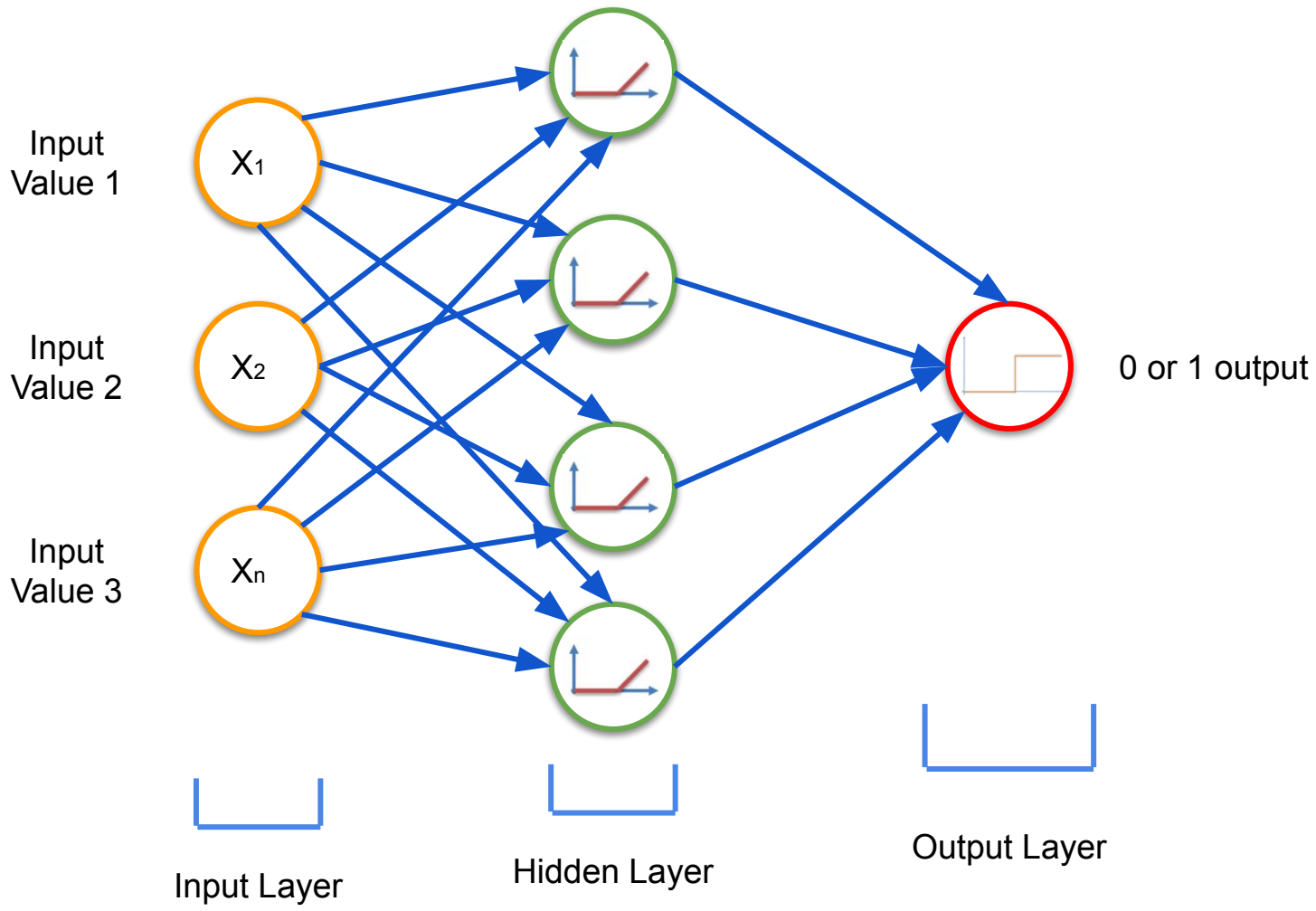
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Activation Functions



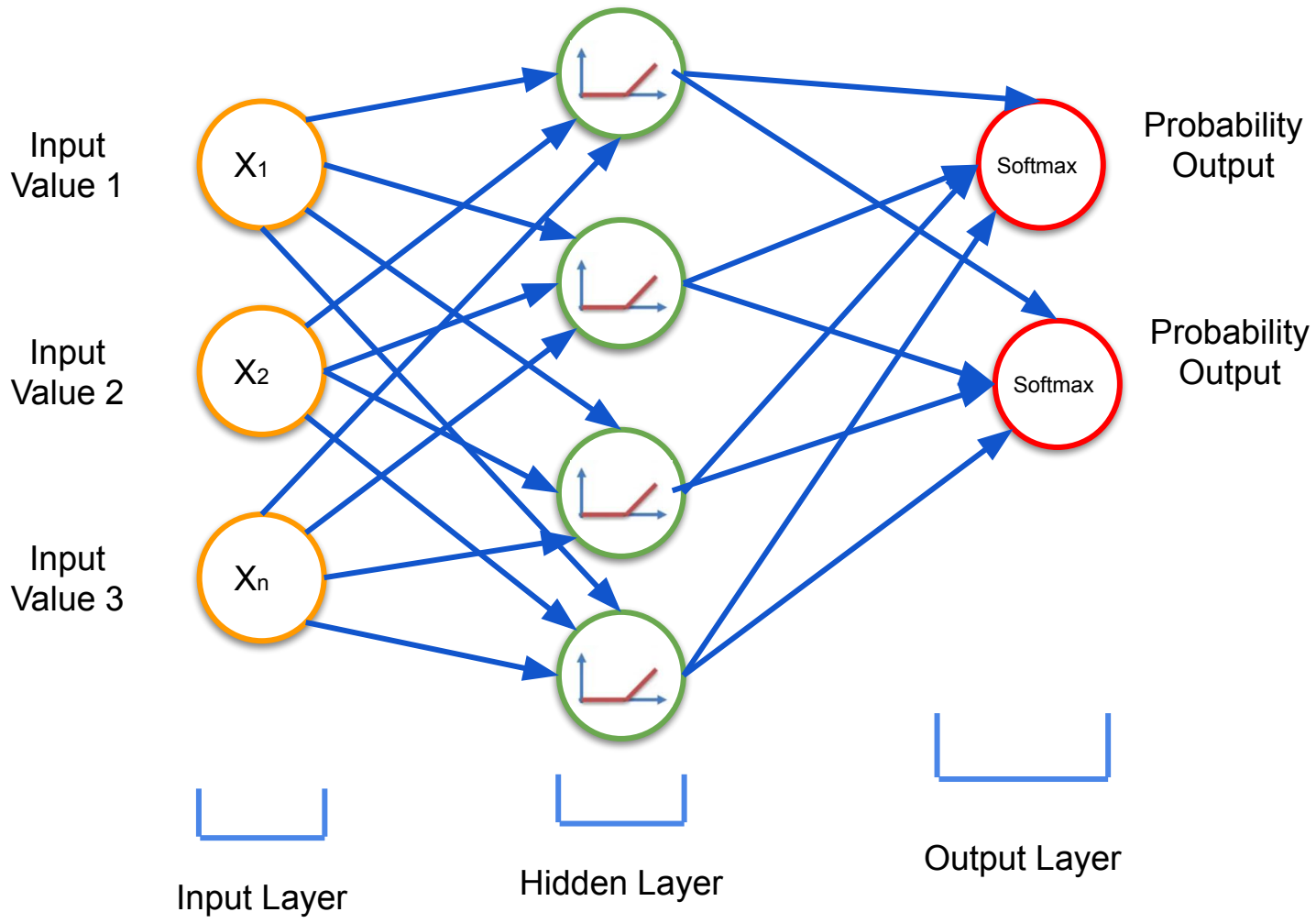
source: superdatascience.com

Activation Functions



source: superdatascience.com

Activation Functions



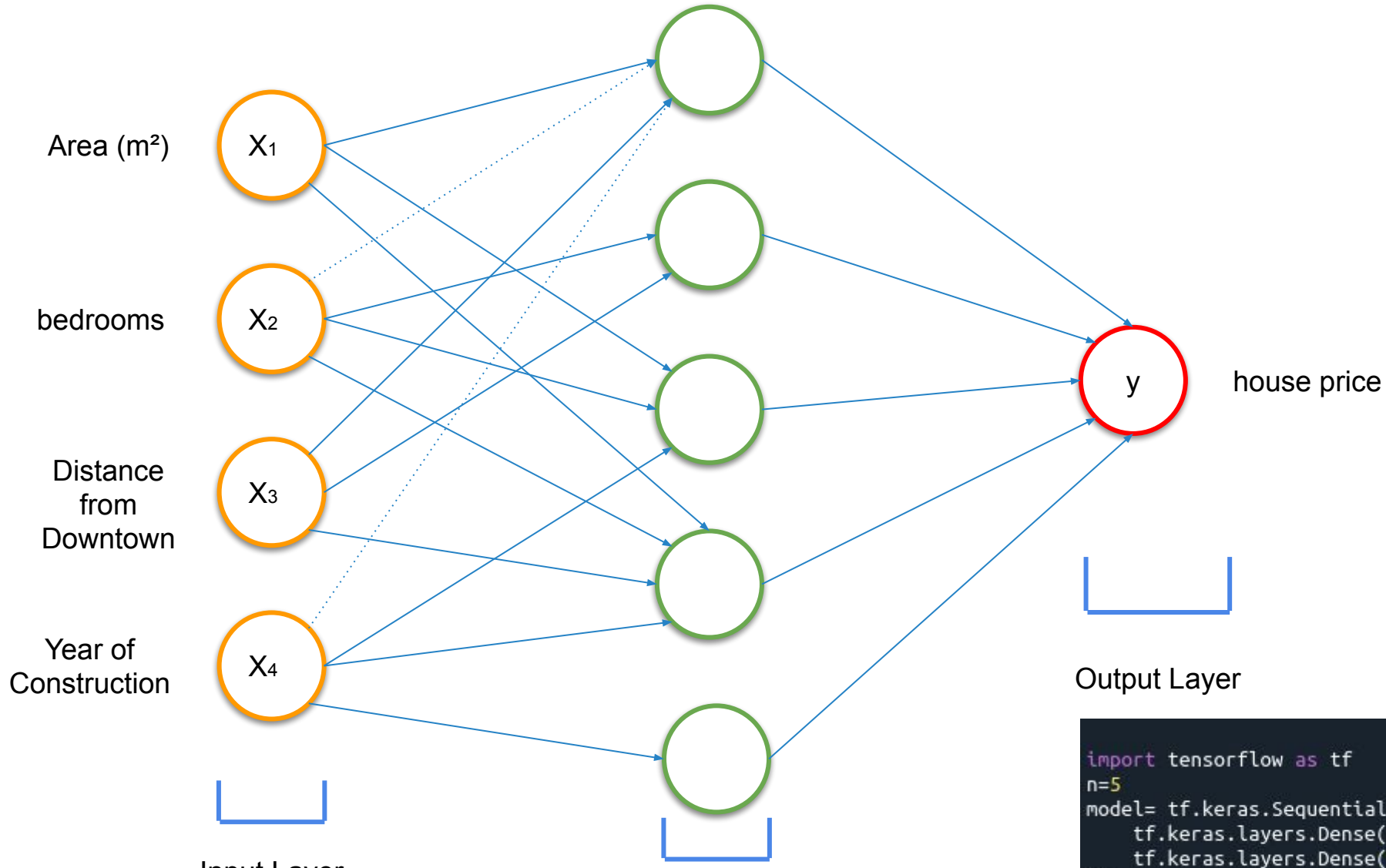
source: superdatascience.com

How does an ANN work?

Example: a house price calculation

Operation of an ANN

Mondragon Unibertsitatea
Faculty of Engineering

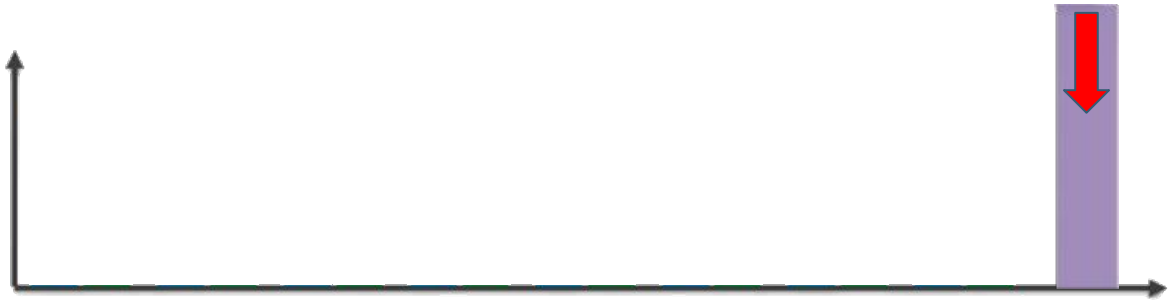



```
import tensorflow as tf
n=5
model= tf.keras.Sequential([
    tf.keras.layers.Dense(n)
    tf.keras.layers.Dense(1)
])
```

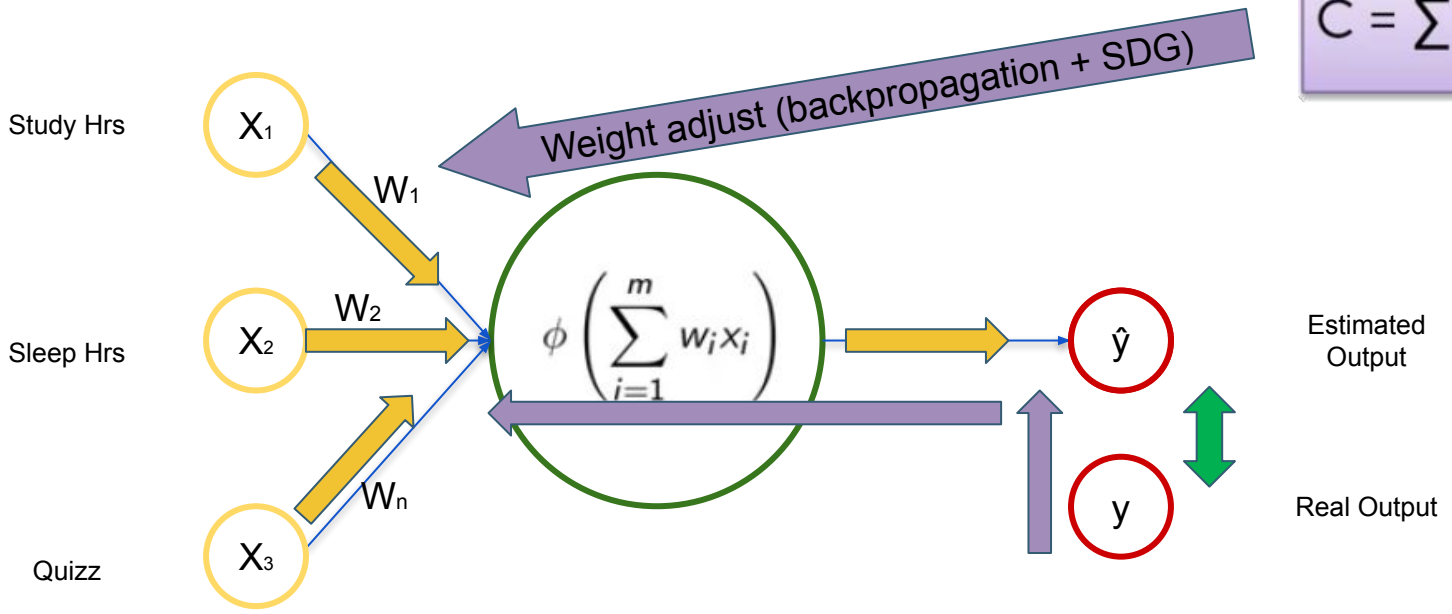
Learning Phase of an ANN

Mondragon Unibertsitatea
Faculty of Engineering

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%



$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$



Loss Functions

The most used Loss Functions

Mondragon Unibertsitatea
Faculty of Engineering

- **Regresion**
  - **Quadratic cost** (aka: *mean squared error*, maximum likelihood, and sum squared error)
  - Mean Squared Logarithmic Error Loss
  - Mean Absolute Error Loss
- **Binary Classification Loss Functions**
  - **Binary Cross-Entropy Loss**
  - Hinge Loss
  - Squared Hinge Loss
- **Multi-Class Classification Loss Functions**
  - **Multi-Class Cross-Entropy Loss**
  - Sparse Multiclass Cross-Entropy Loss
  - Kullback Leibler Divergence Loss

```
import tensorflow as tf

tf.keras.losses.mean_squared_error(y_true, y_pred)
tf.keras.losses.binary_crossentropy(y_true, y_pred)
tf.keras.losses.categorical_crossentropy(y_true,y_pred)
```

More info about loss functions

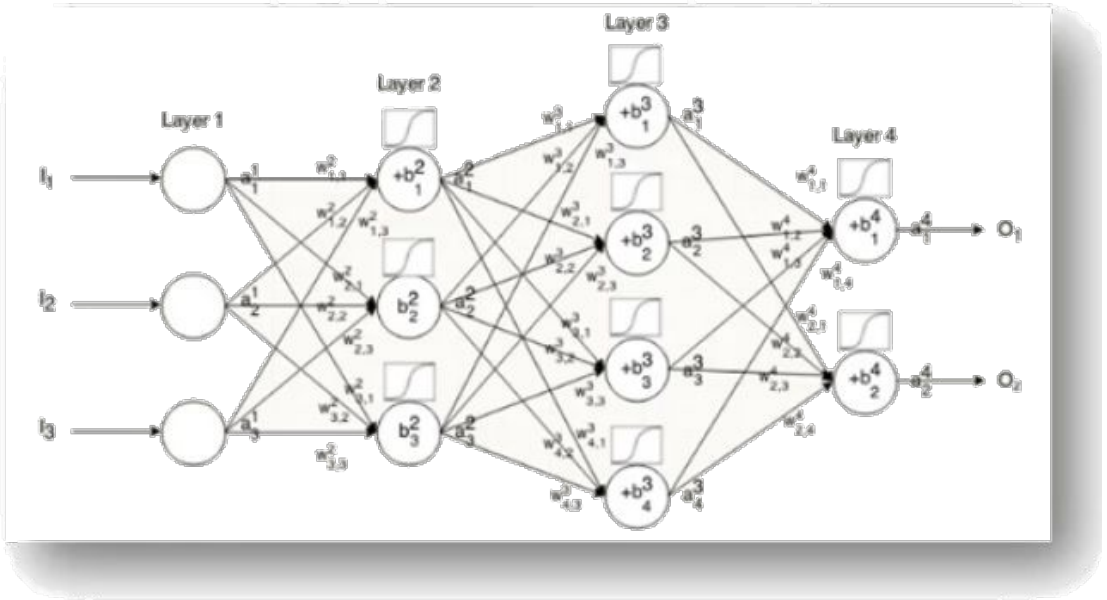
Mondragon Unibertsitatea
Faculty of Engineering

Additional Reading:

A list of cost functions used in neural networks, alongside applications

CrossValidated (2015)

Link:

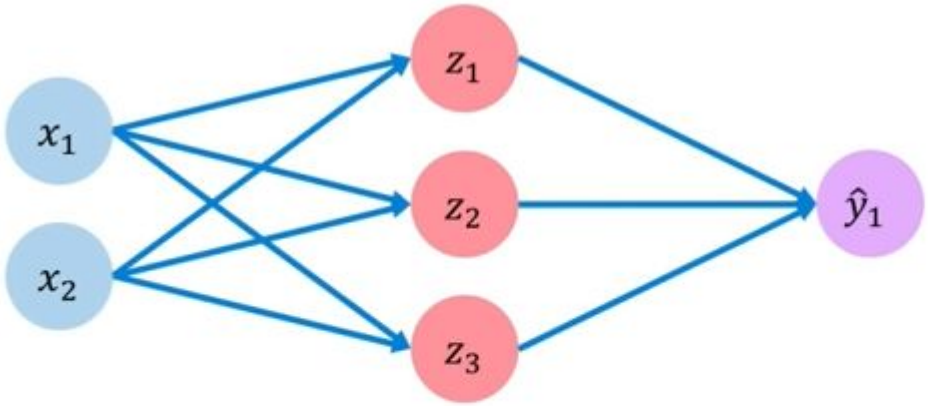


http://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications

source: superdatascience.com

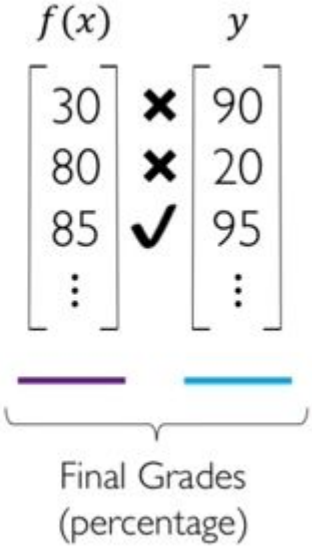
Loss Function

$$X = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2$$

Actual Predicted



source: superdatascience.com

Weight adjustment

Mondragon Unibertsitatea
Faculty of Engineering

- we can not use “Brute force” (try different weight and take the optimum) with the computation power nowadays.
  - a simple network with 4 inputs, 5 hidden neurons and 1 output = 25 weights
  - if we try 1000 different values per weight, it comes out 1000^{25} possibilities (10^{75}).
  - With the strongest Supercomputer (122 PFLOPS) we would need:

$$\frac{10^{75}}{122 * 10^{15}} = 8.2 * 10^{57}\text{seconds} \approx 2.6 * 10^{50}\text{years}$$

>Time elapsed since BigBang [~19.8x10^9 years]

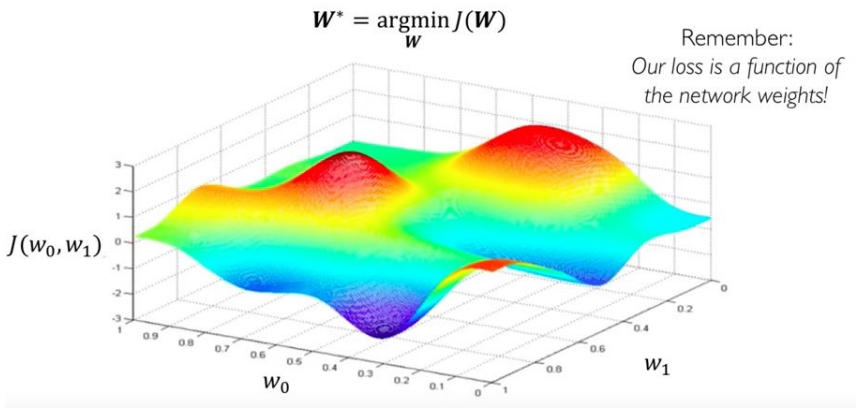
- We are in front of an OPTIMIZATION problem:
  - “**Backpropagation**” and “**Gradient Descend**” are used in order to find the optimum weight values

Optimization - Gradient Descent

Mondragon Unibertsitatea
Faculty of Engineering

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights



We want to find the optimum Weights to minimize the loss function

How can we calculate the GRADIENT?

BackPropagation

Derivatives (RECAP)

Mondragon Unibertsitatea
Faculty of Engineering

- Let us consider the following function and its partial derivatives:

$$f(x, y) = xy$$

Product of two variables.

$$\frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

Partial derivatives of the function w.r.t the inputs.

- Lets the variables be and the partial derivatives be:

$$x = 2; y = 3; f(x, y) = 6$$
$$\frac{\partial f}{\partial x} = 3; \frac{\partial f}{\partial y} = 2$$

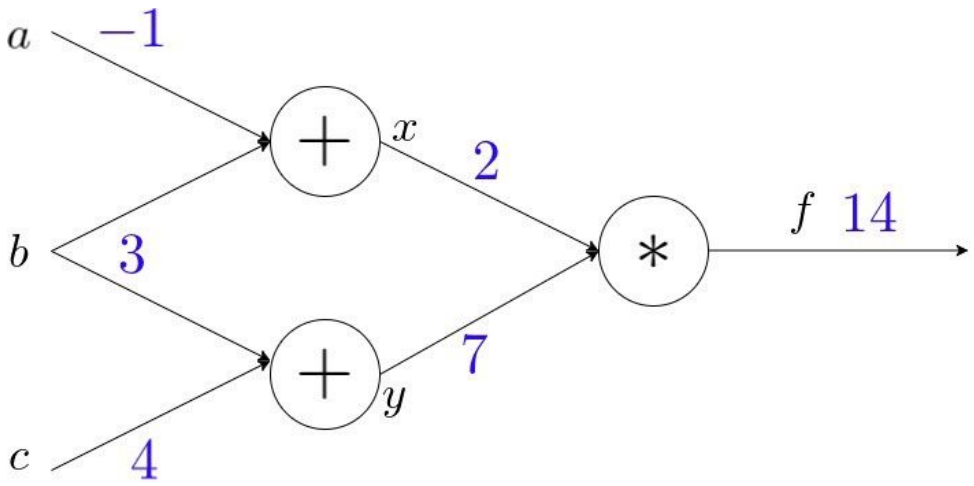
- The derivative of a function on each variable tells us the sensitivity of the function with respect to that variable

Backpropagation - Chain Rule

Mondragon Unibertsitatea
Faculty of Engineering

- In calculus, the **chain rule** is a formula to compute the derivative of a composite function
- Let us consider the complex function:

$$f = (a+b)(b+c) \qquad with \;\; a=-1, \;\; b=3, \;\; c=4$$
$$f(x,y) = xy \qquad\qquad x=2, y=7, f=14$$



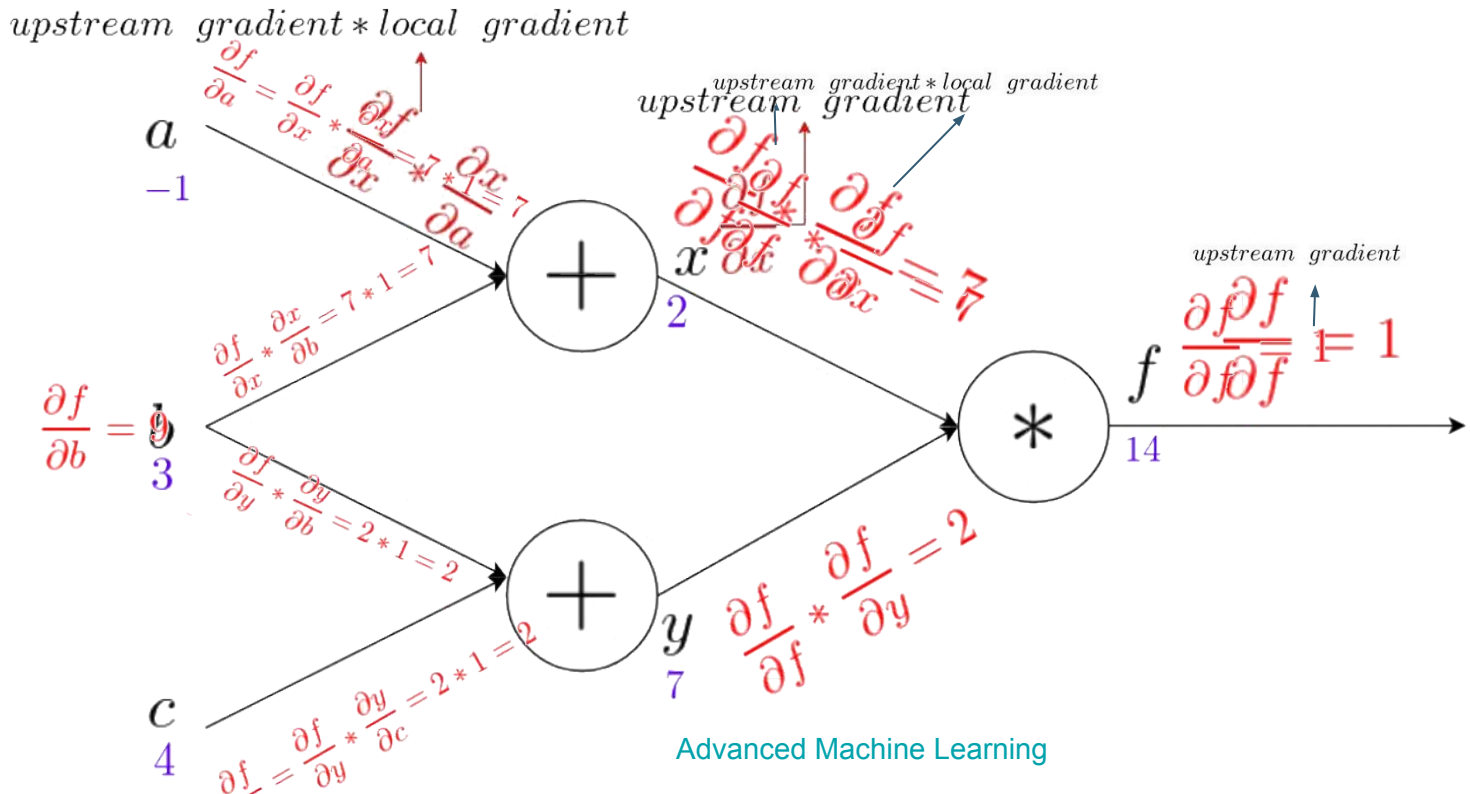
$$x = a+b \qquad \frac{\partial x}{\partial a} = 1 \qquad \frac{\partial x}{\partial b} = 1$$

$$y = b+c \qquad \frac{\partial y}{\partial b} = 1 \qquad \frac{\partial y}{\partial c} = 1$$

$$f = x*y \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

Backpropagation - chain rule

- Backpropagation is a “local” process and can be viewed as a recursive application of the **chain rule**.
- What Chain Rule says:
  - we can calculate the gradient (derivative) by multiplying the upstream gradient with the local gradient.



Backpropagation - Chain Rule

Mondragon Unibertsitatea
Faculty of Engineering

- So for Function:

$$f = (a+b)(b+c) \qquad with \quad a=-1, \quad b=3, \quad c=4$$

$$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial f} * \frac{\partial f}{\partial x} * \frac{\partial x}{\partial a} = 7$$

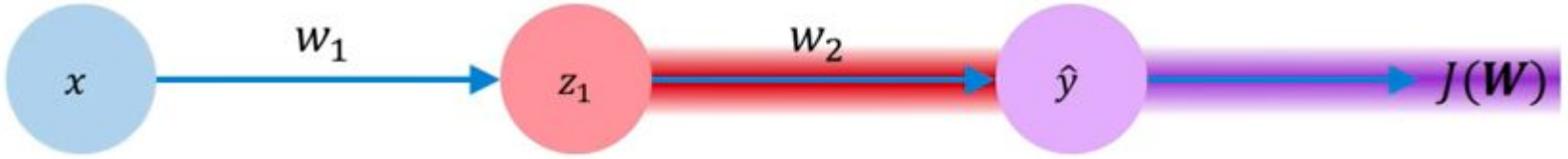
$$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial f} * \frac{\partial f}{\partial x} * \frac{\partial x}{\partial b} + \frac{\partial f}{\partial f} * \frac{\partial f}{\partial y} * \frac{\partial y}{\partial b} = 7 + 2 = 9$$

$$\frac{\partial f}{\partial c} = \frac{\partial f}{\partial f} * \frac{\partial f}{\partial y} * \frac{\partial y}{\partial c} = 2$$

Backpropagation - In NN

Mondragon Unibertsitatea
Faculty of Engineering

- For this easy NN:
  - j(w) -> The loss function, that is dependent of weights
  - example: how does a small change in W2 affect to J(W)??
  - Let us apply the **chain rule**



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

The gradient of our loss, in respect to w2

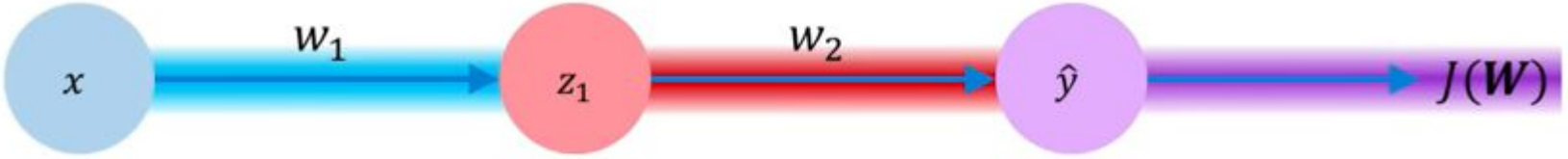
The gradient of our loss, in respect to the output

The gradient of our output, in respect to the w2

Backpropagation - In NN

Mondragon Unibertsitatea
Faculty of Engineering

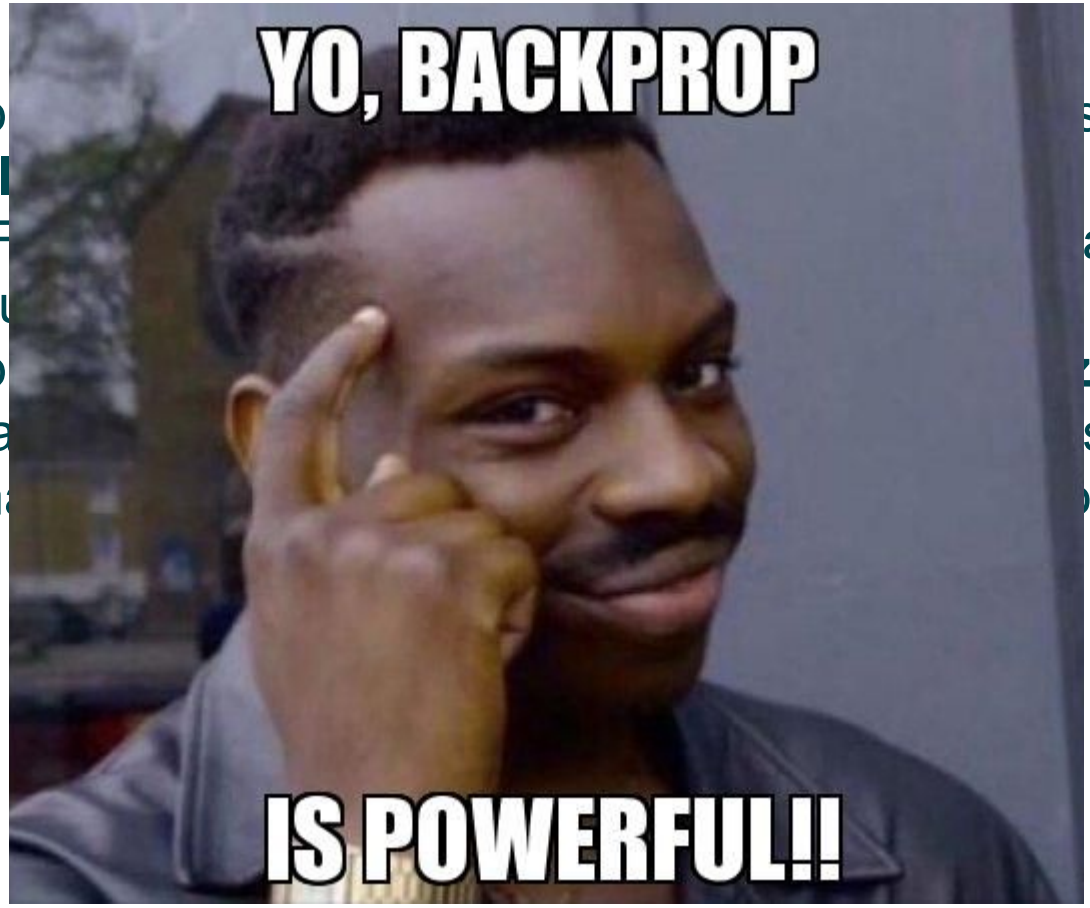
- For this easy NN:
  - And what about W1



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Backpropagation

- Backprop … s for every Weight **in a singl**
- If we do F … a calculation for every inpu
- Backprop … **zation**
- In the exa … s to optimize, but in a norma … f weights!

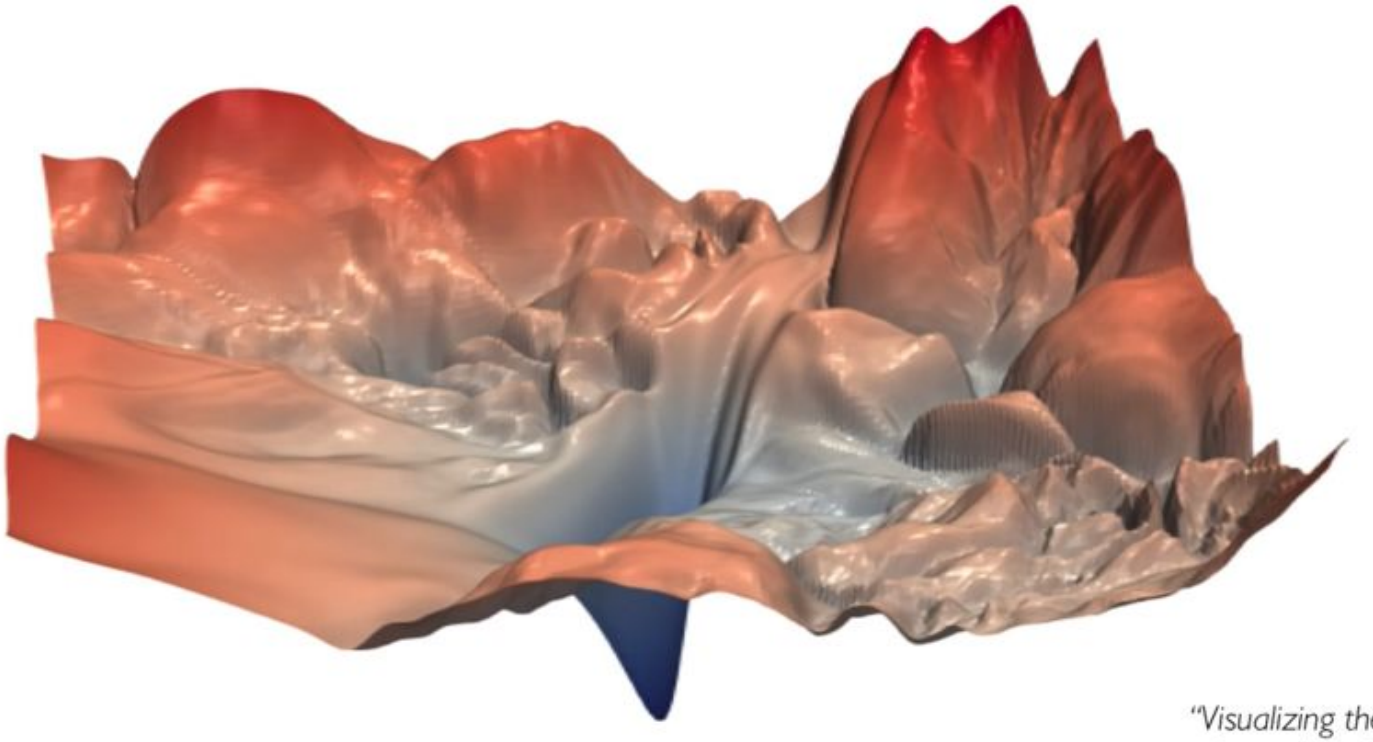


Gradient Descent

Gradient descend

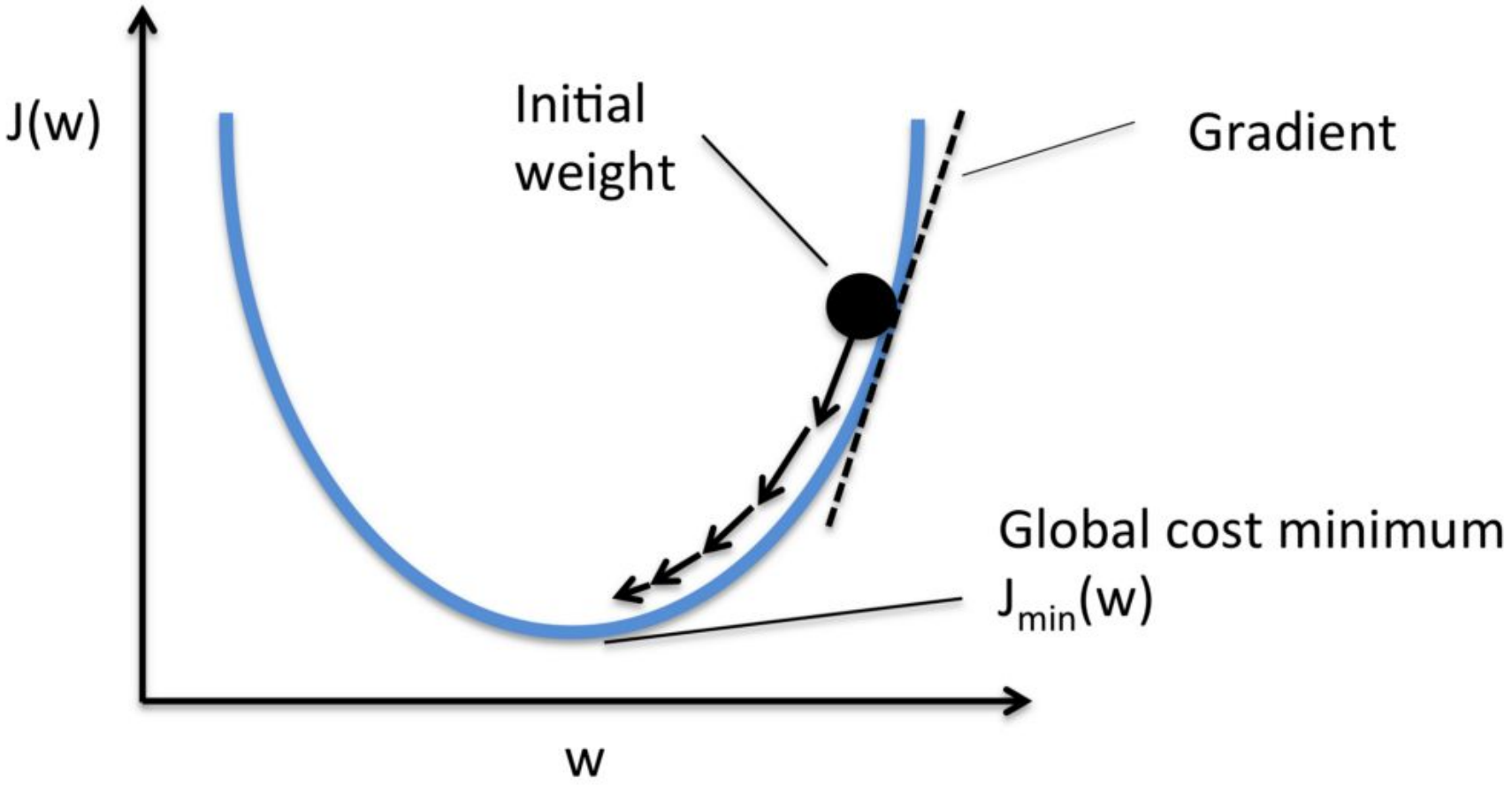
Mondragon Unibertsitatea
Faculty of Engineering

Making an optimization in such a landscape is really difficult!



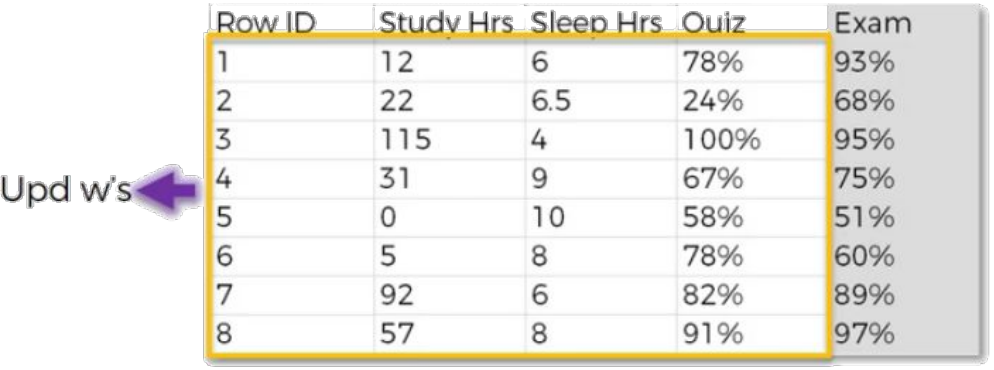
"Visualizing the loss landscape of neural nets". Dec 2017.

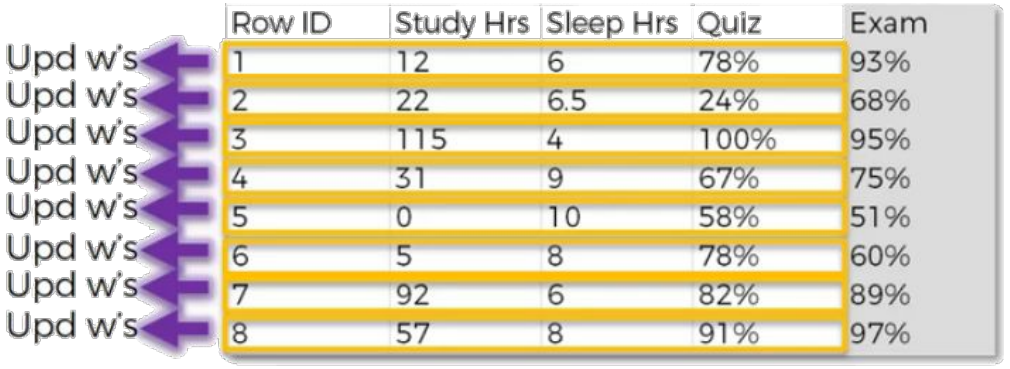
Gradient descend



Gradient descend

Mondragon Unibertsitatea
Faculty of Engineering





Batch Gradient Descent

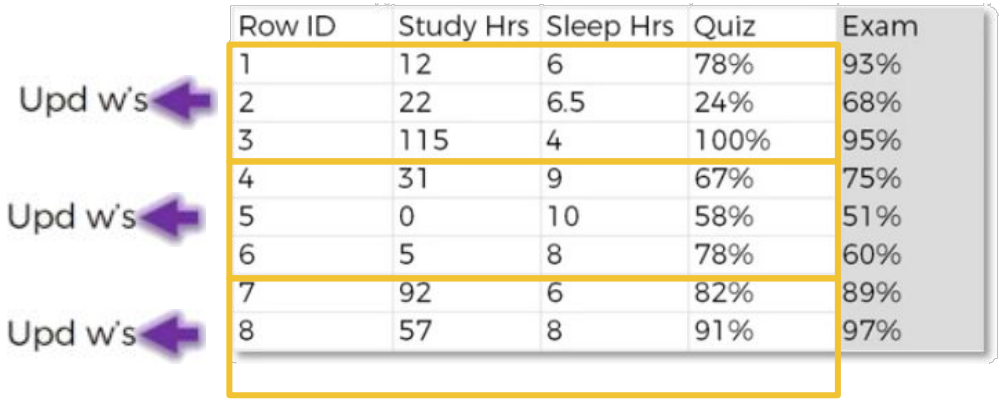
- Lots of memory
- Slow

Stochastic Gradient Descent

- Loss Function fluctuates too much
- this makes difficult to achieve optimum location

Gradient descend

Mondragon Unibertsitatea
Faculty of Engineering



Mini Batch Gradient Descent

- Good tradeoff, depending on batch size

Mini Batch SDG - selecting batch size

Mondragon Unibertsitatea
Faculty of Engineering

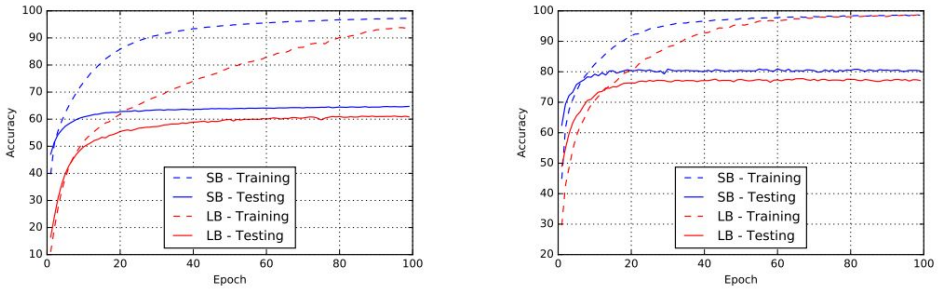
- Large-batch methods tend to converge to sharp minimizers of the training and testing functions (and that sharp minima lead to poorer generalization).
- Small-batch methods consistently converge to flat minimizers. This leads to good generalization

"We have observed that the loss function landscape of deep neural networks is such that large-batch methods are almost invariably attracted to regions with sharp minima and that, unlike small batch methods, are unable to escape basins of these minimizers."

On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima (https://arxiv.org/abs/1609.04836)

Table 2: Performance of small-batch (SB) and large-batch (LB) variants of ADAM on the 6 networks listed in Table 1

	Training Accuracy		Testing Accuracy	
Network Name	SB	LB	SB	LB
F_1	$99.66\% \pm 0.05\%$	$99.92\% \pm 0.01\%$	$98.03\% \pm 0.07\%$	$97.81\% \pm 0.07\%$
F_2	$99.99\% \pm 0.03\%$	$98.35\% \pm 2.08\%$	$64.02\% \pm 0.2\%$	$59.45\% \pm 1.05\%$
C_1	$99.89\% \pm 0.02\%$	$99.66\% \pm 0.2\%$	$80.04\% \pm 0.12\%$	$77.26\% \pm 0.42\%$
C_2	$99.99\% \pm 0.04\%$	$99.99 \pm 0.01\%$	$89.24\% \pm 0.12\%$	$87.26\% \pm 0.07\%$
C_3	$99.56\% \pm 0.44\%$	$99.88\% \pm 0.30\%$	$49.58\% \pm 0.39\%$	$46.45\% \pm 0.43\%$
C_4	$99.10\% \pm 1.23\%$	$99.57\% \pm 1.84\%$	$63.08\% \pm 0.5\%$	$57.81\% \pm 0.17\%$



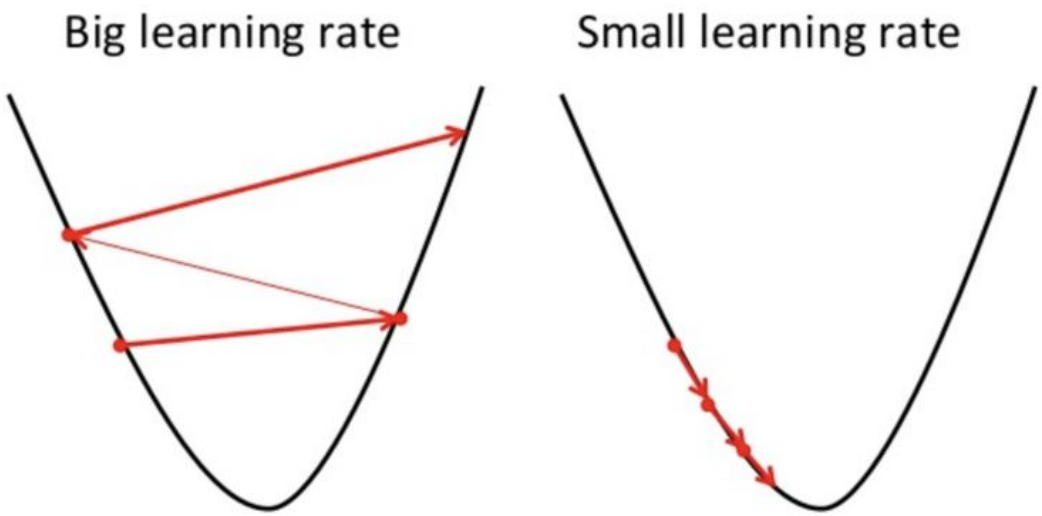
(a) F_2 (b) C_1

Figure 2: Convergence trajectories of training and testing accuracy for SB and LB methods

TensorFlow optimizers

Mondragon Unibertsitatea
Faculty of Engineering

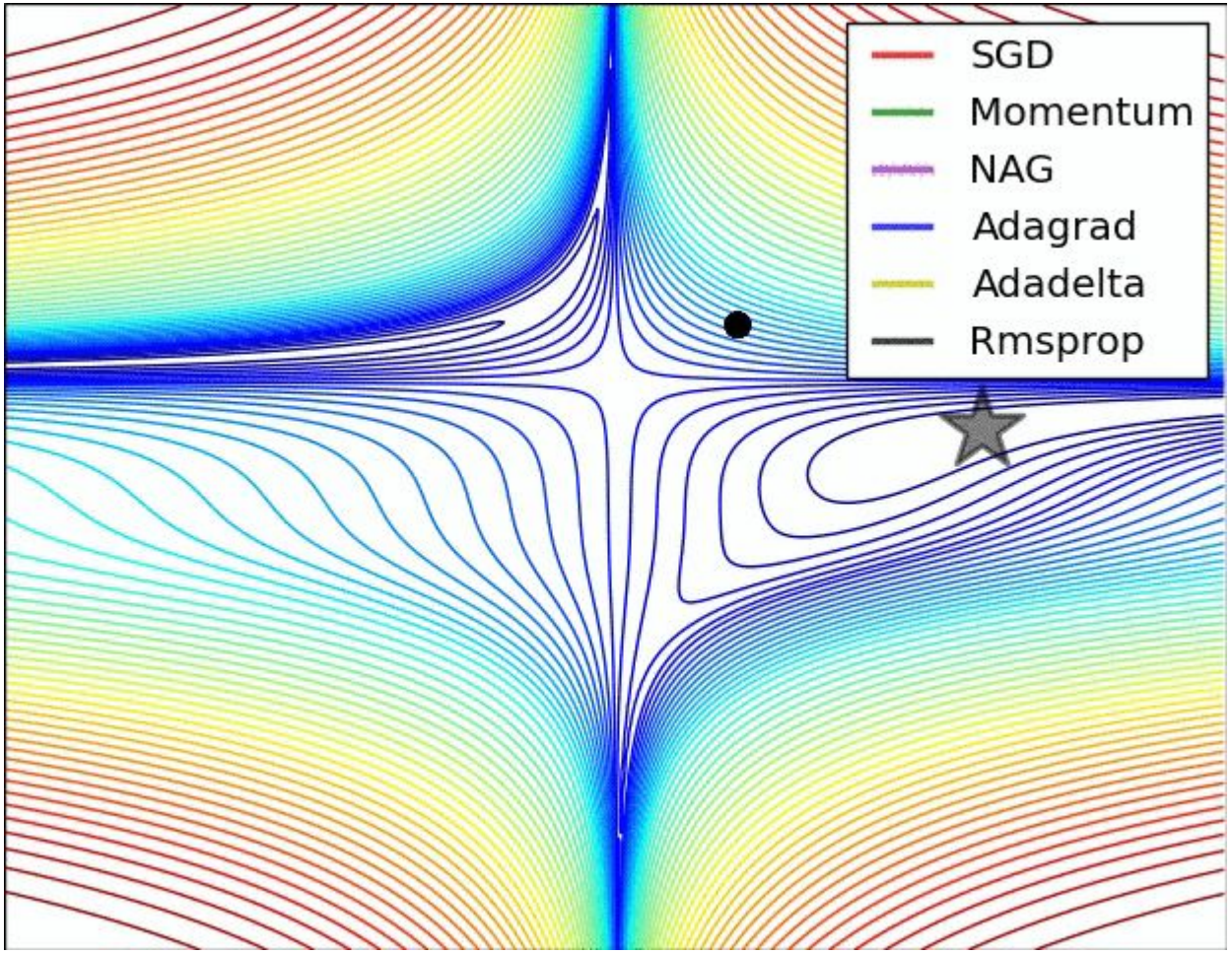
- Basically they are all based on mini batch Gradient Descent, but the implementations are different:
  - **ADAM**
  - NADAM
  - **RMSProp**
  - SDG
- In all of them, there is a value to be fixed: The **Learning Rate**
  - Big Learning rate converges fast but does not achieve best minimum
  - Small Learning rate achieves best minimum, but it takes more time
- **ADAM, RMSProp** have the possibility to make the learning rate adaptive:
  - big at the beginning, small at the end



"The learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate."

— Page 429, Deep Learning, 2016.

Optimizers



Stick it together...

1. The **LOSS** quantifies the quality of your Weights

2. The **Backpropagation** calculates the *gradients of our loss function* with respect to all the weights of the Neural Network.

3. The **Gradient Descent** based optimization, updates the model parameters (weights) in order to *optimize the loss function*

ML basics for NN (RECAP)

Syllabus

Mondragon Unibertsitatea
Faculty of Engineering

- Parts of this subsection:
  - Data Representations for NN
  - Tensor operations
  - Evaluating ML models
  - Overfitting vs Underfitting
  - Dealing with Overfitting in NNs
  - Universal workflow of ML-DL

Data Representations for NN

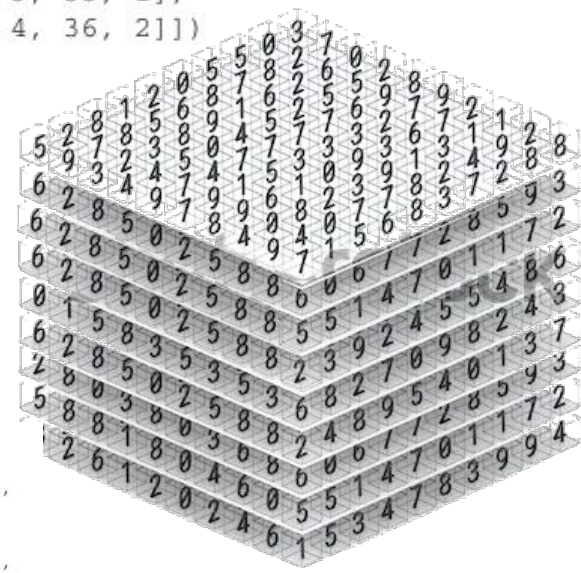
Mondragon Unibertsitatea
Faculty of Engineering

- Scalars (0D tensor)
  - 0 dimensional TENSOR
  - it is a value that its rank or axes is 0
- Vectors (1D tensor)
  - an Array of values is called vector or 1d tensor
- Matrices (2D tensors)
  - A matrix has 2 axes (rows and columns)
- 3D tensors
  - you can pack matrices in a new array
  - This is a 3D tensor, a cube of numbers
- 4D and 5D tensors
  - image and video data

```
>>> import numpy as np
>>> x = np.array(12)
>>> x
array(12)
>>> x.ndim
0
```

```
>>> x = np.array([12, 3, 6, 14])
>>> x
array([12, 3, 6, 14])
>>> x.ndim
1
```

```
>>> x = np.array([[5, 78, 2, 34, 0],
                  [6, 79, 3, 35, 1],
                  [7, 80, 4, 36, 2]])
>>> x.ndim
2
```

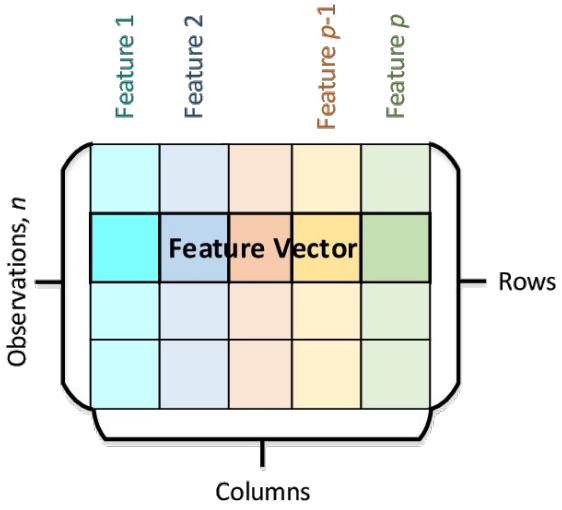


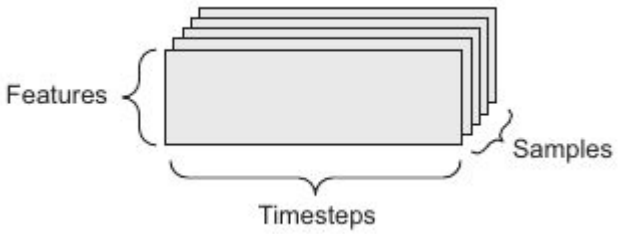
```
>>> x = np.array([[[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]],
                  [[5, 78, 2, 34, 0],
                   [6, 79, 3, 35, 1],
                   [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

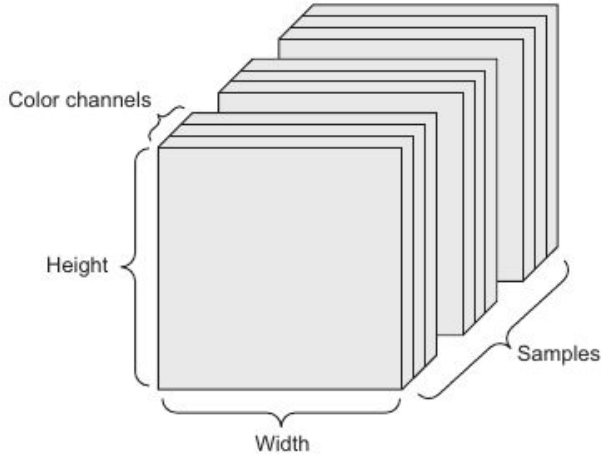
Data Representations for NN

Mondragon Unibertsitatea
Faculty of Engineering

- Real world examples of data tensors
  - **Vector Data** - 2D tensors (samples, features)
  - **TimeSeries Data**- 3D tensors (samples,timesteps,features)
  - **Images** - 4D tensors (samples,height,width,channels)
  - **Video Data** - 5D tensors (samples,frames,height,width,channels)







Tensor Operations

Mondragon Unibertsitatea
Faculty of Engineering

- Element wise operations

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 9 \\ 16 & 25 & 36 \\ 49 & 64 & 81 \end{bmatrix}$$

```
import numpy as np

A=np.array([[1,2,3],[4,5,6],[7,8,9]])

print(A*A)
```

- Tensor dot

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} .* \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \\ 102 & 126 & 150 \end{bmatrix}$$

```
np.dot(A,A)
```

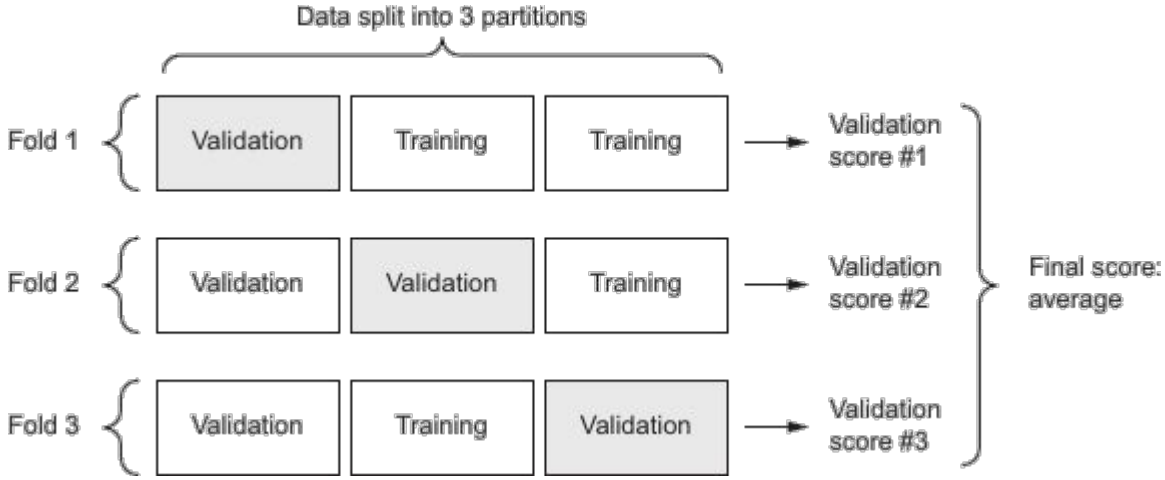
How many? which ones?

1. Supervised Learning
   - learning to map input data to known targets
2. Unsupervised learning
   - Consists of finding interesting correlations in data
   - Using transformations, compression, denoissing…
   - no targets
   - Necessary step to solve supervised problems
3. Self-Supervised Learning
   - supervised learning without humans in the loop (no human annotated labels)
   - also valid for anomaly detection problems (only learns with one target class)
4. Reinforcement Learning
   - Agent receives information about its environment and learns to actuate to maximize reward
   - example: NN learns to play ARKANOID

Evaluating ML models

Mondragon Unibertsitatea
Faculty of Engineering

- Train, validation and test datasets
  - Simple Hold-out validation
  - K-fold validation
  - Iterated K-fold validation with shuffling
    - Applying k-fold multiple times (P times), shuffling the data each time.
    - you will end up with KxP models, it can be expensive



https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/

Things to keep in mind

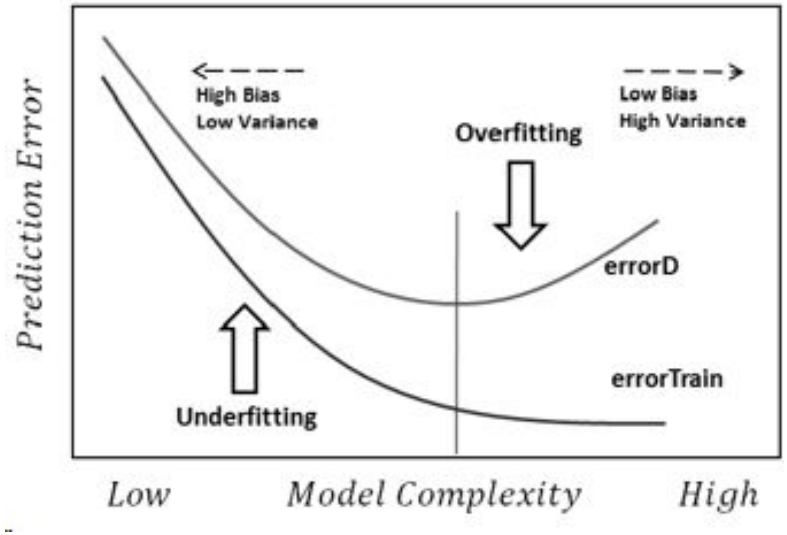
Mondragon Unibertsitatea
Faculty of Engineering

- Data Representativeness:
  - Train and Test sets must contain representative data.
- The Arrow of time:
  - If you are trying to predict the future, you should not shuffle your data
- Redundancy in your data:
  - If some points appear twice, then shuffling the data and splitting into train/test will result in redundancy, meaning that you are testing your model on part of your training data… and this is not correct.
  - Make sure your training and validation are disjoint

Overfitting and underfitting

Mondragon Unibertsitatea
Faculty of Engineering

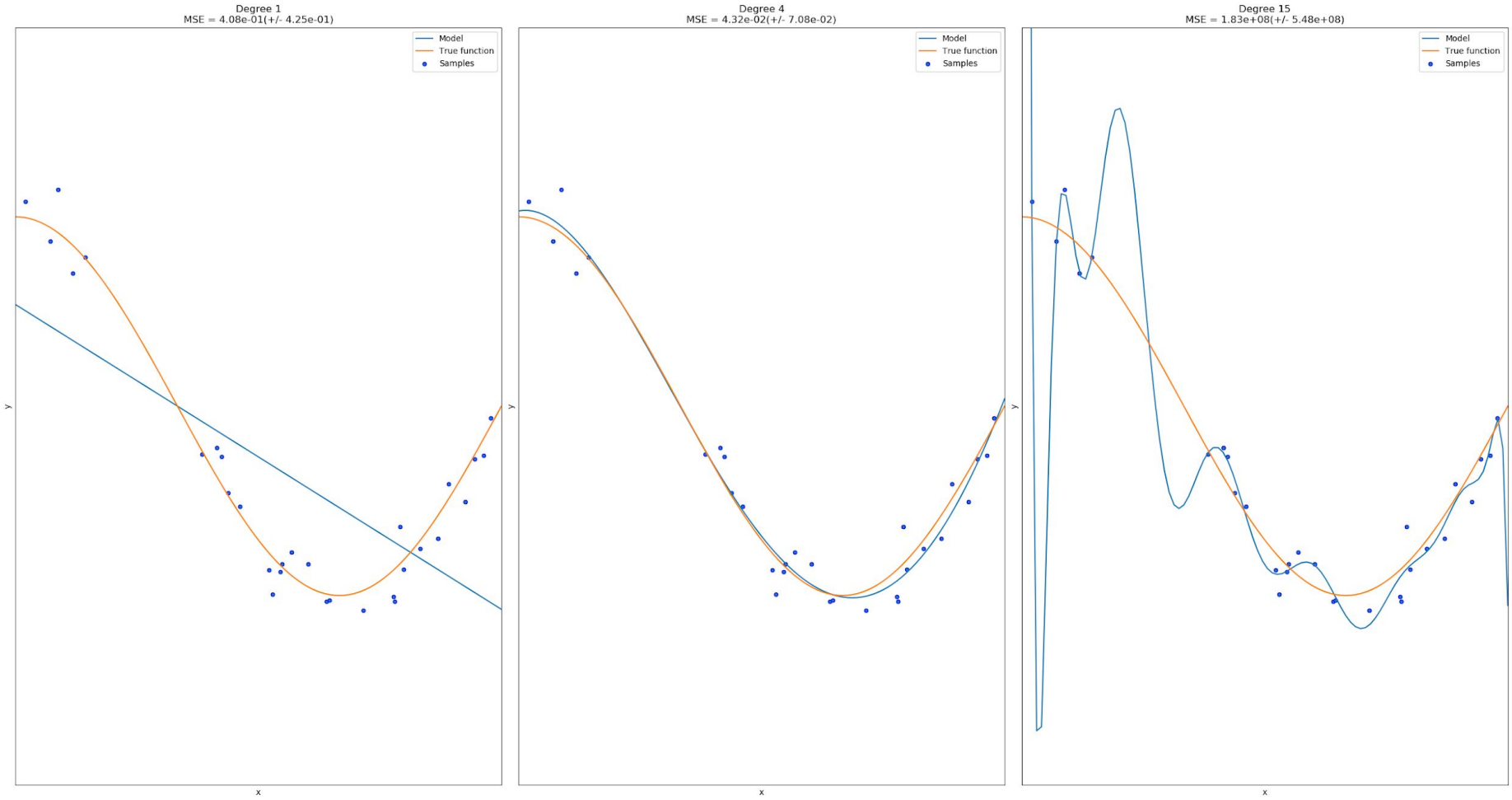
- Deep Neural Networks tend to overfitting
  - They have billions of neurons, they can model (almost) anything.
- complex - ML issue: “Optimization VS Generalization”
  - Optimization: *adjusting a model to get the best performance in training data*
  - Generalization: *how well performs the model with data it has never seen before*
- UnderFit: when the model has not modeled all relevant patterns (more epochs needed)
- Overfit: when OPTIMIZATION is getting better but GENERALIZATION is not getting better



Overfitting and underfitting

Mondragon Unibertsitatea
Faculty of Engineering

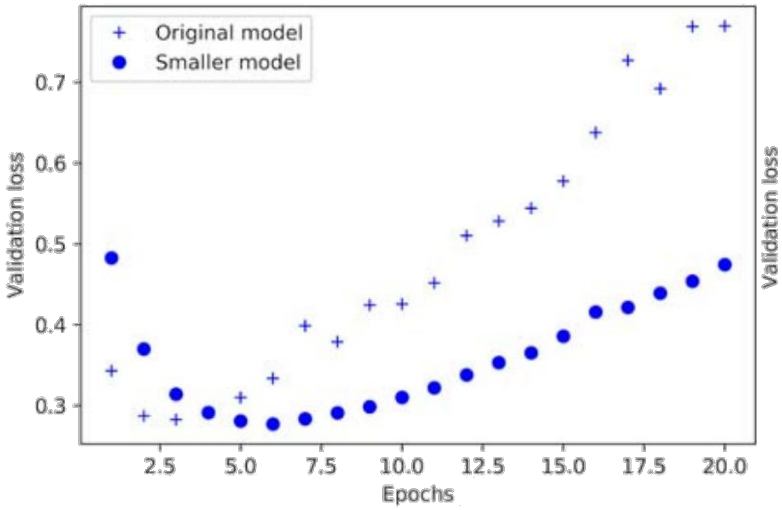
- Underfitting VS Overfitting

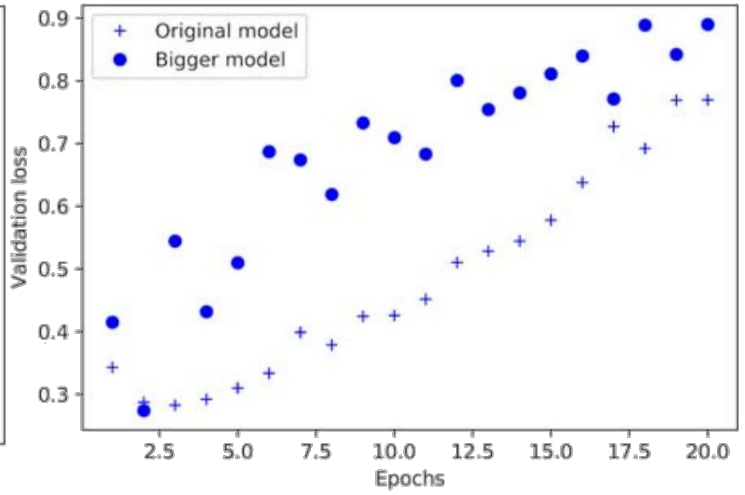


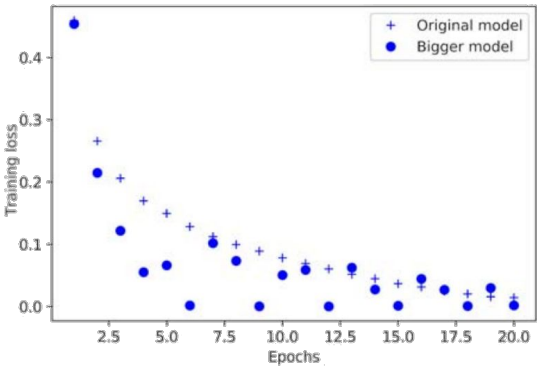
Dealing with OverFitting in NN

Mondragon Unibertsitatea
Faculty of Engineering

- Strategy 1: **Get More DATA**!
  - this prevents a model from learning misleading or irrelevant patterns in training data
- Strategy 2: **Reduce the Network size**
  - As we have seen in the example, if we have too many parameters, we can end up with an overfit model.
  - By reducing the network size, we reduce learnable parameters in the model
  - learnable parameters in the model = *Capacity*
  - Sometimes too much *Capacity* leads to "dictionary like" models.
  - but too low *Capacity* may lead to underfitting
  - **THERE IS NO MAGIC FORMULA TO DETERMINE THE CAPACITY**



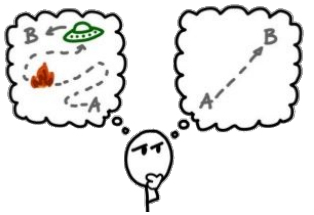




Dealing with OverFitting in NN

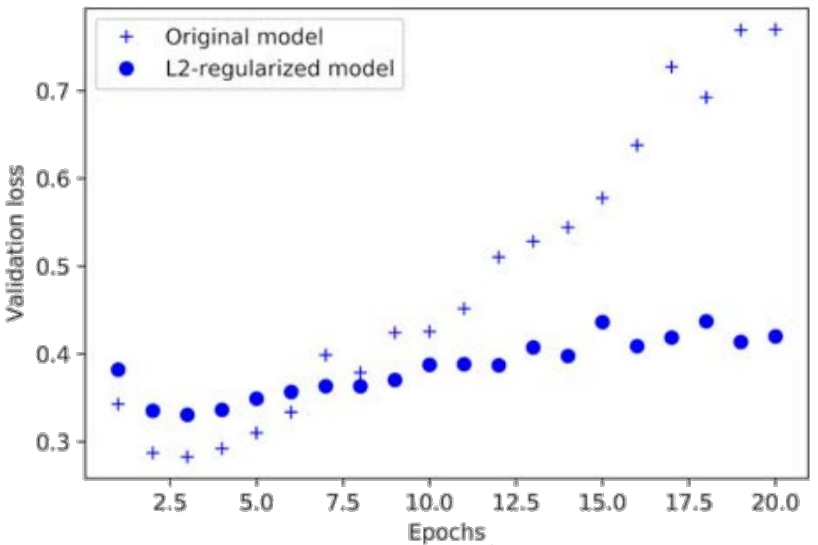
Mondragon Unibertsitatea
Faculty of Engineering

Occam's Razor



*"When facing with two equally good hypotheses, **always choose the simpler one**."*

- Strategy 3: Adding **weight regularization**
  - Multiple sets of weight values could explain the data
  - simpler models are less likely to overfit
  - Simple model:
    - **Distribution of parameters has less entropy**
  - force the weights to take only small values
  - made using L1 and L2 regularizers, monitoring the cost associated to large weights, adding this cost to the loss function.
    - L1 regularization: cost monitoring "absolute value of the weight coefficients"
    - L2 regularization: cost monitoring " square of the value of the weight coefficients"



$$y = Wx + b$$

- Kernel_regularizer: Weights are regularized
- Bias_regularizer: Biass is regularized
- Activity_regularizer: y is regularized

```
import tensorflow as tf
from tf.keras import models,layers,regularizers

model = models.Sequential()
model.add(layers.Dense(16,
                       kernel_regularizer=regularizers.l1(0.001),
                       activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16,
                       kernel_regularizer=regularizers.l2(0.001),
                       activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

l2(0.001) means every coefficient in the weigl matrix of the layer will add 0.001 * weight_coefficient_value to the total loss of the network.

Dealing with OverFitting in NN

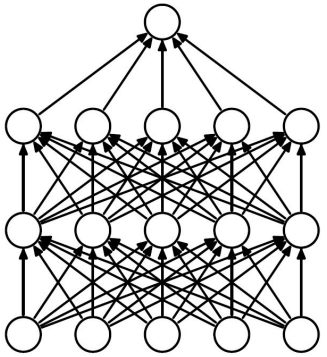
Mondragon Unibertsitatea
Faculty of Engineering

- Strategy 4: Adding **Dropout**
  - Most effective regularization technique
  - Developed by Geoffrey Hinton
  - Basically randomly drops out several neurons in each layer every epoch

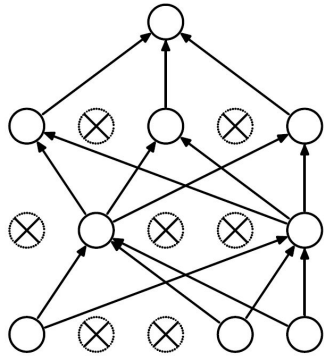
0.3	0.2	1.5	0.0
0.6	0.1	0.0	0.3
0.2	1.9	0.3	1.2
0.7	0.5	1.0	0.0

50% dropout

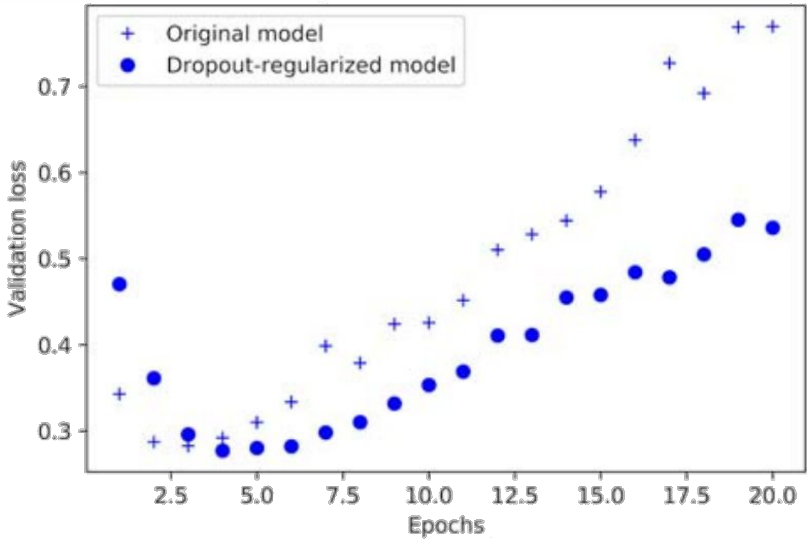
0.0	0.2	1.5	0.0
0.6	0.1	0.0	0.3
0.0	1.9	0.3	0.0
0.7	0.0	0.0	0.0



(a) Standard Neural Net



(b) After applying dropout.



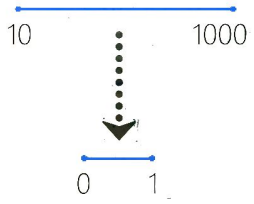

```python
import tensorflow as tf
from tensorflow.keras import models,layers,regularizers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation='sigmoid'))
```

Dealing with OverFitting in NN

Mondragon Unibertsitatea
Faculty of Engineering

- Strategy 5: Adding **Batch Normalization**
  - It is not exactly a regularization technique
  - Speeds up training process
  - Normalizes every output of each layer
    - This way, there is no any big weight that makes the network unstable
  - parameters are optimizable
  - HELPS with OVERFITTING

Normalize



Standardize

$$z = \frac{x - m}{s}$$

Batch Normalization

1. Normalize output from activation function.

$$z = \frac{x - m}{s}$$

2. Multipy normalized output by arbitrary parameter, g.

$$z * g$$

3 Add arbitrary parameter, b, to resulting product.

$$(z * g) + b$$

```
import tensorflow as tf
from tensorflow.keras import models,layers,regularizers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu',
                       input_shape=(10000,)))
model.add(layers.BatchNormalization())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(1, activation='sigmoid'))
```

Dealing with OverFitting in NN

Mondragon Unibertsitatea
Faculty of Engineering

- Strategy 6: **Early Stopping**
  - Just stop before the Overfitting starts

```
from tensorflow.keras import callbacks

ES=callbacks.EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
model.fit(train_X, train_y, callbacks=ES)
```

Tuning hyperparameters in Keras

Mondragon Unibertsitatea
Faculty of Engineering

- Strategy 1: **GridSearch**
  - Use gridSearch from scikit-learn
    - GridSearchCV
    - RandomizedSearchCV

- Strategy 2: Use optimization libraries
  - hyperas : **https://github.com/maxpumperla/hyperas**

Remember:
Using k-fold cross validation, gridSearch, or optimization libraries, multiplies your training time several times, and DeepLearning models tend to have long fitting times….

HANDLE these solutions with **CARE**

Universal workflow of ML-DL

Mondragon Unibertsitatea
Faculty of Engineering

1. Defining the problem and assembling a dataset
   a. What will your input data be? what are you going to predict?
   b. what type of problem are you facing?
      i. binary classification
      ii. multiclass classification
      iii. regresion
      iv. clustering
      v. …
   c. Make a hypothesis: my X is able to predict y (you will validate it later)

2. choose a measure of success
   a. accuracy?
   b. Precision?
   c. Recall?
   d. AUROC?

Universal workflow of ML-DL

Mondragon Unibertsitatea
Faculty of Engineering

3. Deciding on an evaluation protocol
   a. Maintaining a hold-out validation set
      i. way to go if you have plenty of data
   b. doing K-fold cross-validation
      i. if you have no enough data for hold-out
   c. Iterated k-fold validation
      i. in order to have highly accurate model on little data

4. preparing your data (for DL)
   a. put data into tensors
   b. data should be scaled -> [0,1], [-1,1]
   c. if data is heterogeneous, Standardization should be performed.
   d. (feature engineering)

Universal workflow of ML-DL

Mondragon Unibertsitatea
Faculty of Engineering

5. Developing a first model
   a. Goal: get a model that has “statistical power”: a small model that proves that the problem is doable
   b. choices to be made:
      i. Last-layer activation selection
      ii. loss function selection
      iii. optimization configuration (Adam, rmsprop)

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

```
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
model.add(layers.Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

```
model.add(layers.Dense(10, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam')
```

```
model.add(layers.Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
```

```
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='mean_squared_error', optimizer='adam')
```

Universal workflow of ML-DL

Mondragon Unibertsitatea
Faculty of Engineering

6. Scaling up: developing a model that overfits
   a. once we have a model that has “statistical power”, can we make it more powerful?
   b. in order to find a model that is *optimum* and *general,* first we need to cross the border and make a model that overfits, in order to find the best model.
      i. Add layers
      ii. make layers bigger
      iii. train for more epochs
7. Regularizing your model
   a. Once we have an overfit model, add regularization:
      i. Add dropout
      ii. Add batch-normalization
      iii. early-stopping
      iv. try different architectures: add/remove layers
      v. add L1/L2 regularization
8. Tune hyperparameters
   a. Make a hyperparameter tuning in order to find the best model
      i. grid search
      ii. Keras hyperopt
9. Validate final model!!!!!!!!!

Exercises

Mondragon Unibertsitatea
Faculty of Engineering

1. First Steps - Mnist dataset classification problem
2. Sentiment Analysis exercise
3. bank customers modeling exercise

Mondragon Unibertsitatea

Faculty of Engineering

Eskerrik asko

Muchas gracias

Thank you

Ekhi Zugasti
ezugasti@mondragon.edu

Loramendi, 4. Apartado 23
20500 Arrasate – Mondragon
T. 943 71 21 85
info@mondragon.edu