



## 1 Introduction

## 2 Linear regression

### 3 Ingredients for Deep Learning

## Loss function

## Gradient Descent

## 4 Deep Learning

## Introduction

## Perceptron

## Multi-layer Perceptron

## Backpropagation

## How a Neural Net is trained?

## Overfitting

## 1 Introduction

## Loss function

## Gradient Descent

## Introduction

## Perceptron

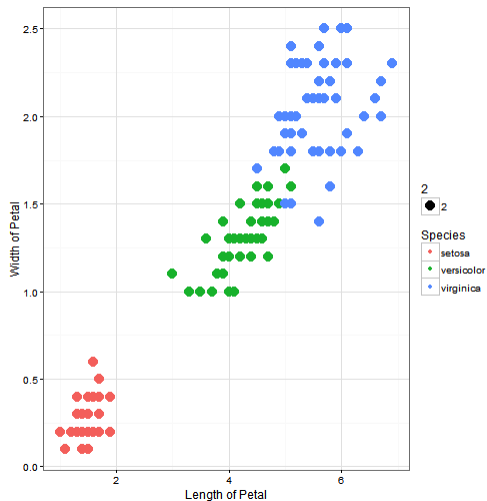
## Multi-layer Perceptron

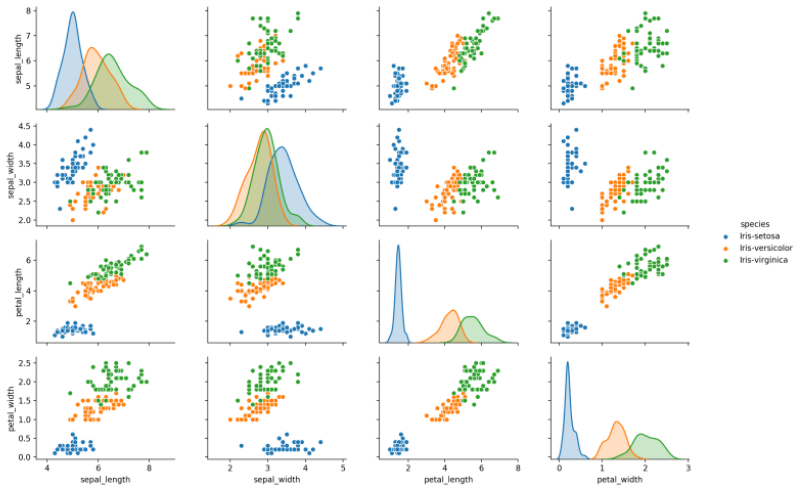
## Backpropagation

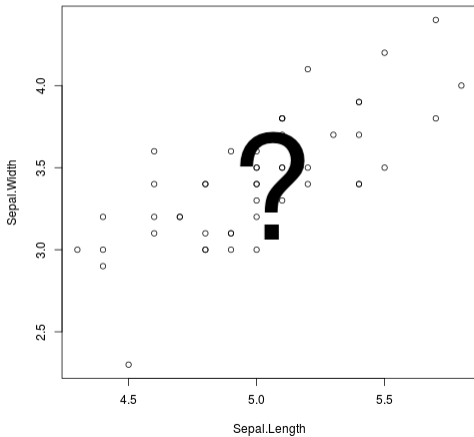
## How a Neural Net is trained?

## Overfitting





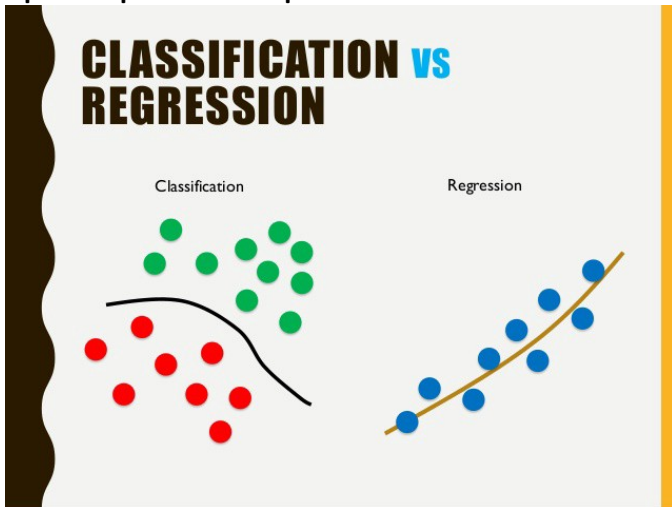








Supervised learning is the machine learning task of learning a function that **maps an input to an output**.





	Athens	Barcelona	Brussels	Calais	Cherbourg	...
Athens	0	3313	2963	3175	3339	...
Barcelona	3313	0	1318	1326	1294	...
Brussels	2963	1318	0	204	583	...
Calais	3175	1326	204	0	460	...
Cherbourg	3339	1294	583	460	0	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮



## 2 Linear regression

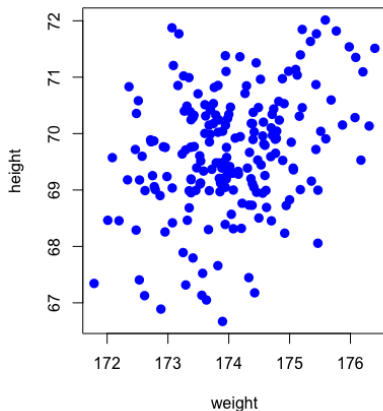
### 3 Ingredients for Deep Learning

- Loss function
- Gradient Descent

- 4 Deep Learning
  - Introduction
  - Perceptron
  - Multi-layer Perceptron
  - Backpropagation
  - How a Neural Net is trained?
  - Overfitting

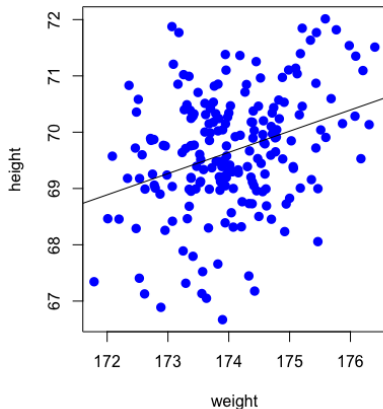


Is there any relationship between *height* and *weight*?



- 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?





### 3 Ingredients for Deep Learning

Loss function  
Gradient Descent

- Introduction
- Perceptron
- Multi-layer Perceptron
- Backpropagation
- How a Neural Net is trained?
- Overfitting





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
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
200	173.6189	70.31279	37.72378	1062.0434

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





### ③ Ingredients for Deep Learning

## Loss function

## Gradient Descent

## Introduction

## Perceptron

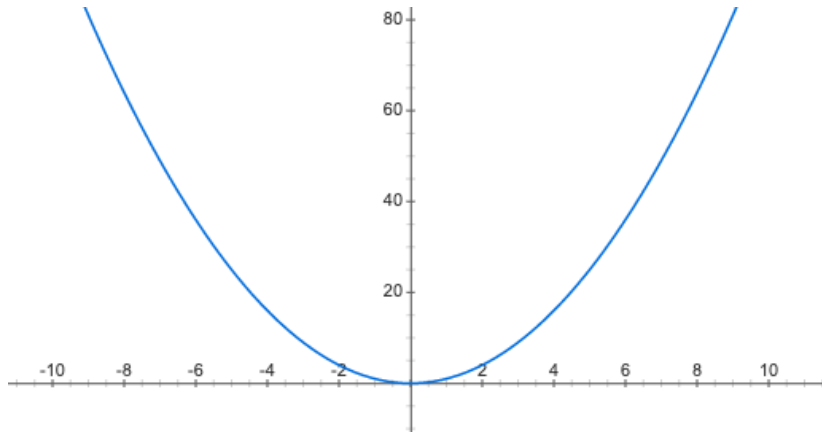
## Multi-layer Perceptron

## Backpropagation

## How a Neural Net is trained?

## Overfitting









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	$w_1$	$f(w_1)$	$ f(w_1) - f(w_0) $
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
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
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



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
  - Adadelat
  - ...

### 3 Ingredients for Deep Learning

- Loss function
- Gradient Descent

## 4 Deep Learning

- Introduction
- Perceptron
- Multi-layer Perceptron
- Backpropagation
- How a Neural Net is trained?
- Overfitting

## Loss function

## Gradient Descent

## 4 Deep Learning

## Introduction

## Perceptron

## Multi-layer Perceptron

## Backpropagation

## How a Neural Net is trained?

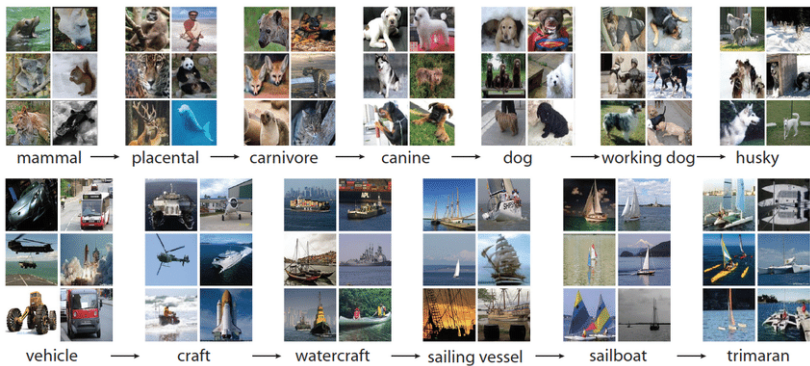
## Overfitting

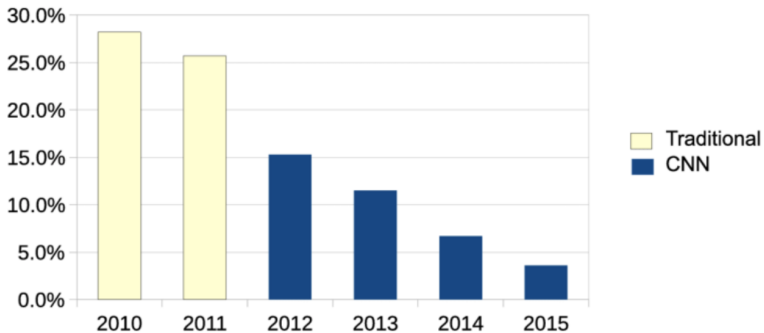
# Introduction

## 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.







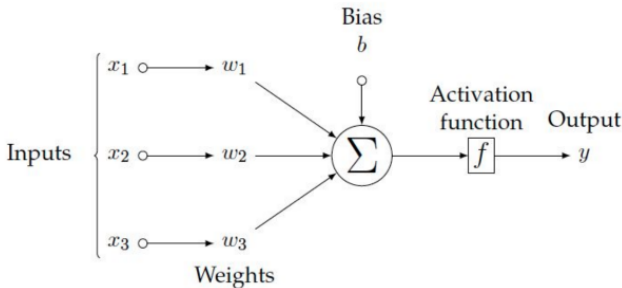


### 3 Ingredients for Deep Learning

- Loss function
- Gradient Descent

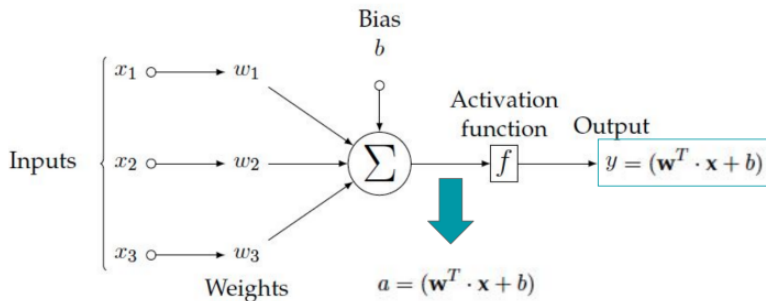
- 4 Deep Learning
  - Introduction
  - Perceptron**
  - Multi-layer Perceptron
  - Backpropagation
  - How a Neural Net is trained?
  - Overfitting

## 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**.

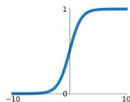
## Single neuron model: Linear Regression



## Activation functions

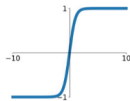
## Sigmoid

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



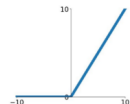
**tanh**

$\tanh(x)$



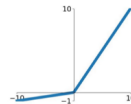
## ReLU

$\max(0, x)$



## Leaky ReLU

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

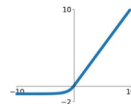


## Maxout

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

# ELU

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



## Loss function

## Gradient Descent

## 4 Deep Learning

## Introduction

## Perceptron

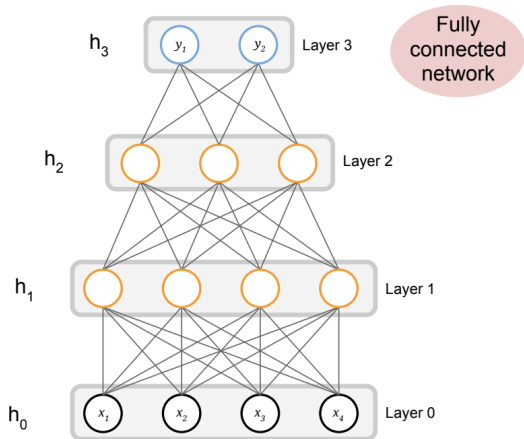
## Multi-layer Perceptron

## Backpropagation

## How a Neural Net is trained?

## Overfitting

# Multi-layer Perceptron



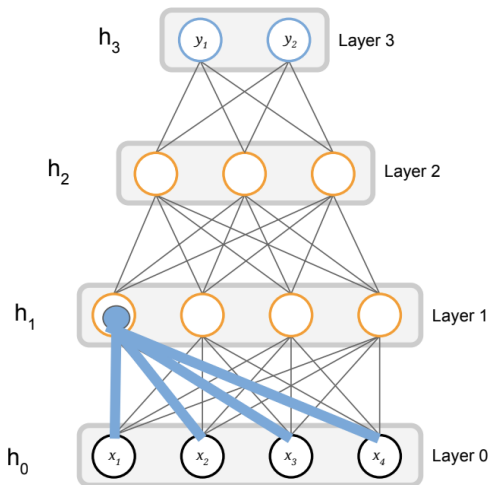


Diagram illustrating the structure of the weight matrix  $W_1$  and the input vector  $h_0$  and bias vector  $b_1$ .

$W_1$  is a 4x4 matrix with elements  $w_{11}, w_{12}, w_{13}, w_{14}$  in the first row,  $w_{21}, w_{22}, w_{23}, w_{24}$  in the second row,  $w_{31}, w_{32}, w_{33}, w_{34}$  in the third row, and  $w_{41}, w_{42}, w_{43}, w_{44}$  in the fourth row.

$h_0$  is a 4x1 vector with elements  $x_1, x_2, x_3, x_4$ .

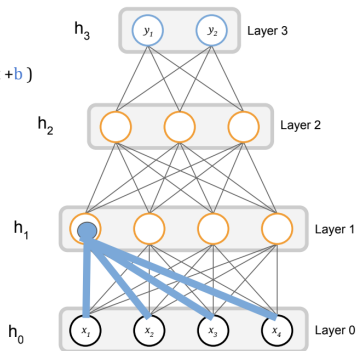
$b_1$  is a 4x1 vector with elements  $b_1, b_2, b_3, b_4$ .

**Forward pass** computes

$$\mathbf{h}_0 = \mathbf{x}$$

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

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





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

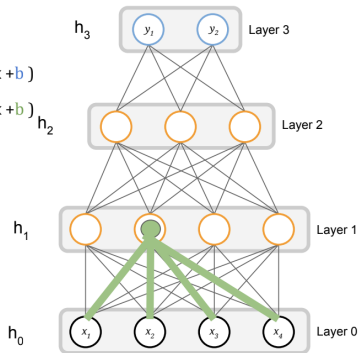
**Forward pass** computes

$$h_0 = x$$

$$h^{(t)} = g(W^{(t)}h^{(t-1)} + b^{(t)})$$

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

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



## Loss function

## Gradient Descent

## ④ Deep Learning

## Introduction

## Perceptron

## Multi-layer Perceptron

## Backpropagation

## How a Neural Net is trained?

## Overfitting

## Forward pass

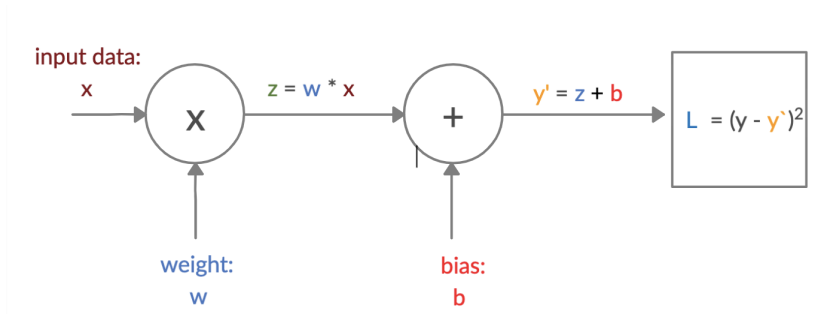
We are going to use the following notation for derivatives:

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

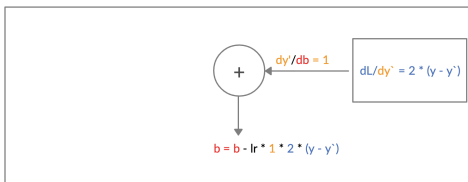
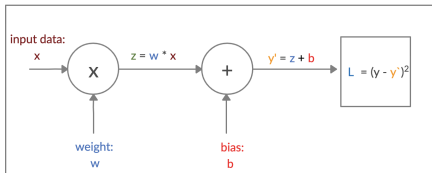
Remember that GD formula is:

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

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

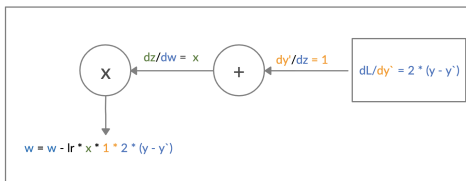
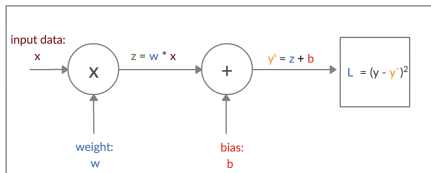


# Backpropagation: Linear case



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

## Backpropagation: Linear case



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

## 1 Introduction

## 2 Linear regression

## 3 Ingredients for Deep Learning

Loss function

Gradient Descent

## 4 Deep Learning

Introduction

Perceptron

Multi-layer Perceptron

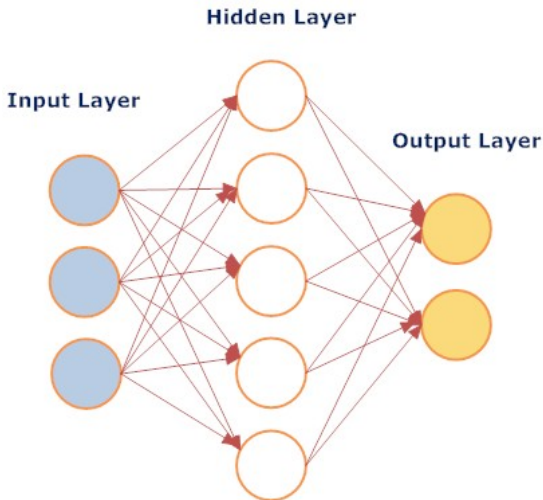
Backpropagation

**How a Neural Net is trained?**

Overfitting

# How a Neural Net is trained?

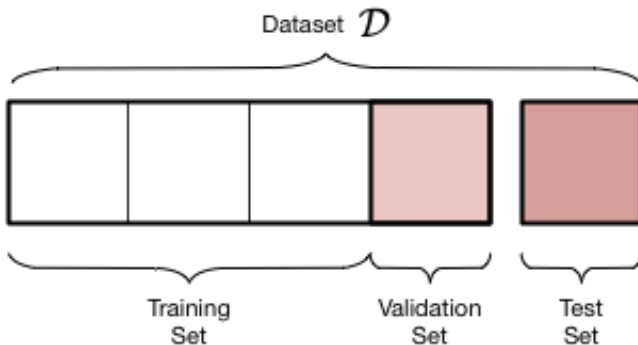
The first step is to decide the structure of the Neural net. To begin with, it should be an easy Net (just to have a first quick model).





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

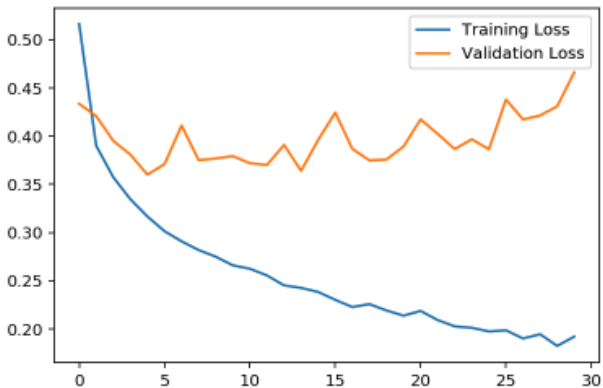
- Training part.
- Validation part.
- Test part.



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.

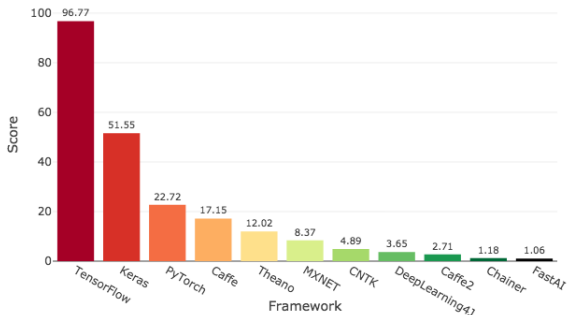
## How a Neural Net is trained?



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).

Deep Learning Framework Power Scores 2018



### 3 Ingredients for Deep Learning

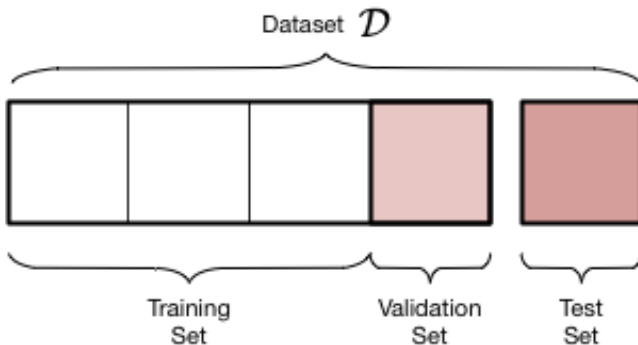
- Loss function
- Gradient Descent

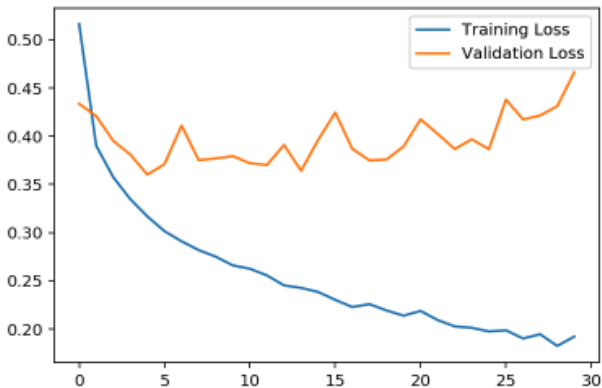
## 4 Deep Learning

- Introduction
- Perceptron
- Multi-layer Perceptron
- Backpropagation
- How a Neural Net is trained?
- Overfitting**

# Overfitting

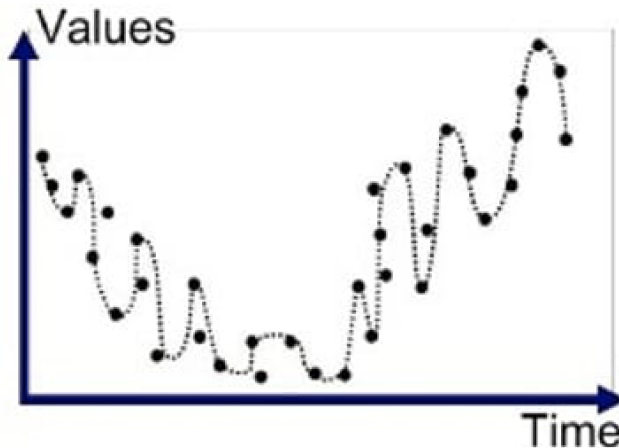
If we need as much data as possible to train a model, why do we have to train with one part of the dataset instead of the whole dataset?







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

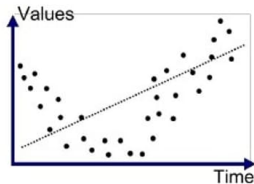




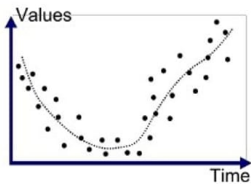


## How to find a "good model"?

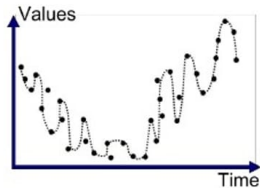
- Start with a really simple model.
- Check it is underfitted.
- Increase the model complexity until it gets overfitted.



## Underfitted



Good Fit/Robust



## Overfitted

## Training Vs. Test Set Error



Perceptron (P)



### Feed Forward (FF)



### Radial Basis Network (RBF)



### Deep Feed Forward (DFF)



# Thank You