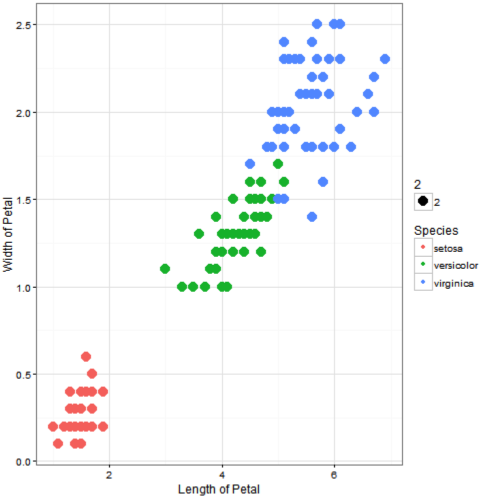
Introduction to Deep Learning

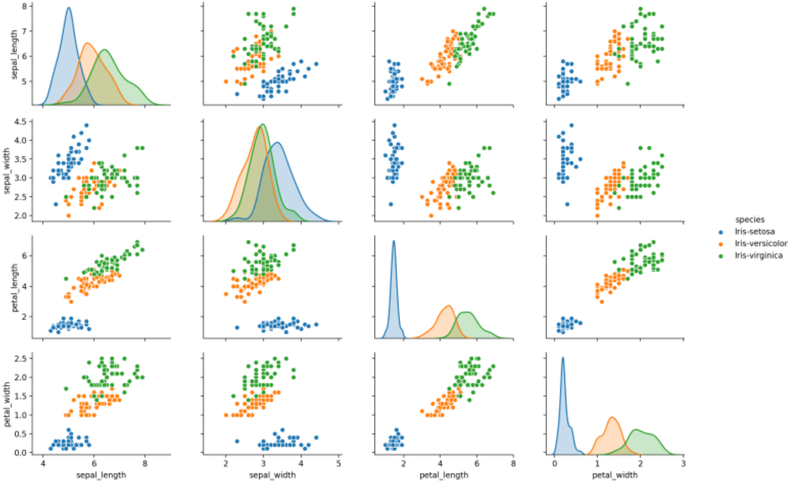
Cristian Pachón García

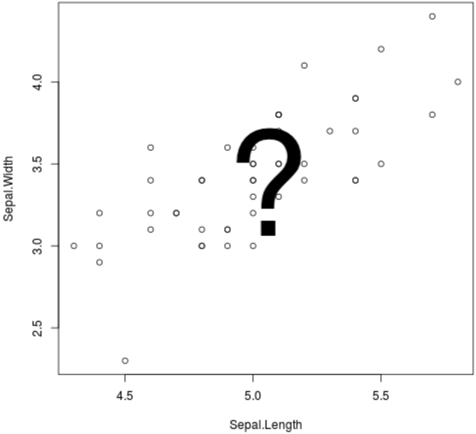
February 4, 2020

What is Machine Learning?

- Machine learning (ML) is the **scientific study** of **algorithms** and **statistical models** that computer systems use to perform a **specific task without using explicit instructions**, relying on patterns **learn from data**.
- The process of making the machine learn is called **the training process**.


2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Width of Petal
Length of Petal
2
2
Species
setosa
versicolor
virginica


sepal_length
sepal_width
petal_length
petal_width
species
Iris-setosa
Iris-versicolor
Iris-virginica

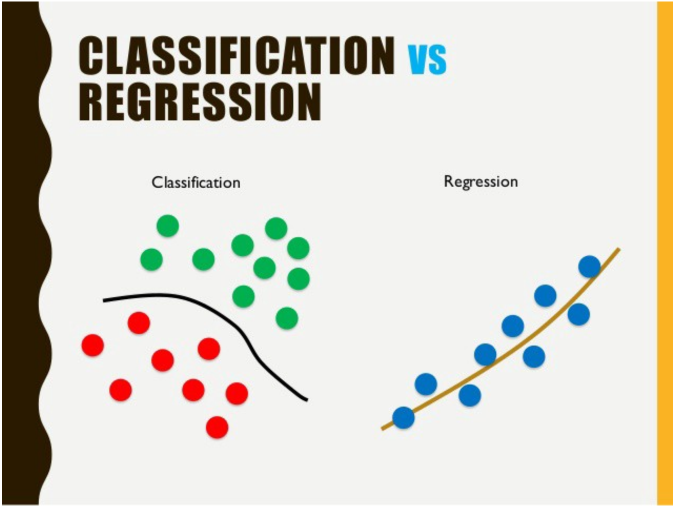


?
Sepal.Width
Sepal.Length
2.5
3.0
3.5
4.0
4.5
5.0
5.5

ML algorithms can be classified into two different groups:

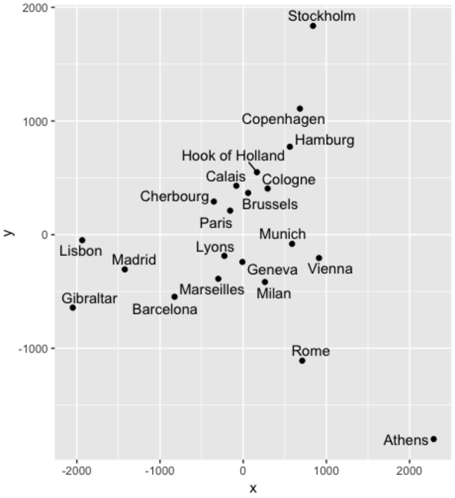
- Supervised learning.
- Unsupervised learning.

Supervised learning is the machine learning task of learning a function that **maps an input to an output**.



Unsupervised learning is the machine learning task that allows us to discover patterns from the data. Rather than prediction a variable (temperature, flower type, etc.) these algorithms are aimed to discover patterns in the data.

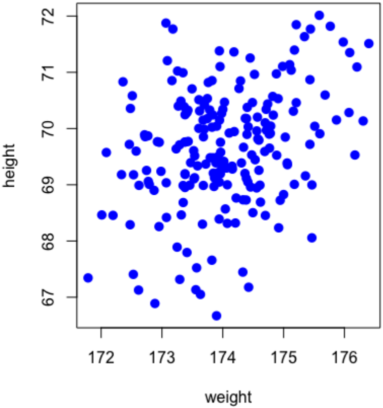
	Athens	Barcelona	Brussels	Calais	Cherbourg	$\cdots$
Athens	0	3313	2963	3175	3339	$\cdots$
Barcelona	3313	0	1318	1326	1294	$\cdots$
Brussels	2963	1318	0	204	583	$\cdots$
Calais	3175	1326	204	0	460	$\cdots$
Cherbourg	3339	1294	583	460	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$


Stockholm
Copenhagen
Hamburg
Hook of Holland
Calais
Cologne
Cherbourg
Brussels
Paris
Munich
Lisbon
Lyons
Madrid
Geneva
Vienna
Marseilles
Milan
Gibraltar
Barcelona
Rome
Athens
2000
1000
0
-1000
-2000
2000
y
x

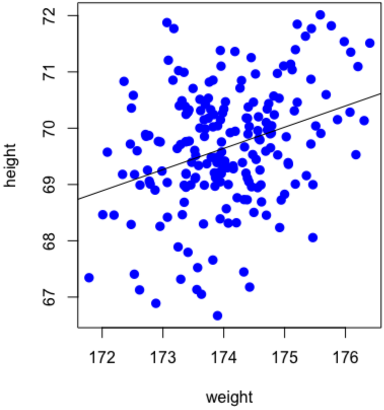
Linear regression

	height	weight
1	173.37	69.76
2	174.18	71.36
3	173.16	70.85
4	175.60	69.91
5	174.33	67.45
6	173.18	71.77
7	174.49	70.46
8	174.74	70.44
9	174.58	69.82
10	173.69	69.99
⋮	⋮	⋮
200	173.61	70.213

Is there any relationship between *height* and *weight*?



- We would like to find two coefficients (w and b) such that $weight = b + w * height$.
- In general, given two pairs of variables x and y, we would like to find two coefficients (w and b) such that $y = b + w * x$.
- w is known as the *weight* and b is known as the *bias*.



How can we find w and b such that $y = b + w * x$ is a "good approximation" to the data points?

More important question than *how* is: **why** do we know to know these parameters?

- If a system is modeled by an equation, it can help to predict what can happen under certain circumstances.

- Let's suppose $b = 4.057$ and $w = 0.3769$. It means

$$y = 4.057 + 0.3769 * x.$$

What would be the *weight* (y) of someone whose height (x) is 174 cm? $y = 4.057 + 0.3769 * 174 = 69.6376$ kg.

Loss function

- Previously, we asked the model to predict the *weight* of a persona whose height is 174 cm.
- The prediction was 69.6376 kg.
- **Notation:** when we use the model to predict, we use a special symbol for the results: $\hat{y}$.
- So, in the previous example: $\hat{y} = 69.6376$

How can we find w and b such that $y = b + w * x$ is a "good approximation" to the data points?

- We need a metric that tells what is "good" and what is "bad".
- We want a metric that is close to 0 when the model is correct.
- We want a metric that increases as long as the model is not correct.

Let's assume we have w and b. For instance, at random we choose $w = 0.2$ and $b = 3$:

	height(x)	weight(y)	prediction($\hat{y}$)	$error = (y - \hat{y})^2$
1	173.37	69.76	37.67	1029.39
2	174.18	71.36	37.84	1123.97
3	173.16	70.85	37.63	1103.53
⋮	⋮	⋮	⋮	⋮
200	173.6189	70.31279	37.72378	1062.0434

$$loss = MSE = \sqrt{\frac{1}{200}(1029.39 + 1123.97 + \cdots + 1062.0434)} = 31.86411$$

- In general, the formula for loss function (for regression problems) is the following one:

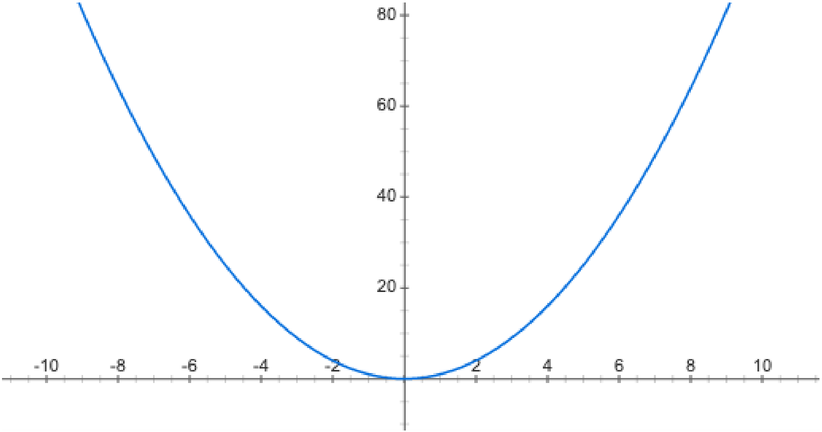
$$loss = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\bigg(y_i - (b + wx_i)\bigg)^2}.$$

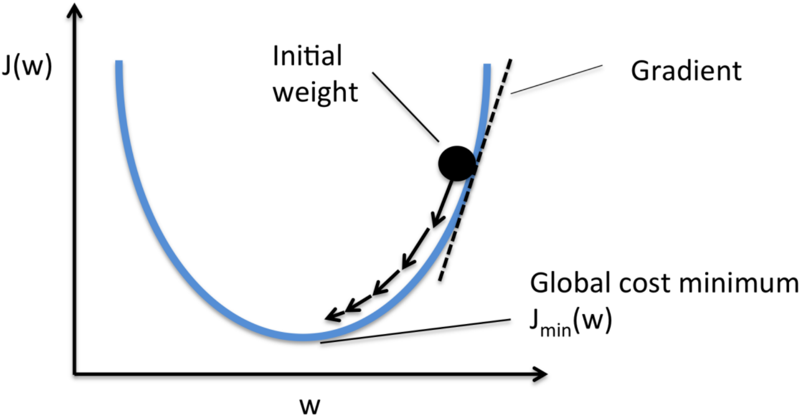
- Our goal is to find b and w such that the loss is minimum.
- How??? Choosing b and w randomly??? We will use **Gradient Descent** algorithm.

Gradient Descent

The idea is the following one:

- Given a function that depends on two parameters $f(b, w)$, we want to find w^* and b^* such that $min\{f(b, w)\} = f(b^*, w^*)$.
- Gradient Descent (GD) allows us to find such parameter.
- GD works with more than two variables, i.e, let's suppose we want to find the minimum value of a function that depends on n variables $f(z_1, z_2, \ldots, z_n)$, GD allows us to find $z_1^*, z_2^*, \ldots, z_n^*$ such that $min\{f(z_1, z_2, \ldots, z_n)\} = f(z_1^*, z_2^*, \ldots, z_n^*)$.

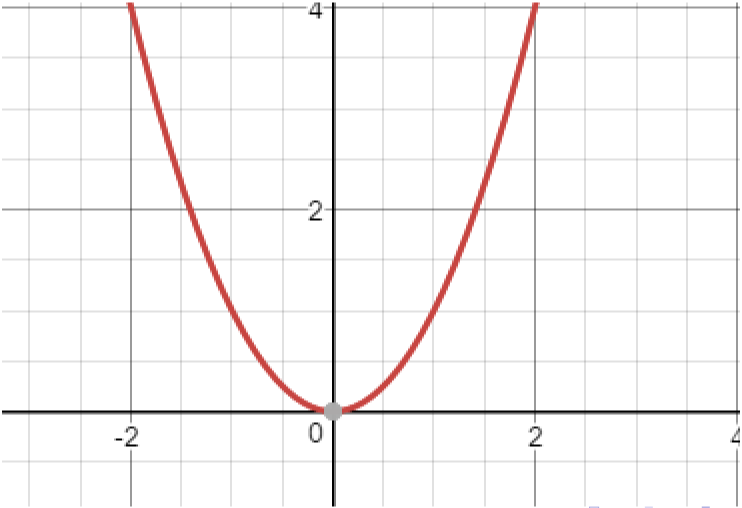

80
60
40
20
-10
-8
-6
-4
-2
2
4
6
8
10


J(w)
Initial weight
Gradient
Global cost minimum
$J_{min}(w)$
w

Algorithm 1: Gradient Descent algorithm

Result: Find w^* such that $min\{f(w)\} = f(w^*)$
$epsilon = 10^{-6}$;
$w_0 =$ (random) initial point;
$is\_minimum = FALSE$;
$learning\_rate = 0.1$;
$f_0 = f(w_0)$;
while *not is_minimum* **do**
    $derivative = f'(w_0)$;
    $w_1 = w_0 - learning\_rate \cdot derivative$;
    $f_1 = f(w_1)$;
    if $\mid f_1 - f_0 \mid < epsilon$ **then**
        $is\_minimum = TRUE$;
    end
    $w_0 = w_1$;
    $f_0 = f_1$;
end
return w_1

Given $f(w) = w^2$ and $w_0 = 1$, we want to apply GD algorithm to obtain the minimum. We expect to obtain, after some iterations, $w^* = 0$ (or close to 0).



Remember that:

- $f(w) = w^2$.
- $derivative = f'(w) = 2w$.
- $w_1 = w_0 - learning\_rate \cdot derivative$.

We choose as learning rate a value of 0.01.

iteration	w_0	$f(w_0)$	derivative	w_1	$f(w_1)$	$\mid f(w_1) - f(w_0) \mid$
2	1.00	1.00	2.00	0.80	0.64	0.36
3	0.80	0.64	1.60	0.64	0.41	0.23
4	0.64	0.41	1.28	0.51	0.26	0.15
⋮	⋮	⋮	⋮	⋮	⋮	⋮
26	0.00	0.00	0.01	0.00	0.00	0.00
27	0.00	0.00	0.01	0.00	0.00	0.00

We want to find the minimum of the following function

$$f(w) = w^4 + w^3 - 3w^2 - 2w + 2.$$

Its derivative function is:

$$f'(w) = 4w^3 + 3w^2 - 6w - 2.$$

We cannot solve manually the following equation:

$$4w^3 + 3w^2 - 6w - 2$$

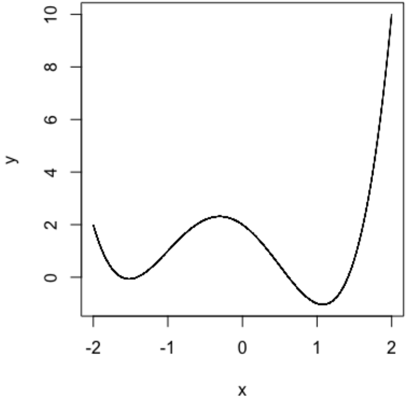
GD can help us to find the minimum value.

Remember that:

- $f(w) = w^4 + w^3 - 3w^2 - 2w + 2$.
- $derivative = f'(w) = 4w^3 + 3w^2 - 6w - 2$.
- $w_1 = w_0 - learning\_rate \cdot derivative$.

We choose as learning rate a value of 0.001.

iteration	w_0	$f(w_0)$	derivative	w_1	$f(w_1)$	$\mid f(w_1) - f(w_0) \mid$
1	2	10	30	1.97	9.124	0.87
2	1.97	9.12	28.40	1.94	8.33	0.78
3	1.94	8.33	26.93	1.91	7.63	0.70
4	1.91	7.63	25.58	1.88	6.99	0.63
5	1.88	6.99	24.33	1.86	6.41	0.57
6	1.86	6.41	23.17	1.84	5.88	0.52
7	1.84	5.88	22.10	1.81	5.41	0.47
8	1.81	5.41	21.10	1.79	4.97	0.43
⋮	⋮	⋮	⋮	⋮	⋮	⋮
367	1.07	-1.03	0.03	1.07	-1.03	0
368	1.078	-1.03	0.03	1.07	-1.03	0


y
10
8
6
4
2
0
-2
-1
0
1
2
x

Back to our initial problem, we want w and b such that minimize

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(weight - (b + w \cdot height)\right)^2}.$$

- Applying GD we obtain $b = 4.0570$ and $w = 0.3769$.
- Loss function is 1.00795.
- Any other set of parameters would provide a loss function greater than 1.00795.

- GD is an algorithm that helps us find the optimal values for the parameters. That is why it is called *optimizer*.
- There are many *optimizers* algorithms:
  - Momentum
  - RMSprop
  - Adam
  - AdaMax
  - Adadelta
  - ...

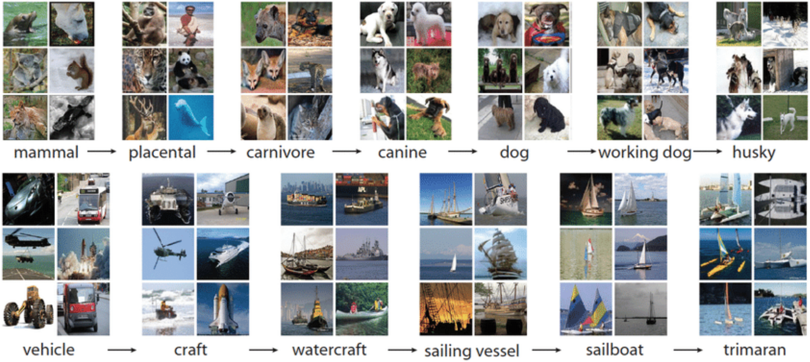
Introduction

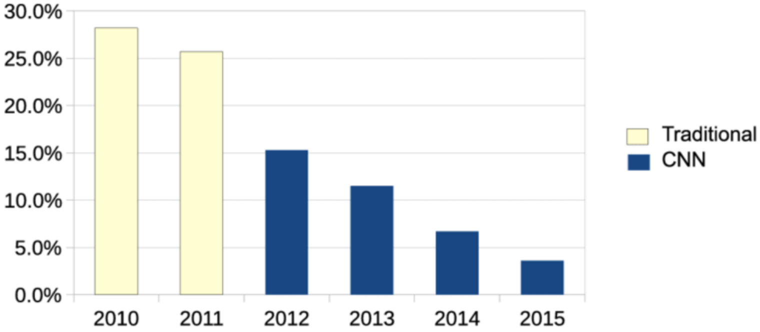
Why do we need *another* kind of models?

- Unfortunately real life is not linear. We need more *complex*/*flexible* models.
- Neural networks (Deep Learning models) are very flexible models. They are able to capture very non-linear patterns and model them with a high precision.

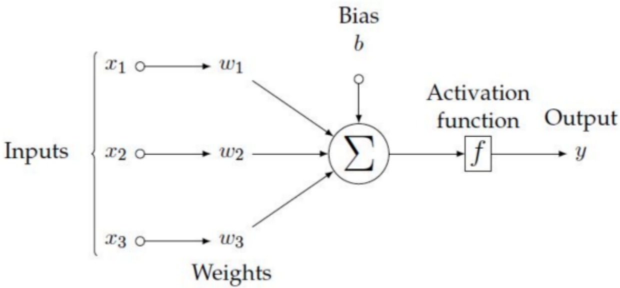
ImageNet problem

- The ImageNet project is a large visual database designed for use in visual object recognition software research.
- More than 14 million images have been hand-annotated.
- ImageNet contains more than 20.000 categories with a typical category, such as "balloon" or "strawberry", consisting of several hundred images.


mammal
placental
carnivore
canine
dog
working dog
husky
vehicle
craft
watercraft
sailing vessel
sailboat
trimaran

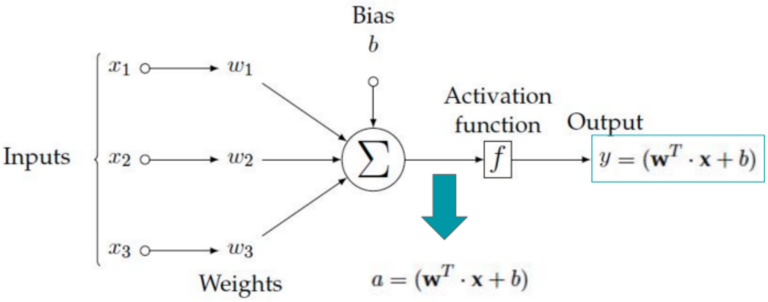

30.0%
25.0%
20.0%
15.0%
10.0%
5.0%
0.0%
2010
2011
2012
2013
2014
2015
Traditional
CNN

Single neuron model (perceptron)



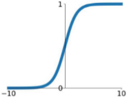
- **Weights** and **bias** are the parameters that define the behavior. They must be estimated during training.
- The **output** (y) is derived from a sum of the weighted inputs plus a bias term.
- The **activation** function **introduces non-linearities**.

Single neuron model: Linear Regression

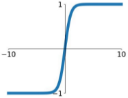


Activation functions

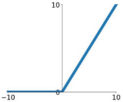
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



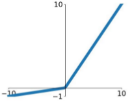
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

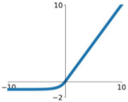


Leaky ReLU
$\max(0.1x, x)$

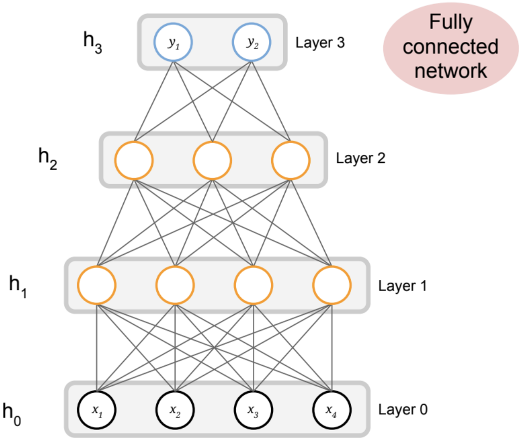


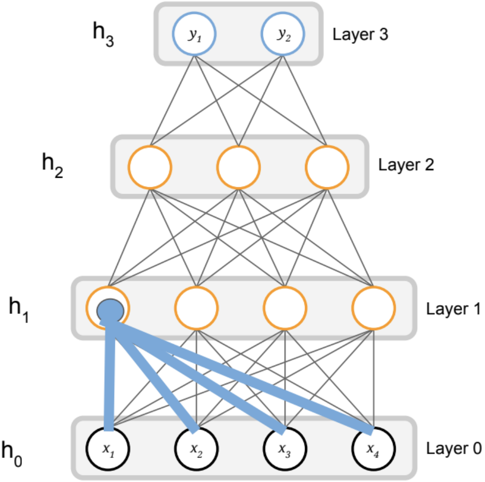
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

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



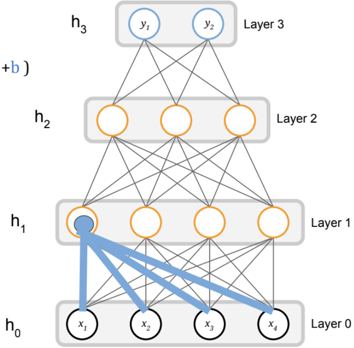
Multi-layer Perceptron




h3
y1
y2
Layer 3
h2
Layer 2
h1
Layer 1
h0
x1
x2
x3
x4
Layer 0

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \quad h_0 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad b_1 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$h_{11} = g(\, \mathbf{wx} + b \,)$



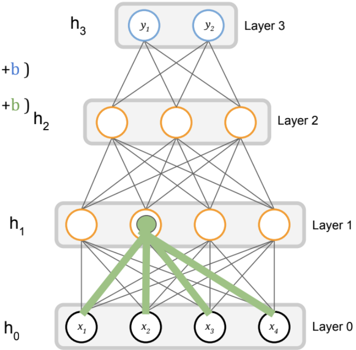
Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$
$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

$$W_1 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \quad h_0 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \quad b_1 = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

$h_{11} = g(\ \mathbf{wx} + b\ )$

$h_{12} = g(\ \mathbf{wx} + b\ )$



Forward pass computes

$$\mathbf{h}_0 = \mathbf{x}$$
$$\mathbf{h}^{(t)} = g(W^{(t)}\mathbf{h}^{(t-1)} + \mathbf{b}^{(t)})$$

Forward pass

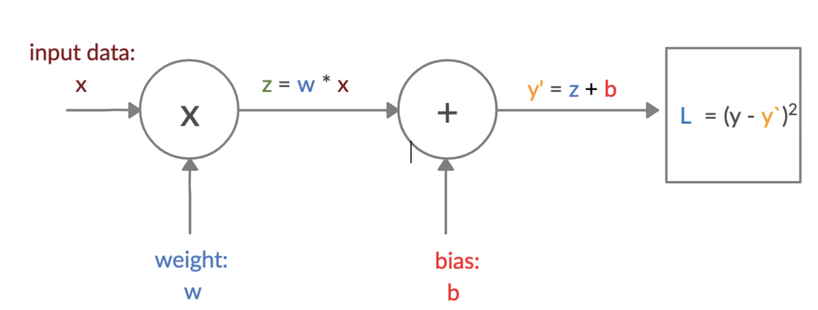
We are going to use the following notation for derivatives:

$$\frac{\partial L}{\partial w}, \frac{\partial L}{\partial b}$$

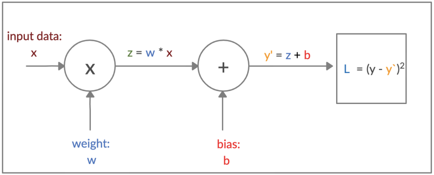
Remember that GD formula is:

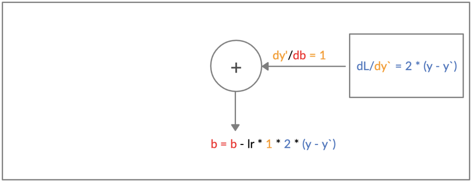
$$w = w - learning\_rate \cdot \frac{\partial L}{\partial w},$$

$$b = b - learning\_rate \cdot \frac{\partial L}{\partial b}.$$


input data:
x
X
z = w * x
+
y' = z + b
L = (y - y`)²
weight:
w
bias:
b

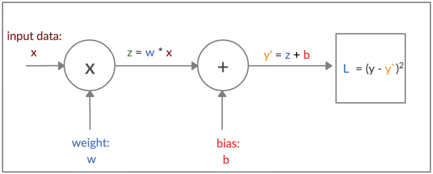
Backpropagation: Linear case

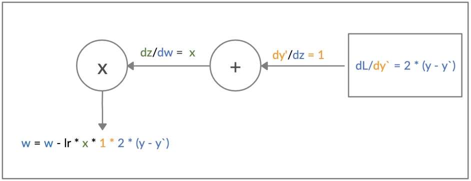




$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y'} \cdot \frac{\partial y'}{\partial b}.$$

Backpropagation: Linear case

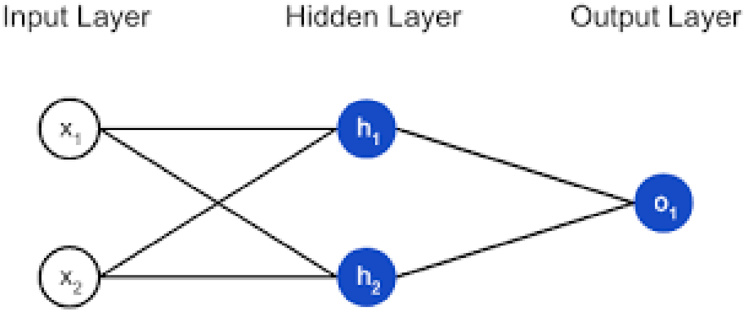




$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y'} \cdot \frac{\partial y'}{\partial z} \cdot \frac{\partial z}{\partial w}.$$

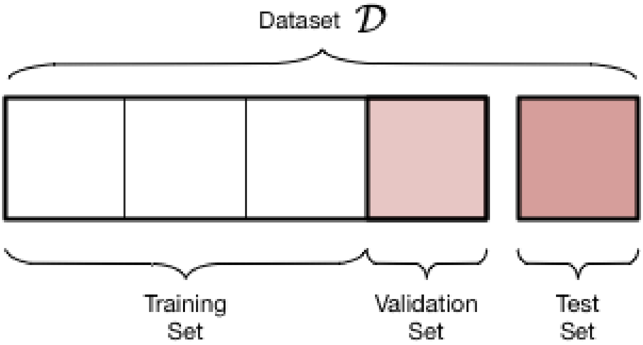
How a Neural Net is trained?

The first step is to decide the structure of the Neural Net. To begin with, it should be an easy Net (just to have a first quick model).



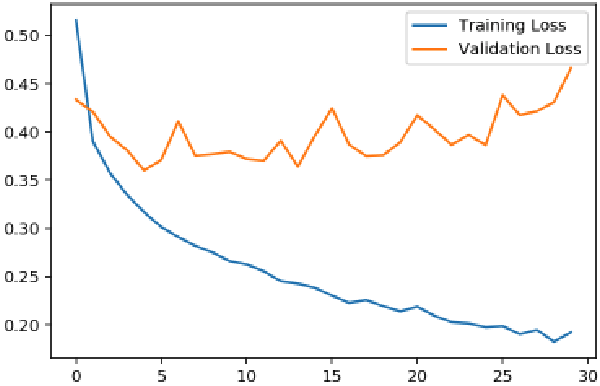
Given a dataset (big dataset) we split the data set into three part:

- Training part.
- Validation part.
- Test part.



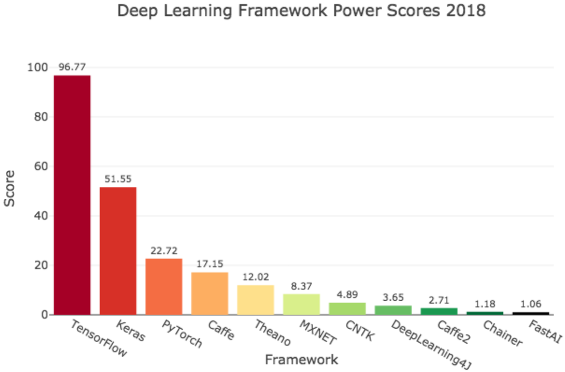
Split the training dataset into n parts (*epoch*).

- For every epoch train the model using the training part.
- Compute the loss metric (and more metrics if you want) in this epoch.
- Compute the loss metric (and more metrics if you want) in the validation part.


Training Loss
Validation Loss
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0
5
10
15
20
25
30

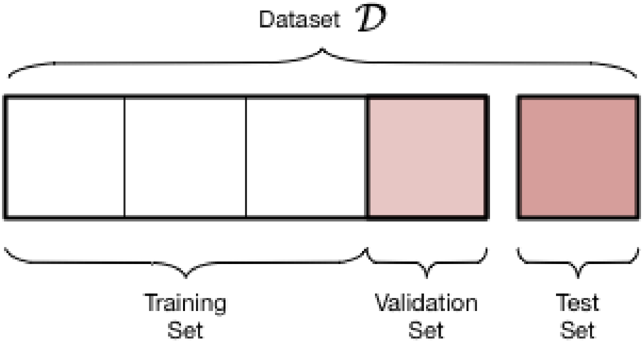
Finally, compute the loss over the test dataset and compare it with the loss of the training dataset and the validation dataset.

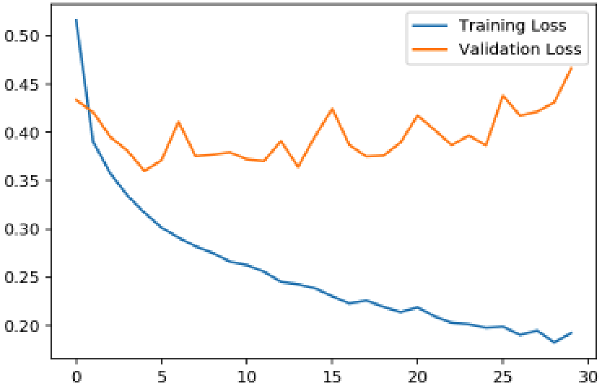
- All Deep Learning Frameworks come with an efficient implementation of the previous steps (especially the training loop).
- The most popular ones are *TensorFlow (Google)*, *Keras ("Google")* and *PyTorch (Facebook)*.



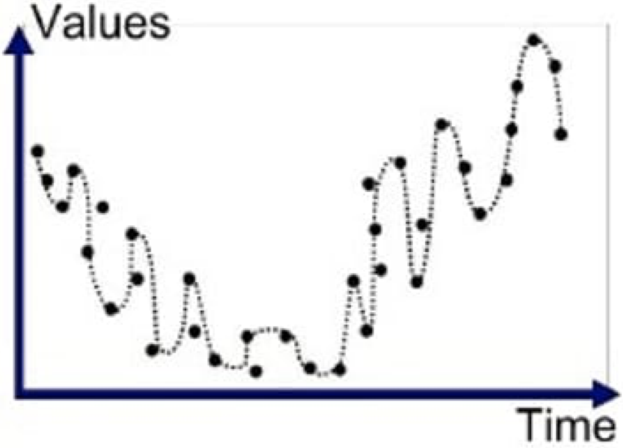
Overfitting

If we need as much data as possible to train a model, why do have to train with one part of the dataset instead of the whole dataset?




Training Loss
Validation Loss
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0
5
10
15
20
25
30

Overfitting is a modeling error which occurs when a function is too closely fit to a limited set of data points.



The way of fighting overfitting is using what is called *regularization* techniques. It consists in adding a penalty in the loss function:

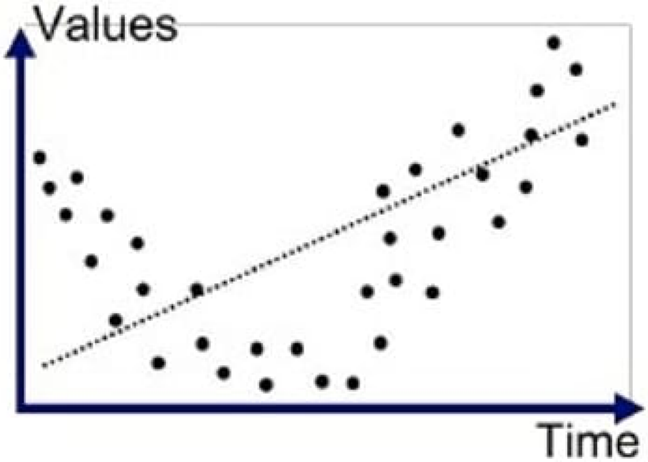
$$loss = \left(y - (b + wx)\right)^2,$$

$$loss_{reg} = loss + \lambda(b^2 + w^2),$$

where λ is a value that we have to choose a priori. Normally $\lambda > 0.001$ and $\lambda < 1$.

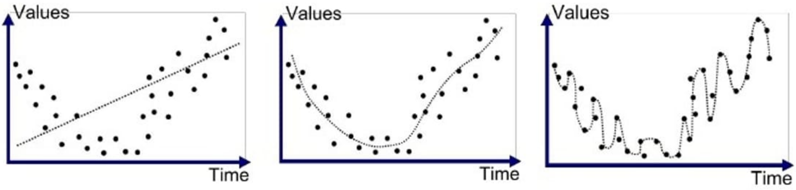
Underfitting

When the model is too easy that is not able to learn important patters from them data, it is said that the model is *underfitted*.

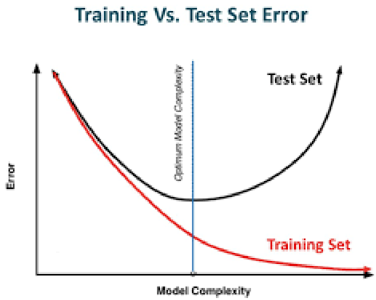


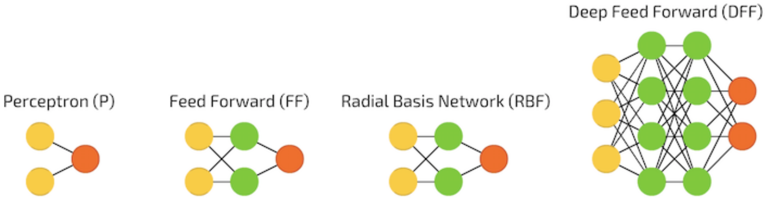
How to find a "good model"?

- Start with a really simple model.
- Check it is underfitted.
- Increase the model complexity until it gets overfitted.



Underfitted Good Fit/Robust Overfitted




Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)

Thank You