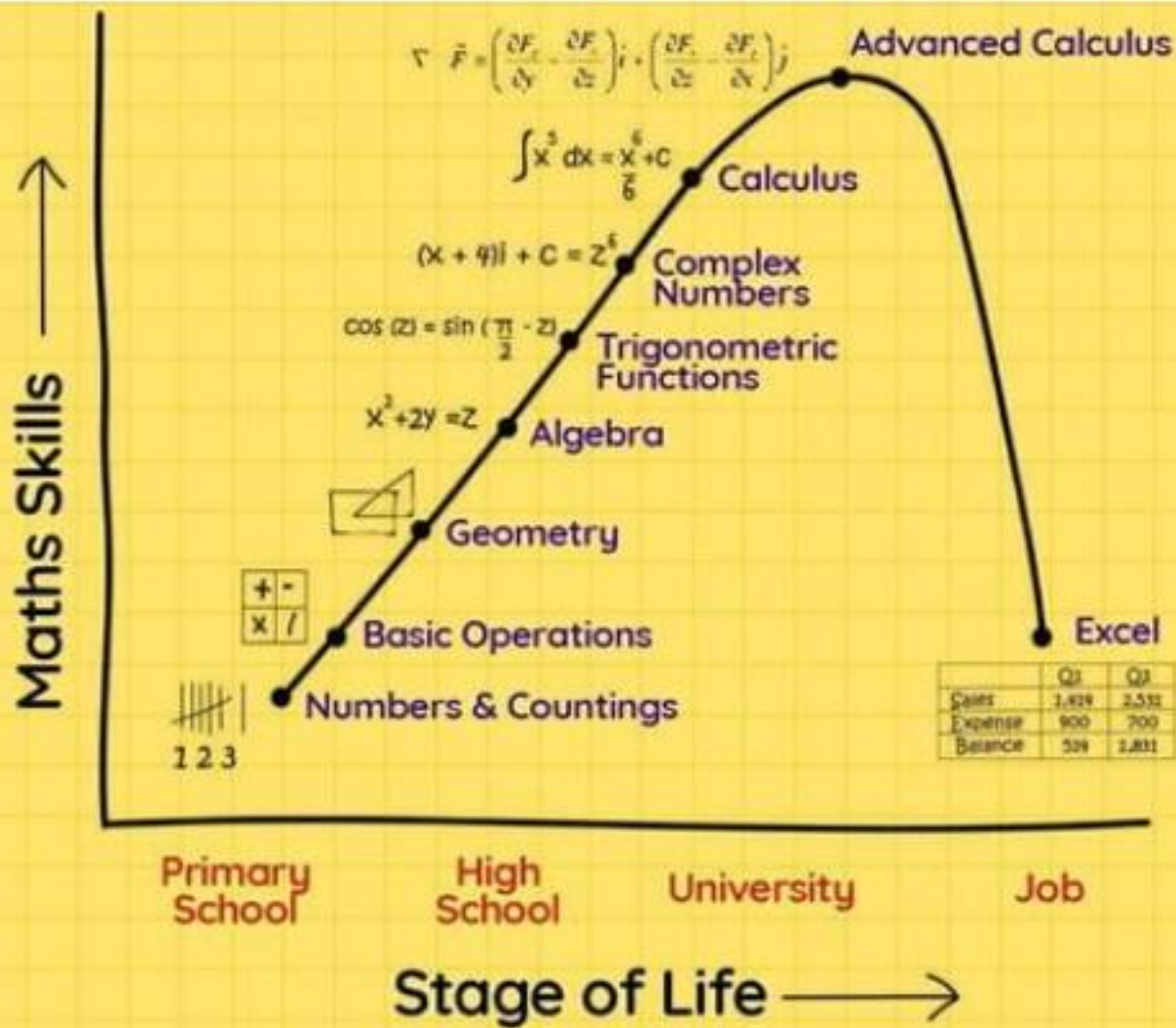


Computer Vision with Machine Learning

Manuel Boissenin, Samuel Rochette



- About me
- Name, age, specificity of your learning route
- What do I want to do after graduation
- What do I expect from this course
- What kind of project did you work on your internship?
- Are you willing to continue on the same branch?



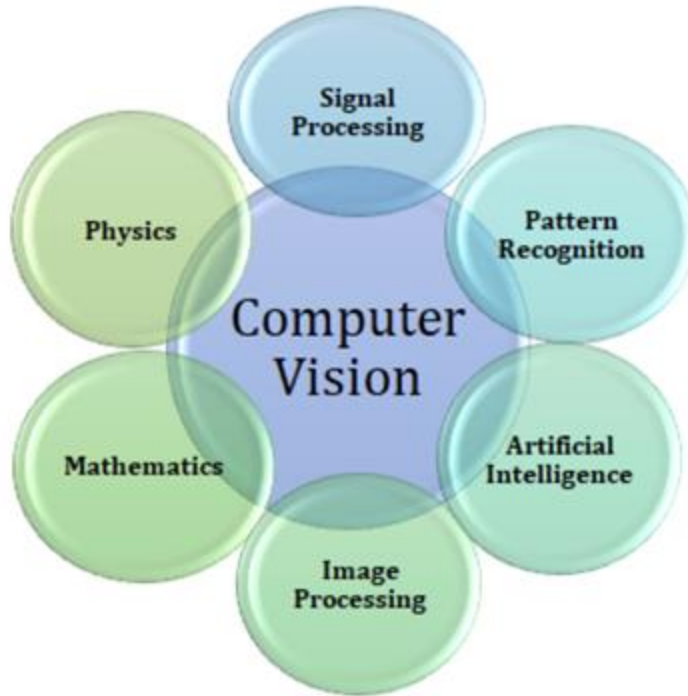
Objectives for this module

1. Understand and explain the specificity of CV when using ML tools
2. Give you a grasp (an insight) of the depth and width of the field
3. Solve and implement a CV problem with a ML solution
4. Choose the best architecture / algo to solve a CV problem with ML
5. We will more specifically focus on Convolutional Neural Networks (CNN)
6. We will also have a quick look at a detection algorithm derived from this architecture
7. Learn to learn



What is Computer Vision

Definition



Computer vision is the science of teaching a computer how to interpret images. The goal is to bridge the gap between pixels on the screen and the meaning behind them.

CV is composed of lots of different tasks:
<https://paperswithcode.com/area/computer-vision>

Applications

- Image segmentation (medical imaging)
- Stereovision, ToF, Human Machine Interfaces
- Video surveillance (airport, roads)
- Factory automation (pills, bottle)
- Docking the space shuttle to the ISS
- Automatic fruit recollection with a robot

Applications

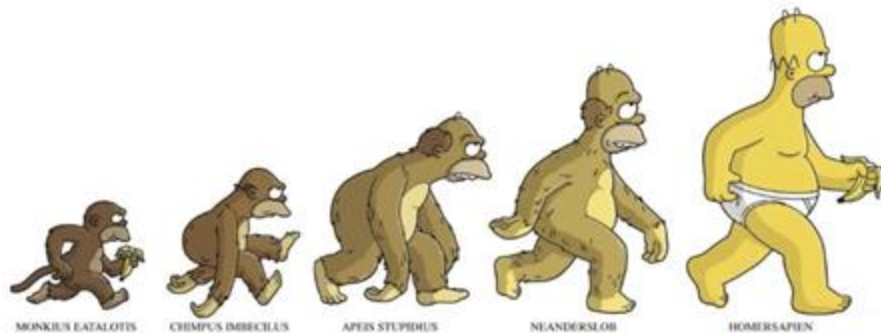
- Recognition of plants
- Funny things: filters on instagram
- Special effects, deep fake
- Movie colorization, super-resolution, compression
- Alzheimer evolution and characterisation on MRI

Summary

1. History

2. Classic CV
3. ML Basics
4. From a neuron to a neural net
5. Backpropagation
6. Activation functions
7. Batch normalization
8. DNN, zoo

History

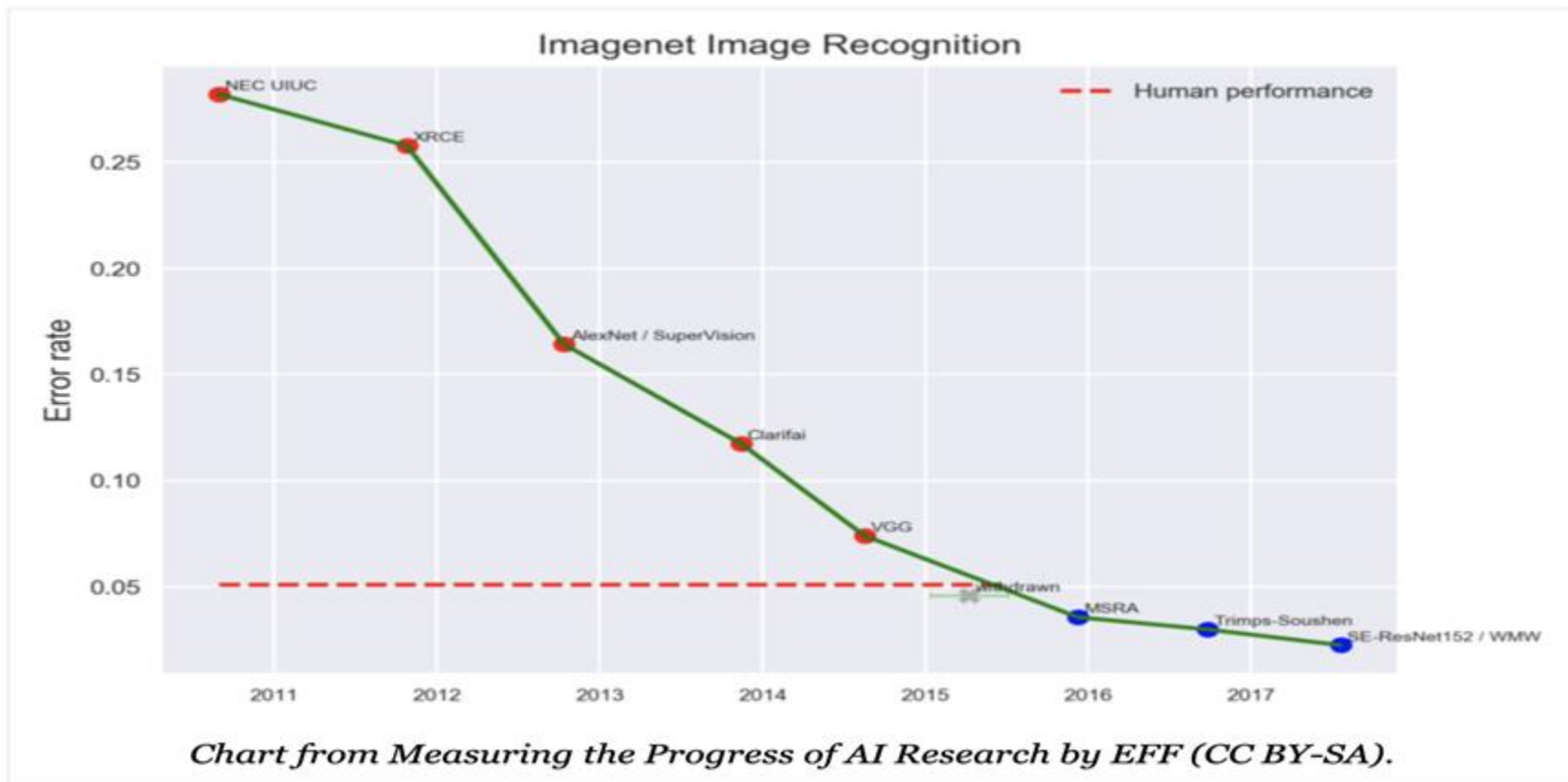


HOMERSAPIEN

History

- Historical roots of computer vision began in the 1950s - closely intertwined with the study of human vision
- To teach computers how to be able to "see," researchers have sought inspiration from visual perception in humans.
- Classification problem
- 2012, the revival of neural networks, Imagenet (Stanford)
- 2015, resnet
- 2017 Vaswani (Transformers)
- [Some milestones](#)

History



History

- CV with ML has become very popular for 2 reasons:
 - Really good results
 - Pretty easy to use and understand compared to classic CV
- 3 main factors are used to explain the advent of CNNs and deep learning in general:
 - Availability of data (Imagenet)
 - Improvements of algorithms, solving issues of vanishing gradient, easy to use libraries (TensorFlow, Pytorch)
 - The rise and generalisation of GPU
- The main drawbacks:
 - Can be difficult to interpret (“black box” models)
 - Often need lots of data to train on

Summary

1. History
- 2. Classic CV**
3. ML Basics
4. From a neuron to a neural net
5. Backpropagation
6. Activation functions
7. Batch normalization
8. DNN, zoo

Classical CV



Classical CV is –hard- (can be complex)

- I said previously that “CVwML is easy to use and understand compared to classic CV”. You’d understand this better when you implement algo from both from scratch.
- Here is an example of the really famous C++ library “open-cv” documentation (Homography is a transformation that maps the points in one image to the corresponding points in the other image - e.g. to remove perspective):

https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html

- We saw in a previous slide that it is linked to pattern recognition & signal processing. Those are open research maths field (some fields medal worked on image denoising).

Classical CV, image

- What is an image? How is it defined? What range?
- What is a filter? Convolution?

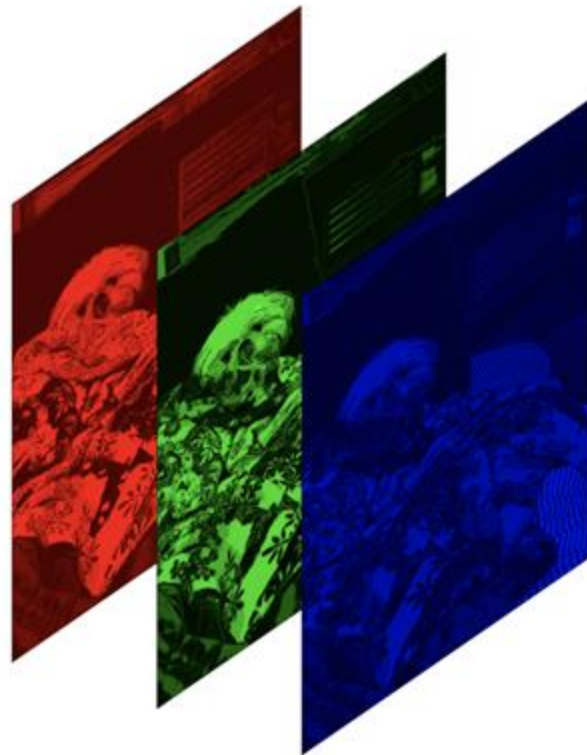
Classical CV, Grey Image



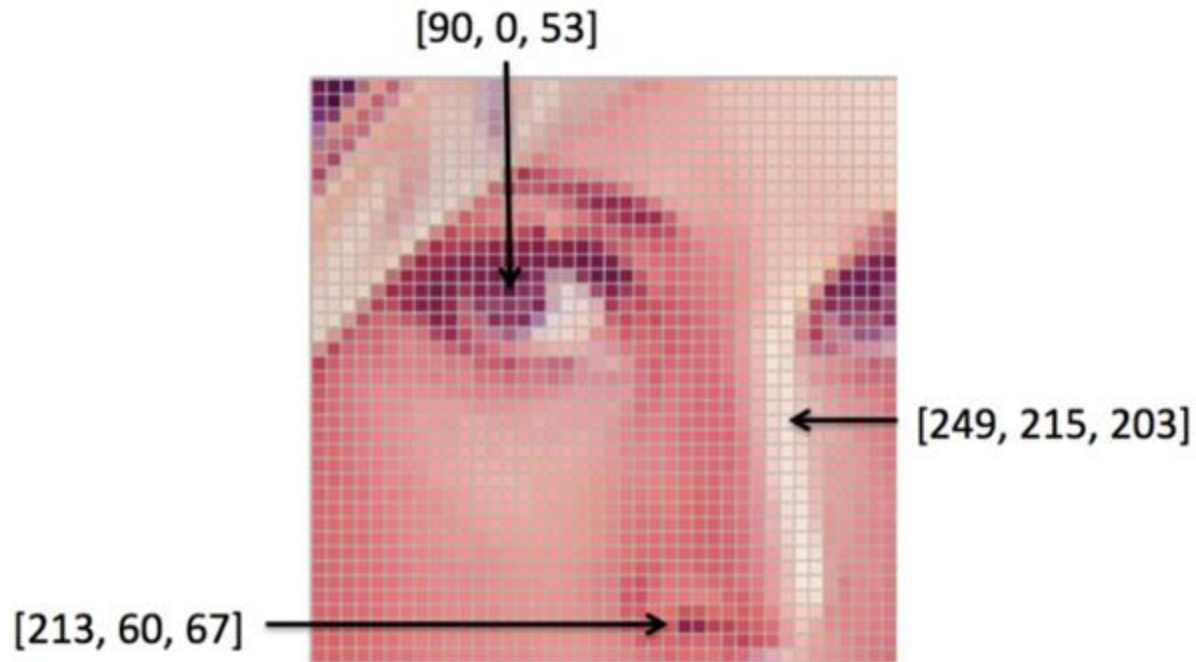
157	153	174	168	150	152	129	151	172	161	165	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	165	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

157	153	174	168	150	152	129	151	172	161	165	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Classic CV, Color Image



Classic CV, Color Image



Classic CV, Operation



$$g(x,y) = f(x,y) + 20$$



$$g(x,y) = f(-x,y)$$

Classic CV, Segmentation

$$g[n, m] = \begin{cases} 255, & f[n, m] > 100 \\ 0, & \text{otherwise.} \end{cases}$$



Classic CV, Noise Reduction

- Given a camera and a still scene, how can you reduce noise of a picture?

Classic CV, Noise Reduction

- Given a single image?

Modify the pixels in an image based on some function of a local neighborhood of each pixel

10	5	3
4	5	1
1	1	7

Local image data

Some function



	7	

Modified image data

Classic CV, Average Filter

$$F[x, y]$$

[illegible]

$$G[x, y]$$

A 10x10 grid with a red box around the top-left cell containing the number 0.

Classic CV, Average Filter

$$F[x, y]$$

[illegible]

$$G[x, y]$$

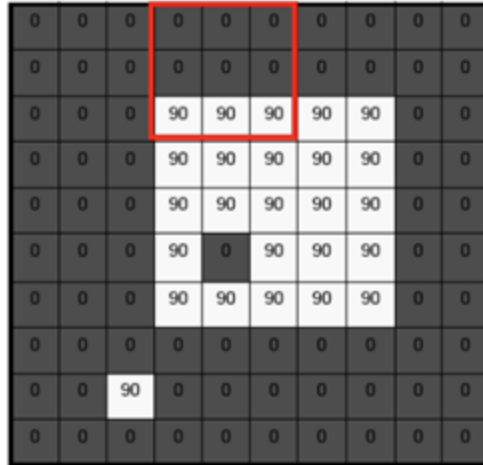
[illegible]

Classic CV, Average Filter

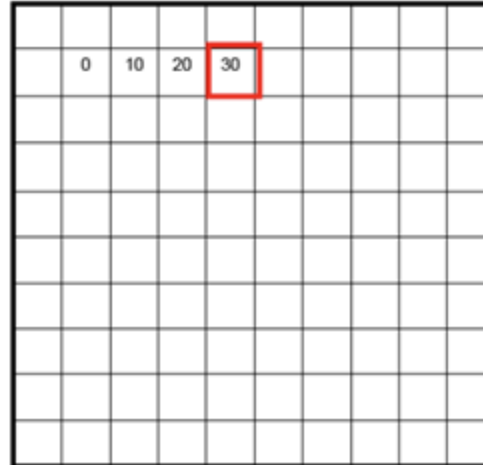
$$F[x, y]$$
[illegible]
$$G[x, y]$$
[illegible]

Classic CV, Average Filter

$$F[x, y]$$



$$G[x, y]$$



Classic CV, Average Filter

$$F[x, y]$$

[illegible]

$$G[x, y]$$

[illegible]

Classical CV, Convolution

Convolution is a mathematical operation which is fundamental to many common image processing operators:

- provides a way of 'multiplying together' two arrays of numbers, to produce a third array of numbers of the same dimensionality with some quirks
- can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values

In an image processing context, one of the input arrays is normally just a gray level image. The second array is usually much smaller, also two-dimensional and is known as the kernel.

Classical CV, Convolution

\mathbf{I}_{11}	\mathbf{I}_{12}	\mathbf{I}_{13}	\mathbf{I}_{14}	\mathbf{I}_{15}	\mathbf{I}_{16}	\mathbf{I}_{17}	\mathbf{I}_{18}	\mathbf{I}_{19}
\mathbf{I}_{21}	\mathbf{I}_{22}	\mathbf{I}_{23}	\mathbf{I}_{24}	\mathbf{I}_{25}	\mathbf{I}_{26}	\mathbf{I}_{27}	\mathbf{I}_{28}	\mathbf{I}_{29}
\mathbf{I}_{31}	\mathbf{I}_{32}	\mathbf{I}_{33}	\mathbf{I}_{34}	\mathbf{I}_{35}	\mathbf{I}_{36}	\mathbf{I}_{37}	\mathbf{I}_{38}	\mathbf{I}_{39}
\mathbf{I}_{41}	\mathbf{I}_{42}	\mathbf{I}_{43}	\mathbf{I}_{44}	\mathbf{I}_{45}	\mathbf{I}_{46}	\mathbf{I}_{47}	\mathbf{I}_{48}	\mathbf{I}_{49}
\mathbf{I}_{51}	\mathbf{I}_{52}	\mathbf{I}_{53}	\mathbf{I}_{54}	\mathbf{I}_{55}	\mathbf{I}_{56}	\mathbf{I}_{57}	\mathbf{I}_{58}	\mathbf{I}_{59}
\mathbf{I}_{61}	\mathbf{I}_{62}	\mathbf{I}_{63}	\mathbf{I}_{64}	\mathbf{I}_{65}	\mathbf{I}_{66}	\mathbf{I}_{67}	\mathbf{I}_{68}	\mathbf{I}_{69}

\mathbf{K}_{11}	\mathbf{K}_{12}	\mathbf{K}_{13}
\mathbf{K}_{21}	\mathbf{K}_{22}	\mathbf{K}_{23}

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1) K(k, l)$$

Classical CV, Convolution

Questions:

1. Compute $O(5, 7)$

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1) K(k, l)$$

1. The average filter is a 3x3 matrix convolution: which matrix?



Classical CV, Convolution

Questions:

1. Compute $O(5, 7)$

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1)K(k, l)$$

1. The average filter is a 3x3 matrix convolution: which matrix?

Classical CV, Convolution

Questions:

1. Compute $O(5, 7)$

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}	I_{17}	I_{18}	I_{19}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}	I_{27}	I_{28}	I_{29}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}	I_{37}	I_{38}	I_{39}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}	I_{47}	I_{48}	I_{49}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}	I_{57}	I_{58}	I_{59}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}	I_{67}	I_{68}	I_{69}

K_{11}	K_{12}	K_{13}
K_{21}	K_{22}	K_{23}

$$O_{57} = I_{57}K_{11} + I_{58}K_{12} + I_{59}K_{13} + I_{67}K_{21} + I_{68}K_{22} + I_{69}K_{23}$$

$$O(i, j) = \sum_{k=1}^m \sum_{l=1}^n I(i+k-1, j+l-1)K(k, l)$$

1. The average filter is a 3x3 matrix convolution: which matrix?

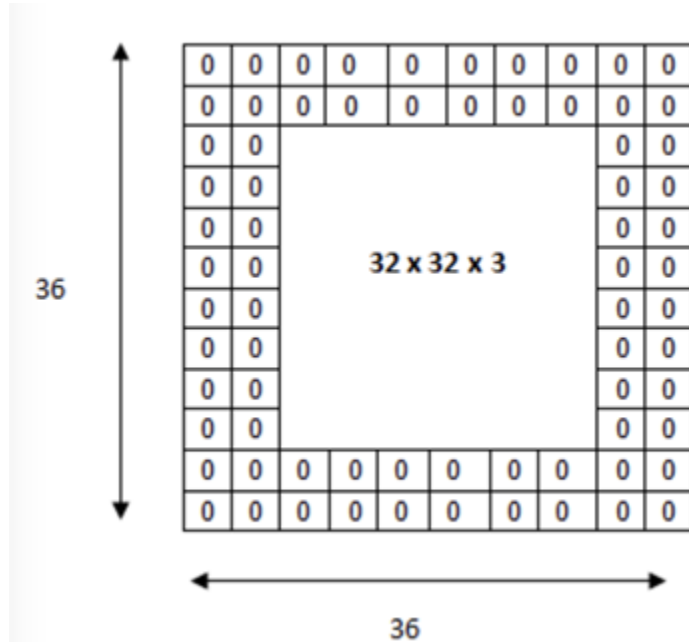
$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Classical CV, Convolution

- Note that the output image is clipped ($N \times M$ input image gives $(N-1) \times (M-1)$ output image) because the kernel can only be moved to positions where it fits entirely within the image.
- Several implementations typically add zeroes around the image to slide the kernel to all positions.
- One advantage of this approach is that the output image is the same size as the input image. Unfortunately, the invented pixel values often distort the output image at these places.
- This is called padding (zero-padding here). This is a parameter in convolutional neural network.

Classical CV, Convolution

- Zero padding example for a 5x5 convolution matrix



Classical CV, Convolution



Original

•0	•0	•0
•0	•1	•0
•0	•0	•0

=

?

Classical CV, Convolution



Original

•0	•0	•0
•0	•1	•0
•0	•0	•0

=



Filtered
(no change)

Classical CV, Convolution



Original

•0	•0	•0
•0	•0	•1
•0	•0	•0

=

?

Classical CV, Convolution



Original

•0	•0	•0
•0	•0	•1
•0	•0	•0

=



Shifted right
By 1 pixel

Classical CV, Convolution



Original

 $\frac{1}{9}$

•1	•1	•1
•1	•1	•1
•1	•1	•1

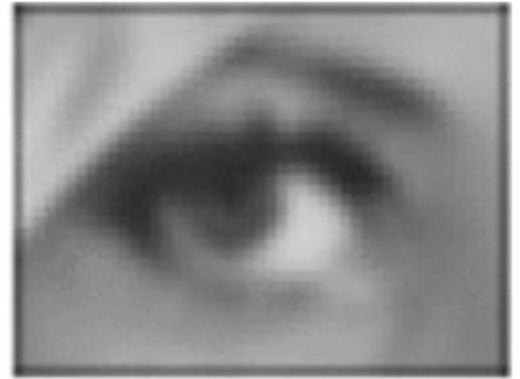
= ?

Classical CV, Convolution



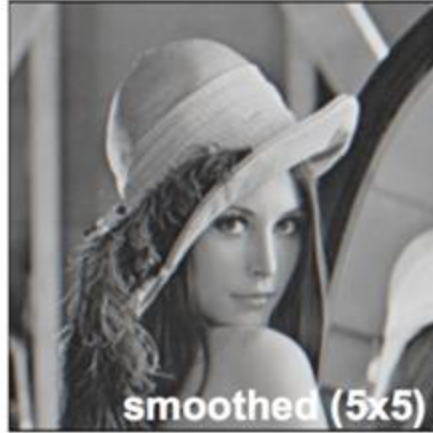
Original

$$\frac{1}{9} \begin{bmatrix} \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \\ \bullet 1 & \bullet 1 & \bullet 1 \end{bmatrix} =$$



Blur (with a
box filter)

Classical CV, Edge Detection



•0	•0	•0
•0	•1	•0
•0	•0	•0

-

•1	•1	•1
•1	•1	•1
•1	•1	•1

$\frac{1}{9}$

=

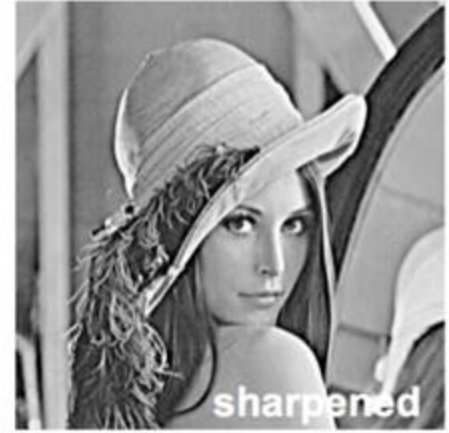
Classical CV, Sharpening



+ a



=



•0	•0	•0
•0	•1	•0
•0	•0	•0

+

•0	•0	•0
•0	•1	•0
•0	•0	•0

$-\frac{1}{9}$

•1	•1	•1
•1	•1	•1
•1	•1	•1

=

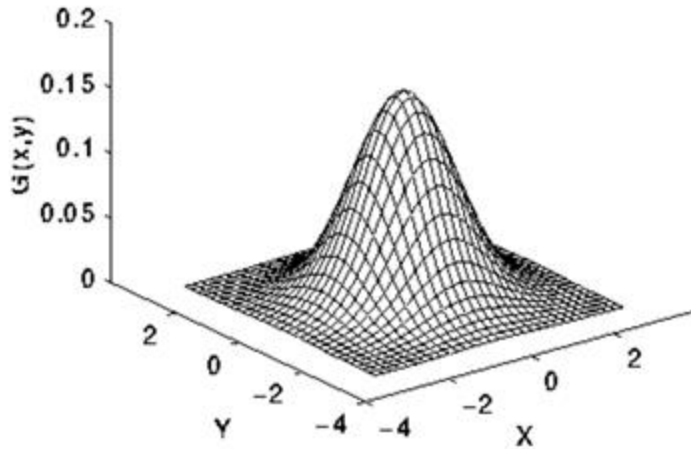
•0	•0	•0
•0	•2	•0
•0	•0	•0

$-\frac{1}{9}$

•1	•1	•1
•1	•1	•1
•1	•1	•1

Classical CV, Gaussian Blur

The Gaussian smoothing operator is a 2-D convolution operator that is used to blur images. In this sense it is similar to the mean filter, but it uses a different kernel that represents the shape of a Gaussian hump.



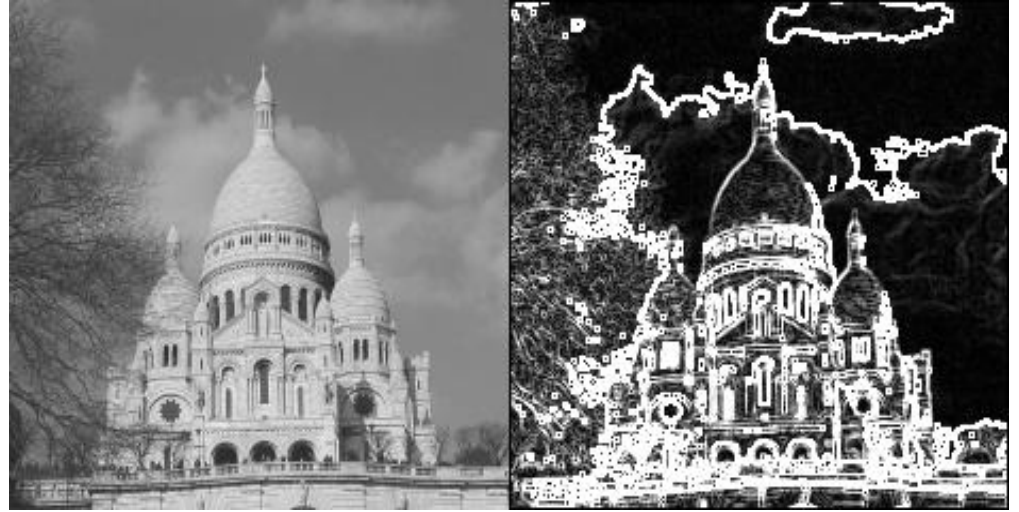
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\frac{1}{273}$$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

Filtre de Sobel

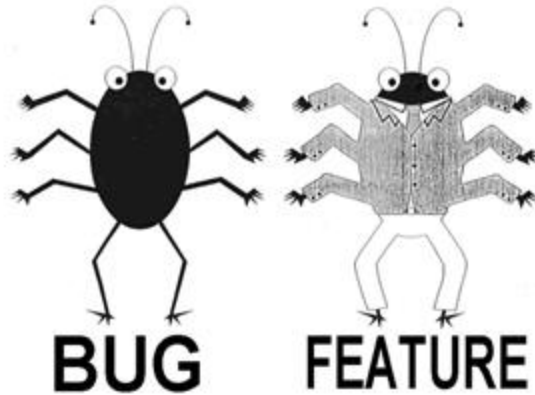
$$G = \sqrt{G_x^2 + G_y^2}$$

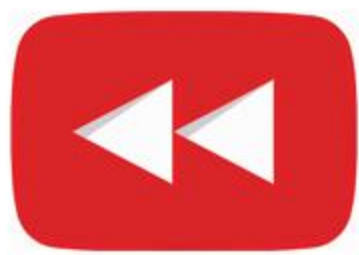


$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad \text{et} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

A few words of what is a tensor

Tutorial 1





RECAP

Research paper (Homework)

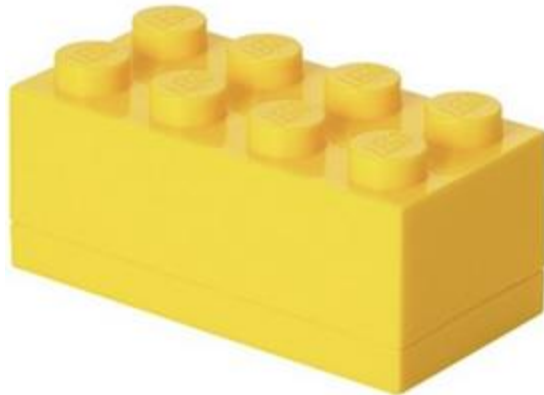
- At some point you will have to go to sources not only tutorial or wikipedia pages
- YOLO 2015
- How to find it?
- How to read it (spend 2 hours on it)



Summary

1. History
2. Classic CV
- 3. ML Basics**
4. From a neuron to a neural net
5. Backpropagation
6. Activation functions
7. Batch normalization
8. DNN, zoo

ML Basics

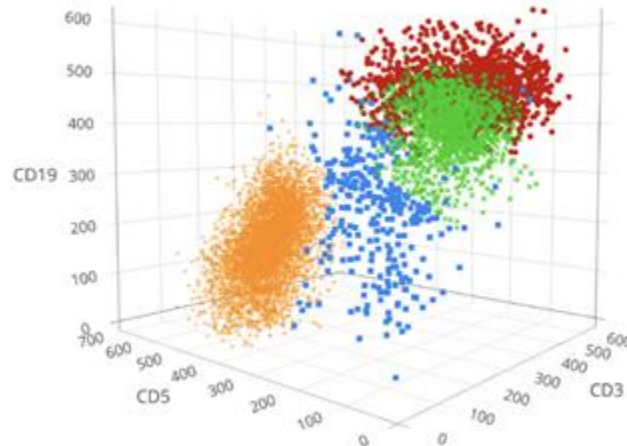


Objectives for this module

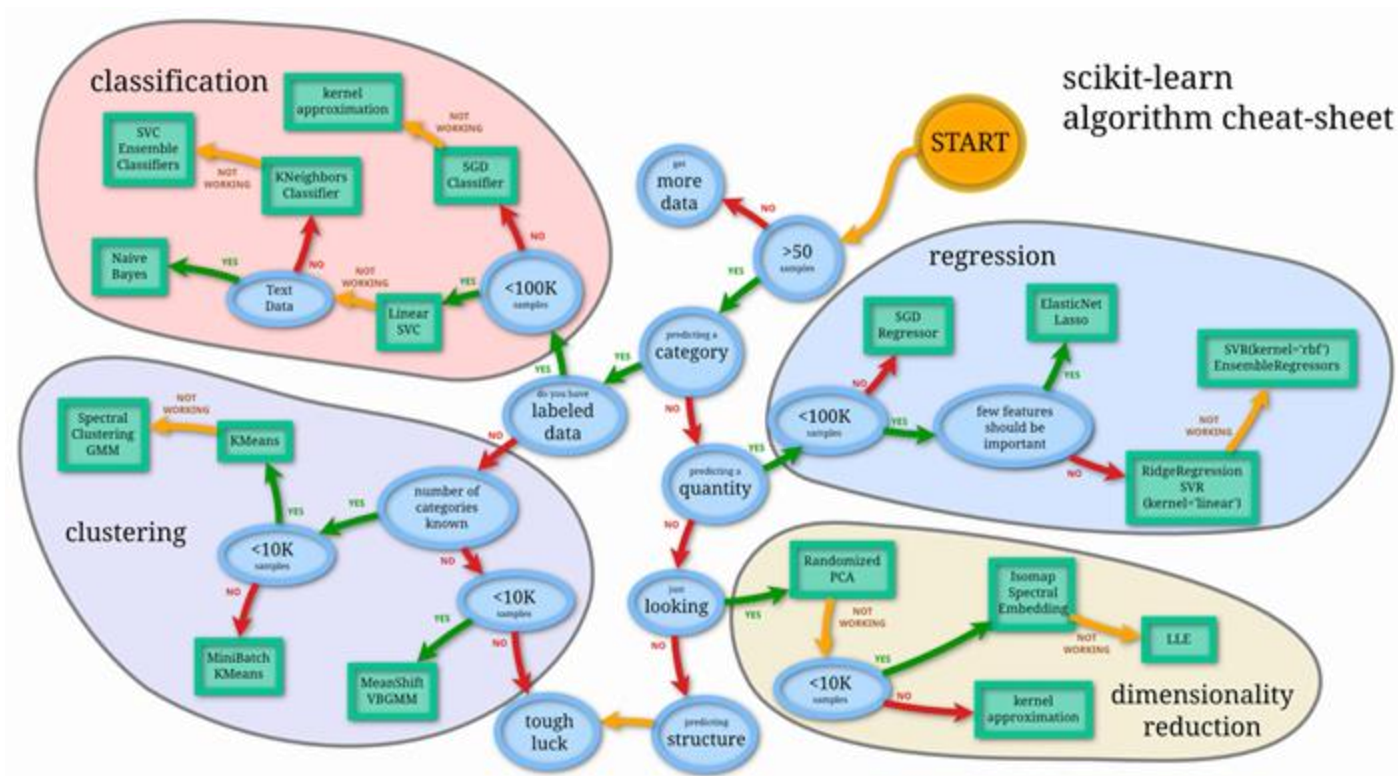
1. Explain the specificity of CV when using ML tools
2. Give you a grasp of the depth and width of the field
3. Solve and implement a CV problem with a ML solution
4. Choose the best architecture / algo to solve a CV problem with ML
5. We will more specifically focus on Convolutional Neural Networks (CNN)
6. Learn to learn

ML Basics

- Different types of learning: Supervised unsupervised learning
- https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html



ML Basics



Classical ML algorithms

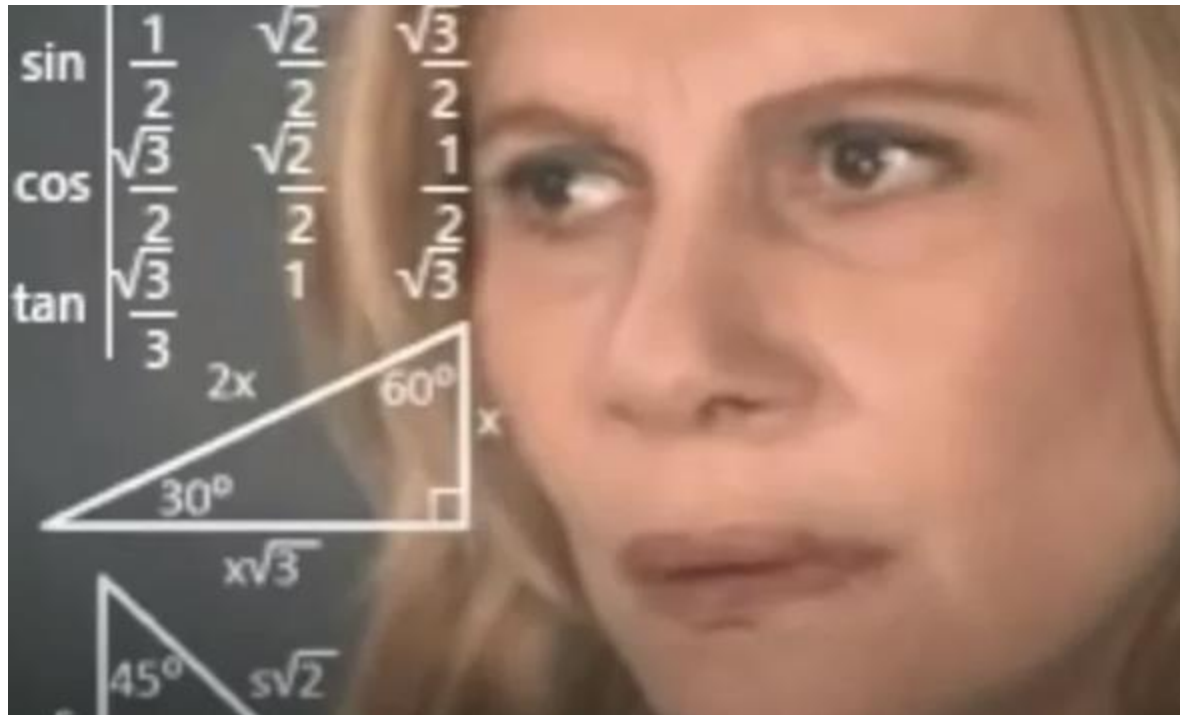
- K-mean
- SVM Support Vector Machine (Séparateur à Vaste Marge)
- Random Forest (decision trees)
- PCA
- MLP Multi-Layer Perceptron
- t-SNE Hinton
- Adaptive boosting (AdaBoost)

ML Basics, supervised learning

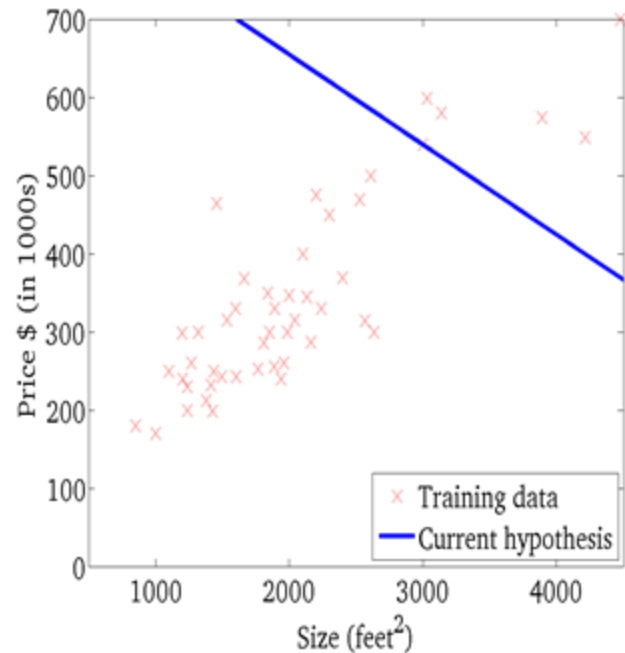
- Dataset with labels ($X \Rightarrow y$)
- We know what we want to predict
- Classification / Regression
- How can an algorithm learn?
- What is a model?



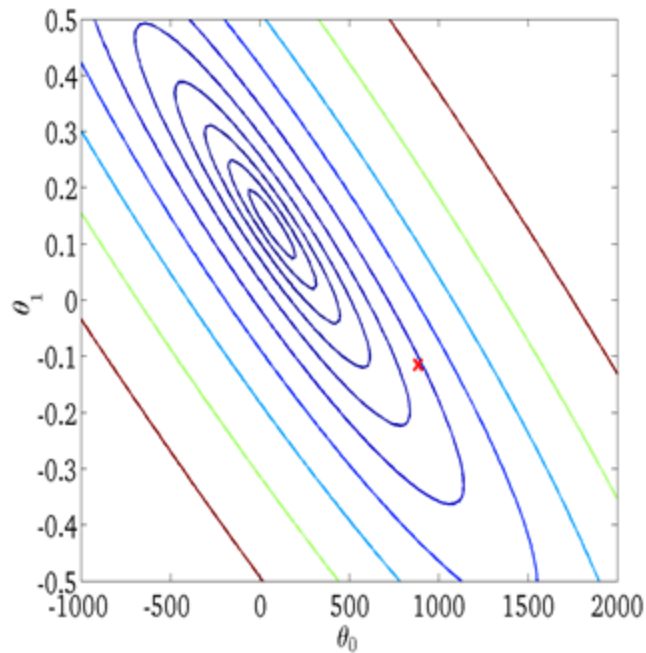
Univariate linear regression



Gradient descent algorithm



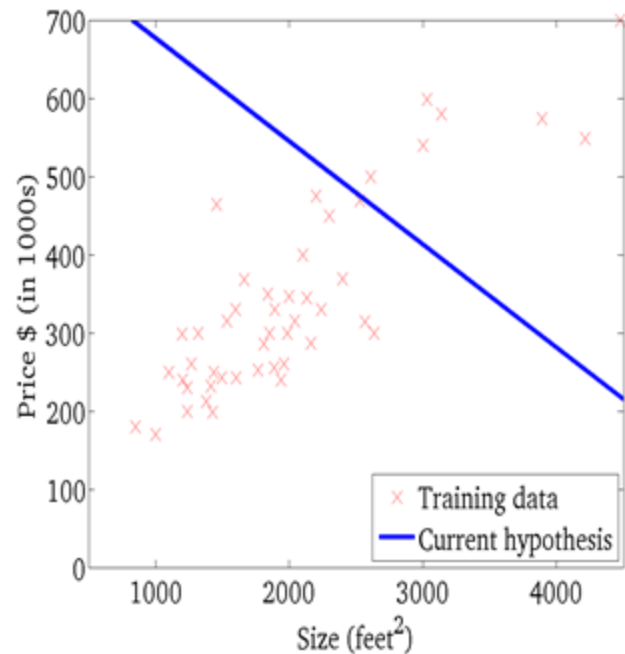
$h_{\theta}(x)$



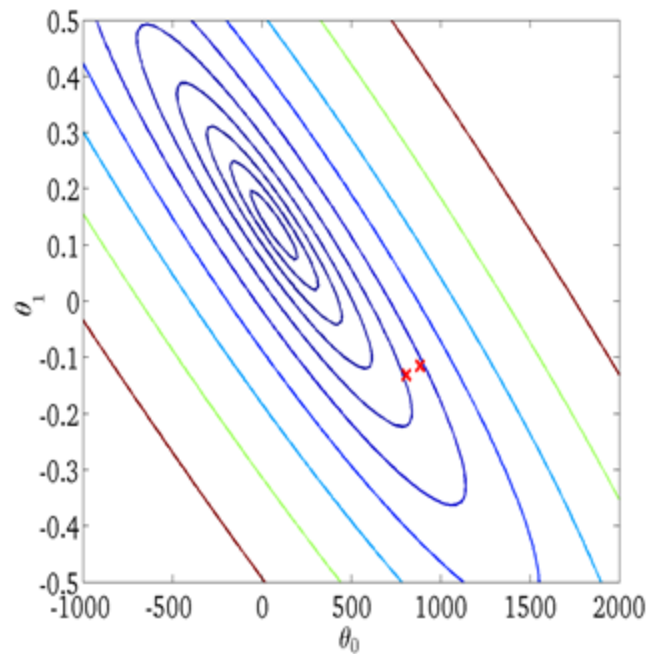
$J(\theta_0, \theta_1)$

Andrew NG

Gradient descent algorithm



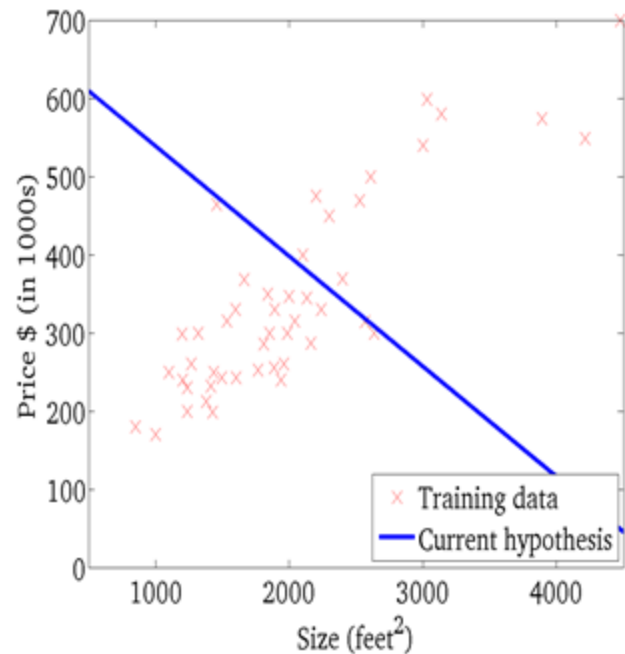
$h_{\theta}(x)$



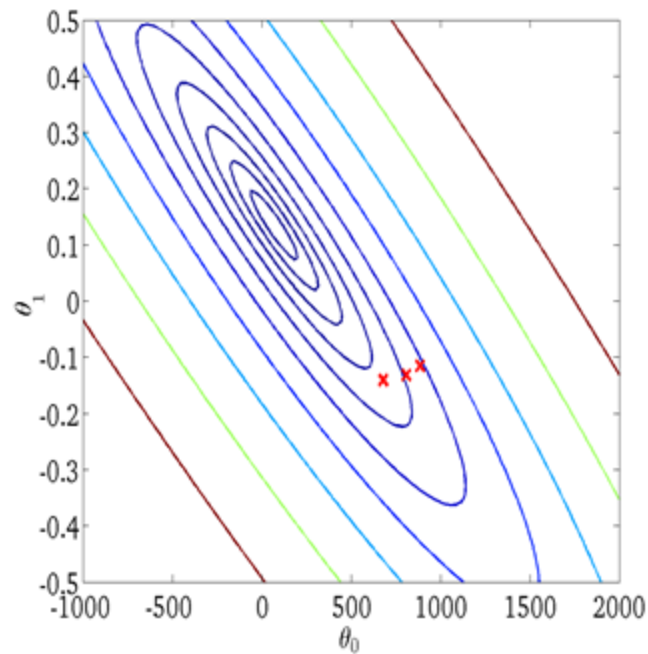
$J(\theta_0, \theta_1)$

Andrew NG

Gradient descent algorithm



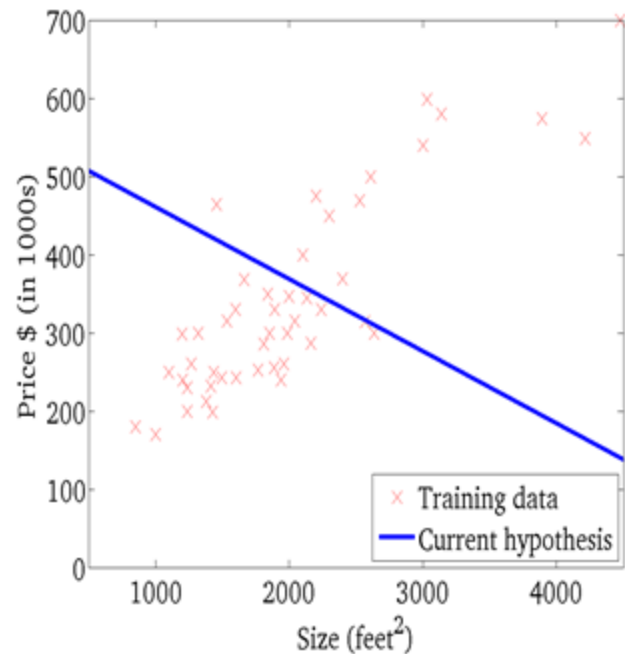
$$h_{\theta}(x)$$



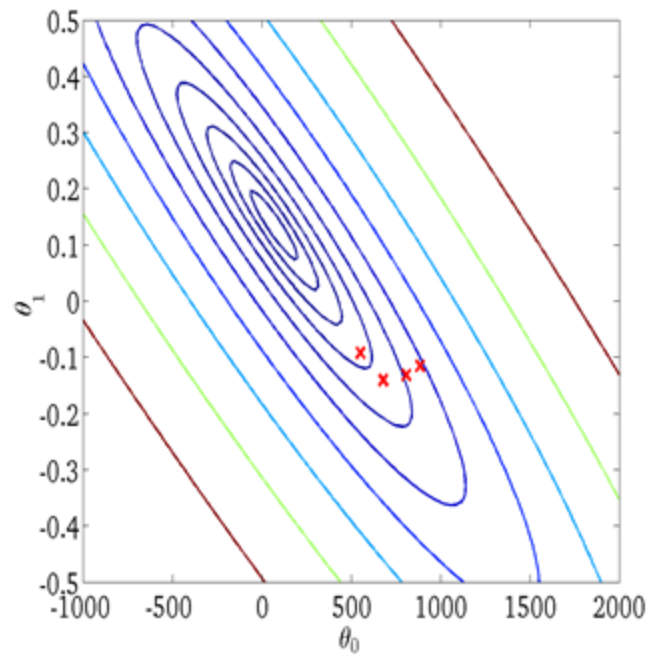
$$J(\theta_0, \theta_1)$$

Andrew NG

Gradient descent algorithm



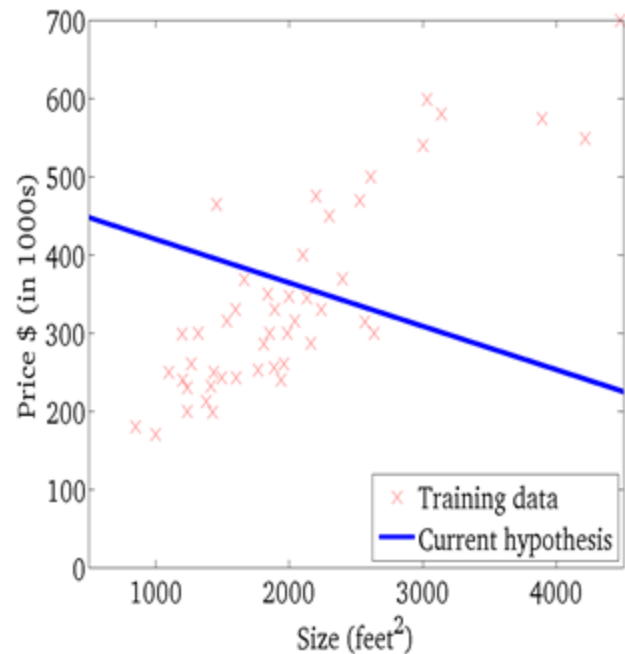
$$h_{\theta}(x)$$



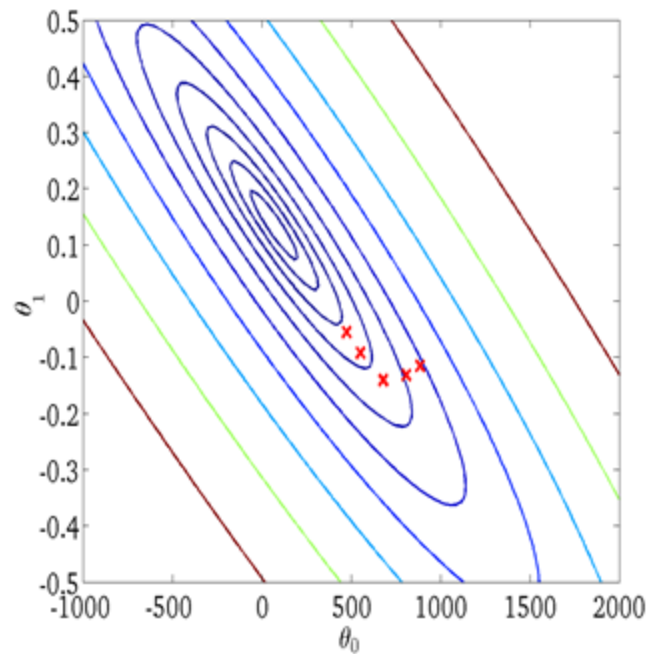
$$J(\theta_0, \theta_1)$$

Andrew NG

Gradient descent algorithm



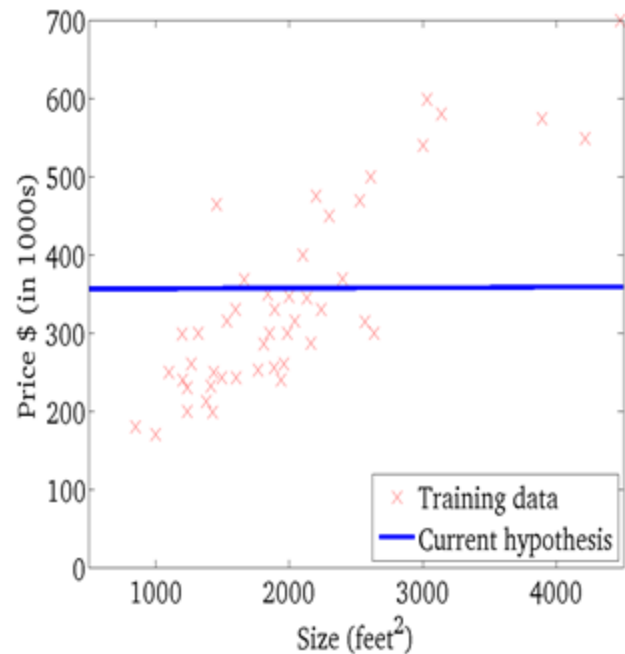
$h_{\theta}(x)$



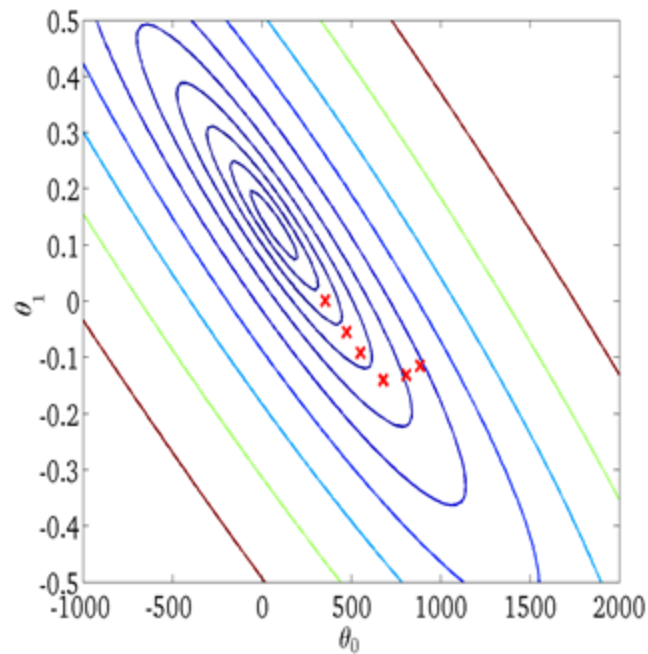
$J(\theta_0, \theta_1)$

Andrew NG

Gradient descent algorithm



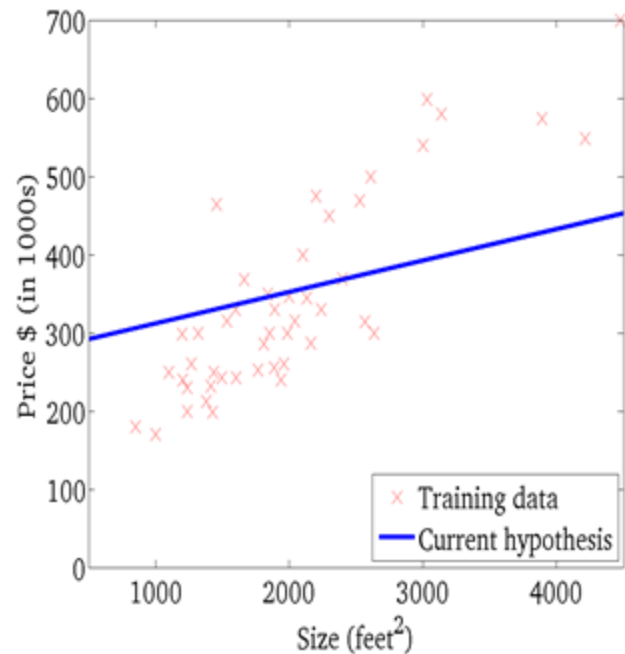
$$h_{\theta}(x)$$



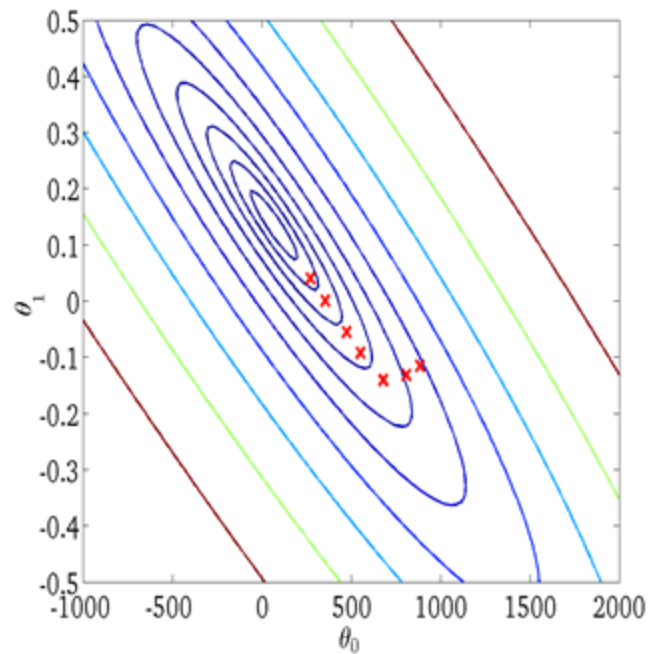
$$J(\theta_0, \theta_1)$$

Andrew NG

Gradient descent algorithm



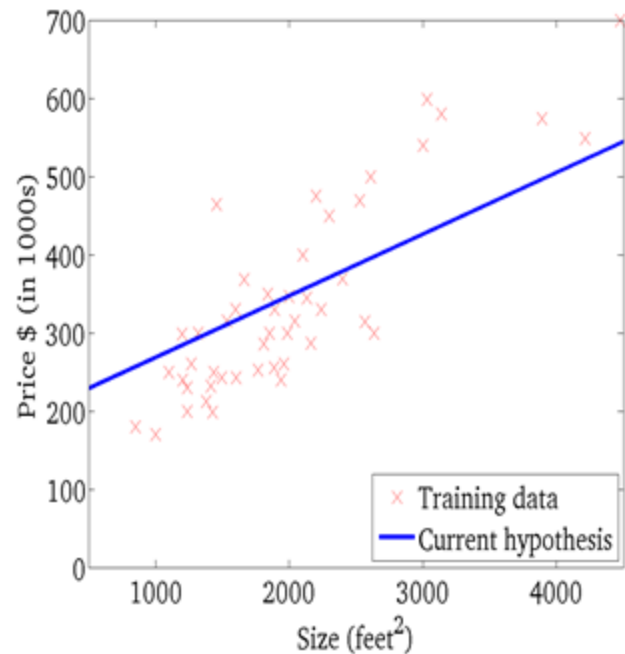
$$h_{\theta}(x)$$



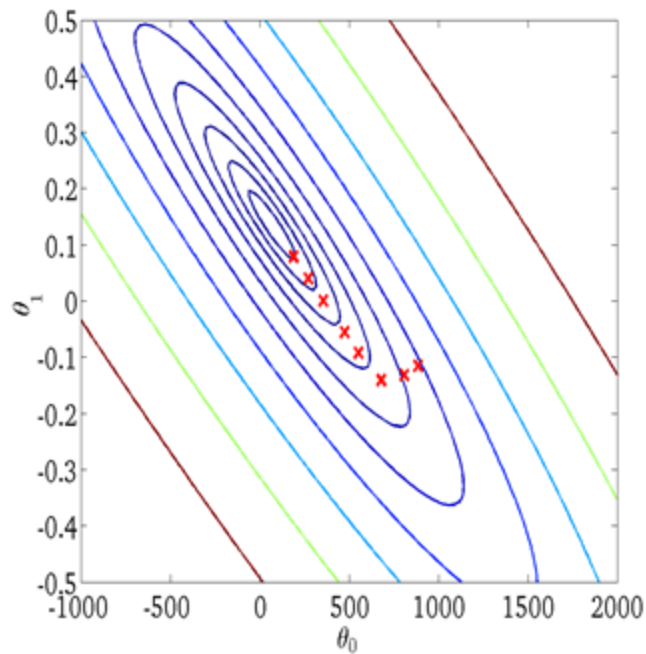
$$J(\theta_0, \theta_1)$$

Andrew NG

Gradient descent algorithm



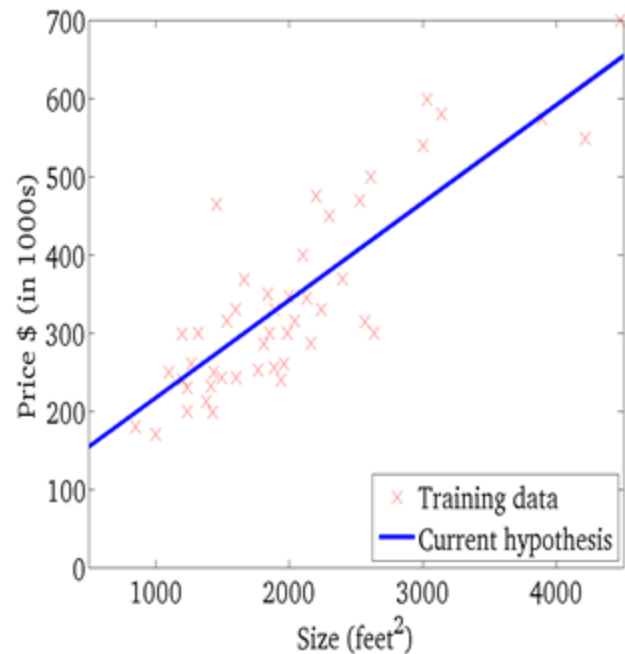
$$h_{\theta}(x)$$



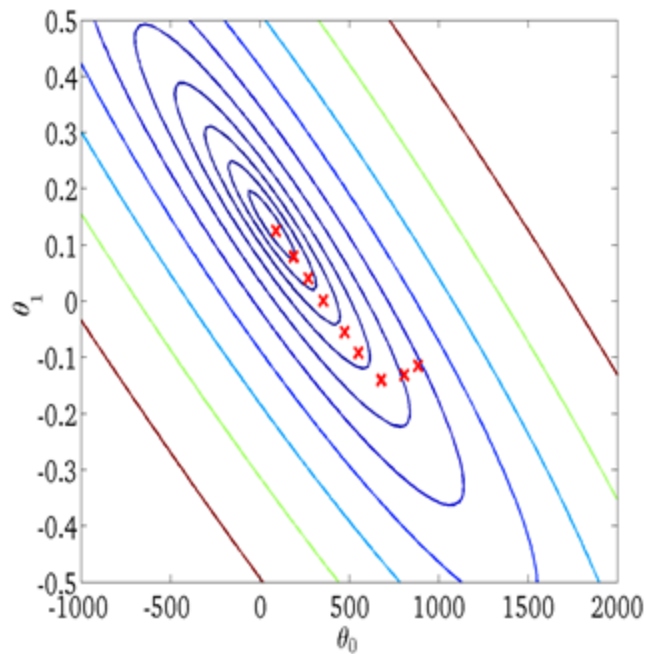
$$J(\theta_0, \theta_1)$$

Andrew NG

Gradient descent algorithm



$$h_{\theta}(x)$$

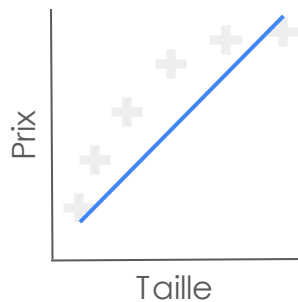


$$J(\theta_0, \theta_1)$$

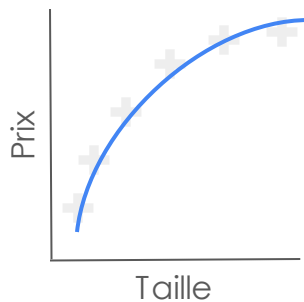
Andrew NG

ML Basics, Overfitting

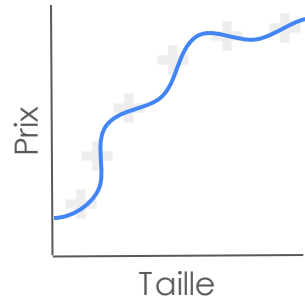
- Theoretically the cost can be null



$$\theta_0 + \theta_1 x$$



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

ML Basics, Regularisation

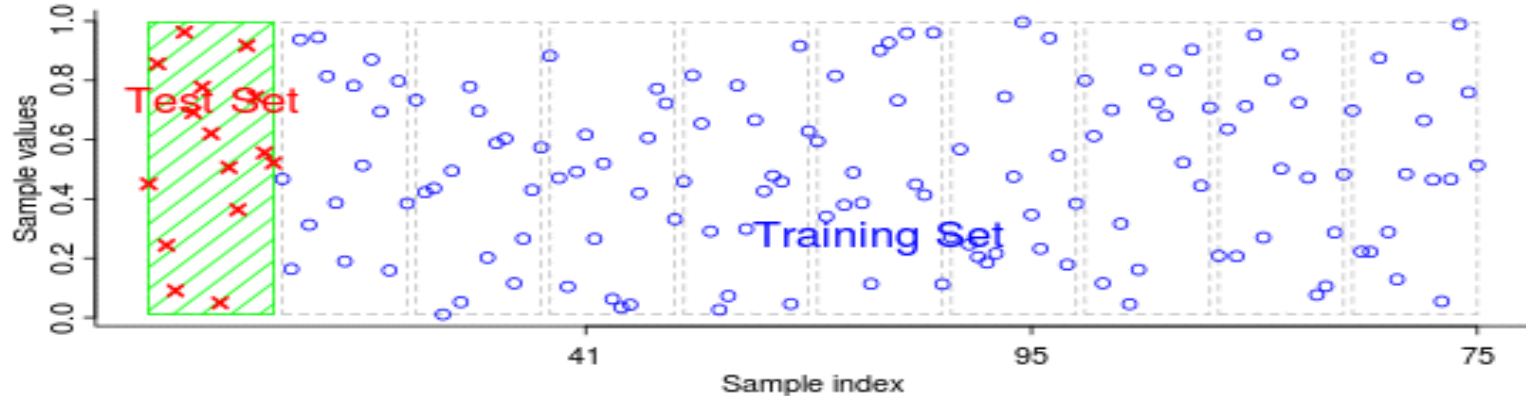
$$J(\theta) = \frac{1}{2m} \left(\sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2 + \lambda_1 \sum_{j=1}^n |\theta_j| + \lambda_2 \sum_{j=1}^n \theta_j^2 \right)$$

Data

- Training data and test data
- Validation data, the issue of avoiding bias linked to the results of models on the test set (ensemble models)...

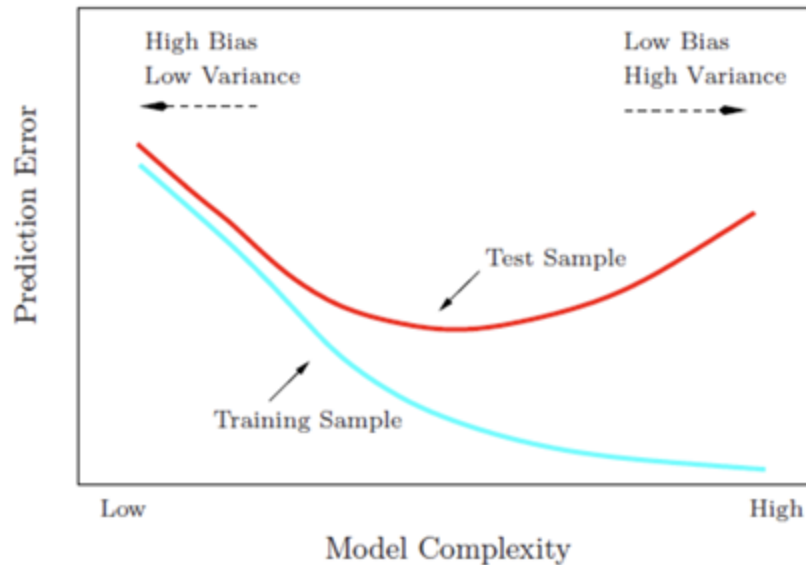
ML Basics, k-fold Cross-validation

- Is your model good?



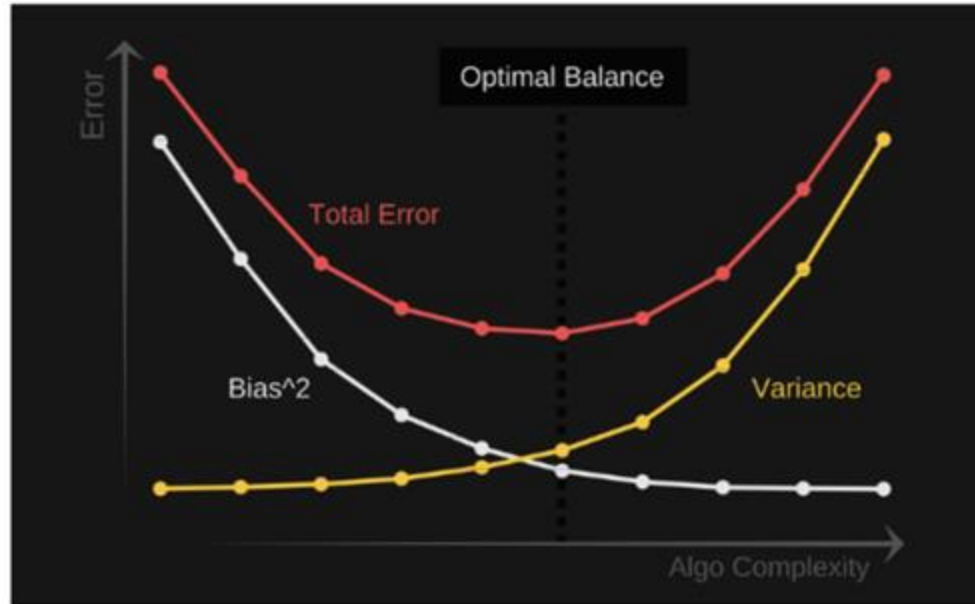
ML Basics, Cross-validation

- Make good hyperparameter choices (and stop at right time)



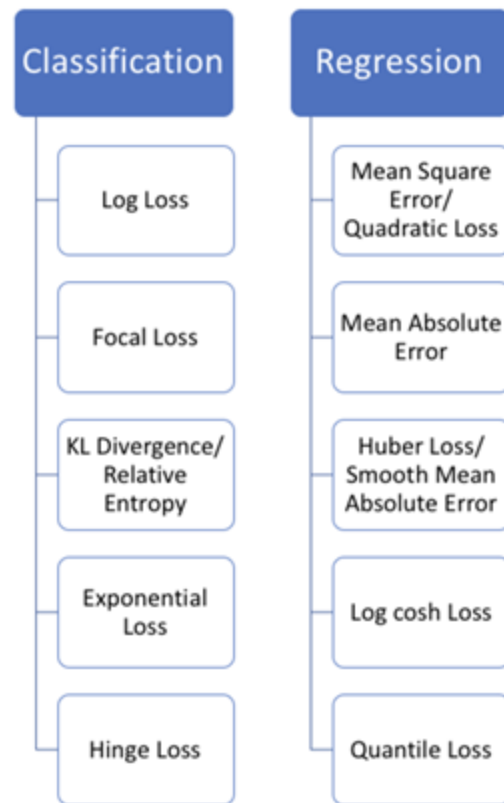
ML Basics, Bias/Variance tradeoff

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



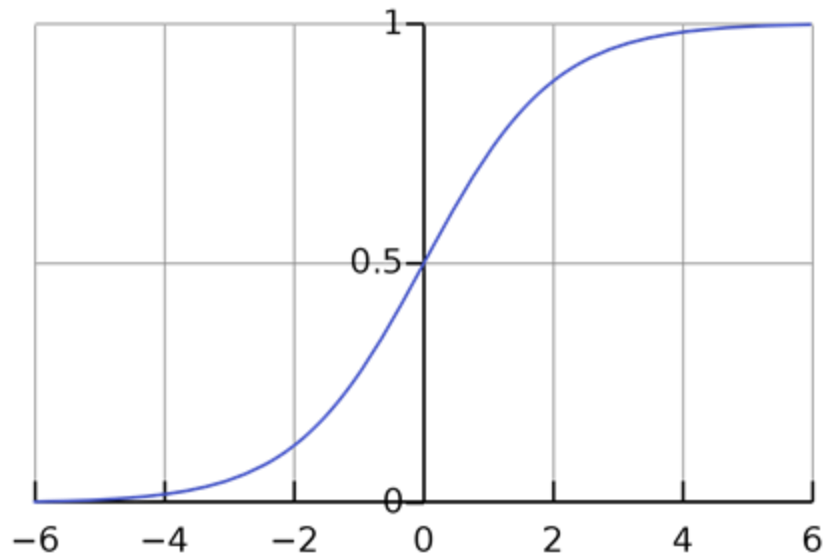
ML Basics, Loss & metrics

- MSE for convexity and derivability (Mean Square Error)
 - MAE (Mean Absolut Error) for robustness against outliers (median)
 - We also speak about objective function
-
- <https://heartbeat.comet.ml/5-regression-loss-functions-all-4fb140e9d4b0>
 - <https://keras.io/api/metrics/>
 - <https://keras.io/api/losses/>



ML Basics, Logistic Regression

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



ML Basics, Logistic Regression

- Classification problem



- Outputs a probability

- Sigmoid + threshold

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

ML Basics, Logistic Regression

- Cross-entropy loss for binary classification

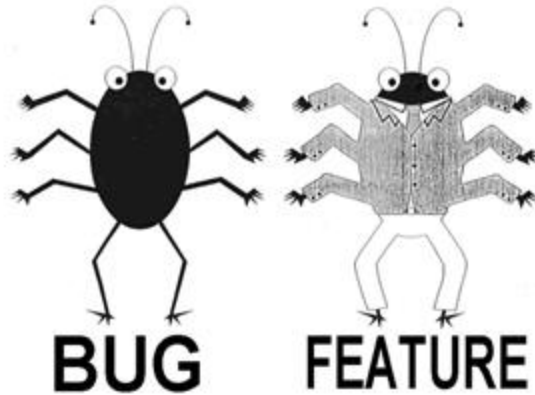
$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

Loss functions (fitness or cost function)

- Cross-entropy (binary classification)
- Softmax (hardmax)
- Negative log likelihood
- The probabilistic framework to find the loss function
- Quadratic minimisation
- The essence of the algorithm, many algorithms work like this in CV (optical flow, model fitting). An essential idea

Tutorial 2

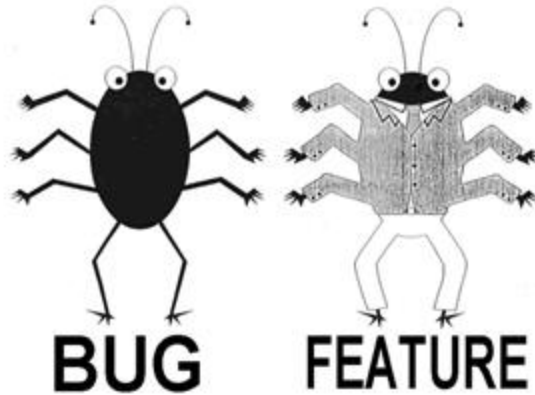


YOLO 2015



RECAP

Tutorial 3





RECAP

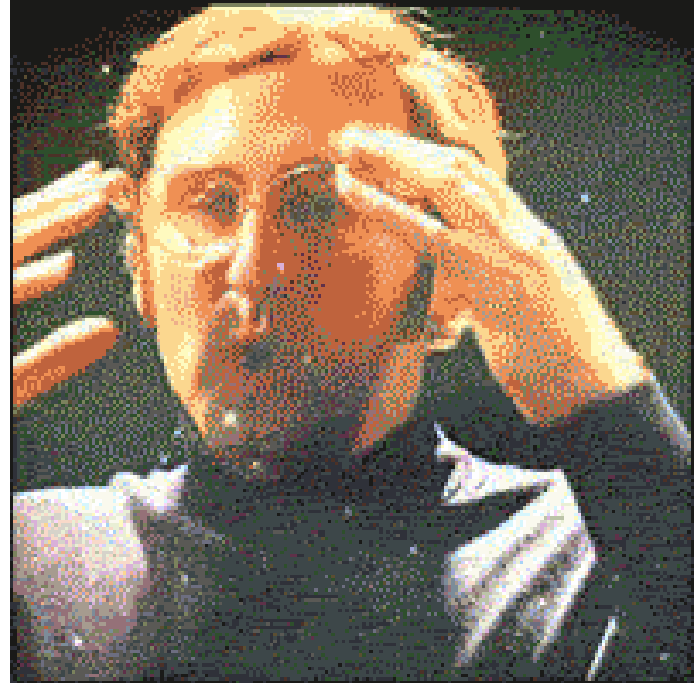
Demo => car-detector

code: <https://github.com/Saxamos/car-detector>

articles: <https://blog.octo.com/ia-embarquee-deployer-du-deep-learning-sur-un-raspberry/>

<https://blog.octo.com/lia-embarquee-entrainer-deployer-et-utiliser-du-deep-learning-sur-un-raspberry-partie-2/>

<https://blog.octo.com/lia-embarquee-entrainer-deployer-et-utiliser-du-deep-learning-sur-un-raspberry-partie-3/>



Summary

1. History
2. Classic CV
3. ML Basics
- 4. From a neuron to a neural net**
5. Backpropagation
6. Activation functions
7. Batch normalization
8. DNN, zoo

From a neuron to a NN

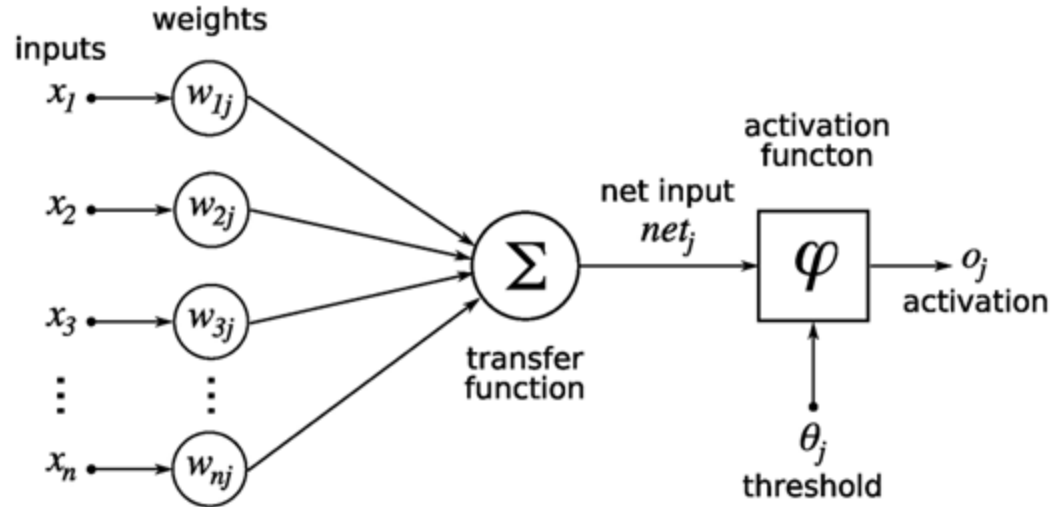


Mille milliards
de mille sabords

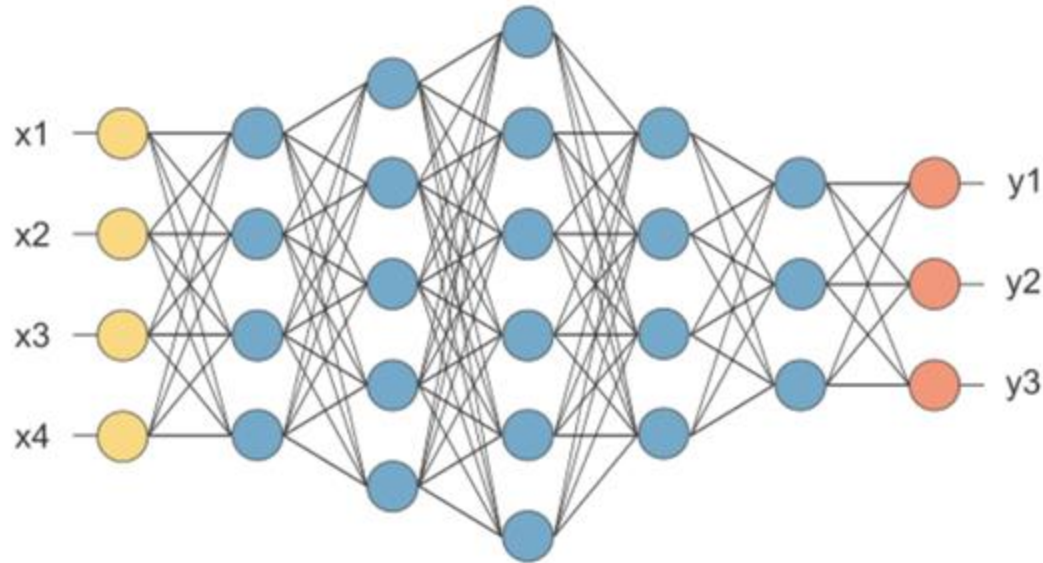


10^{15} sabords

A neuron, brain inspiration...



A neuron, composed by several neuron layers (MLP)



Batch

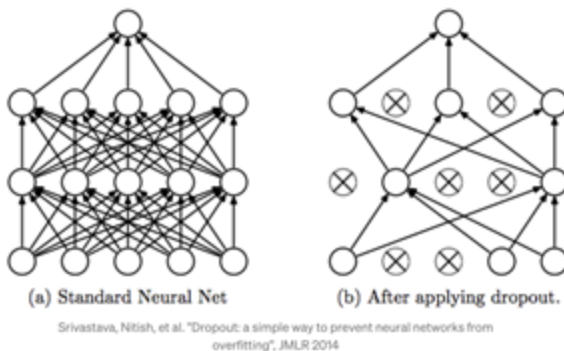
- for training, one often uses a SGD optimizer to find weights
 - for ULR, we computed loss and gradient over the whole train dataset
 - IRL, whole dataset too big for RAM & computation of loss + gradient heavy
 - solution: do each step on a sample of train set ([convergence?](#))
-
- *batch can be reduced to one item (online learning)*

NN, training

1. **init weights** => millions of θ initialized randomly
2. **feedforward** => give input images (batch), look at the results
3. **loss computation** => compare predictions to labels (proba, cross-entropy)
4. **backpropagation** => 1 SGD step, update all weights
5. **iterate** => go back to 2. (while loss > eps or n_iter < epoch)

NN, Dropout

- L1 and L2 were used but not sufficient to prevent overfitting in NN



- Dropout introduction in a 2014 paper
- Widely used in deep learning to prevent overfitting (not for C layer)
- Now replaced by **batch normalisation** (2015)

NN, Dropout

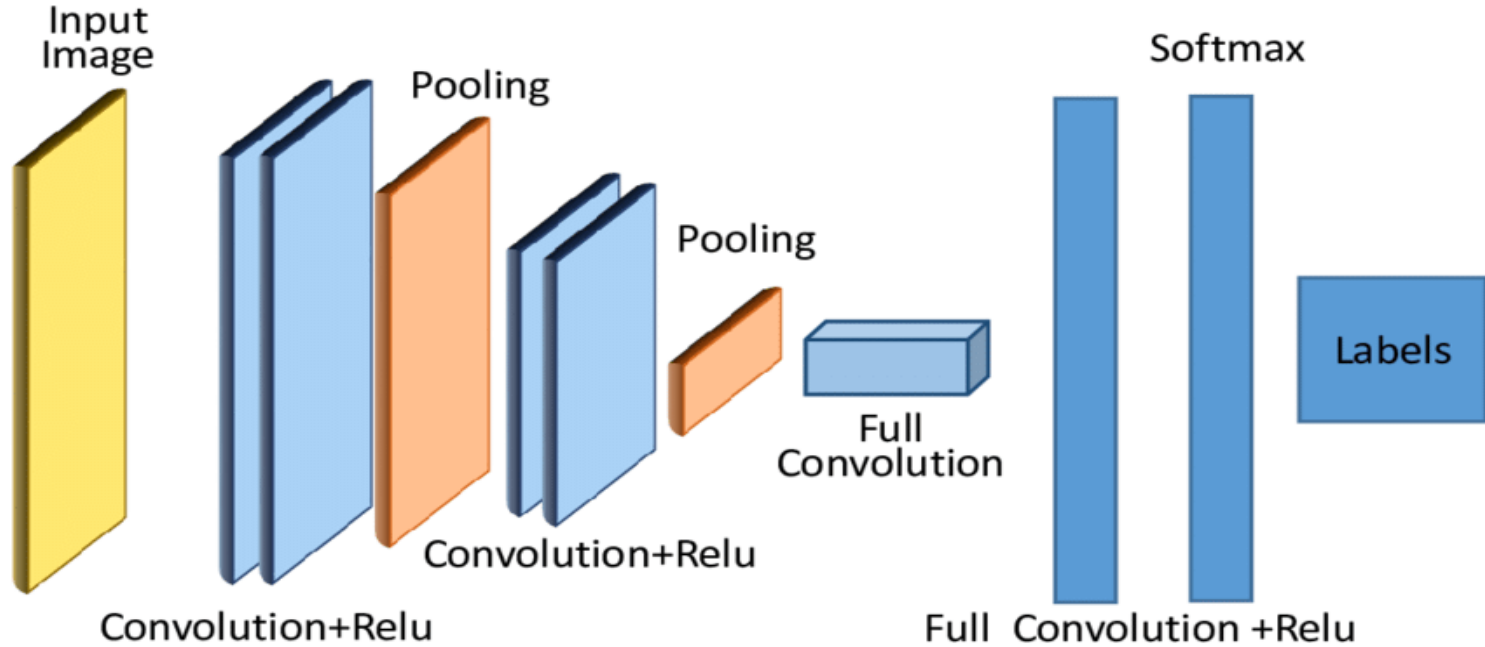
Implementation:

- During training time, dropout randomly sets neurons to zero with probability p_{kill}
- During inference time, dropout does not kill neurons, but all the weights in the layer were multiplied by $p_{keep} = 1 - p_{kill}$
- One of the major motivations of doing so is to make sure that the distribution of the values after affine transformation during inference time is close to that during training time

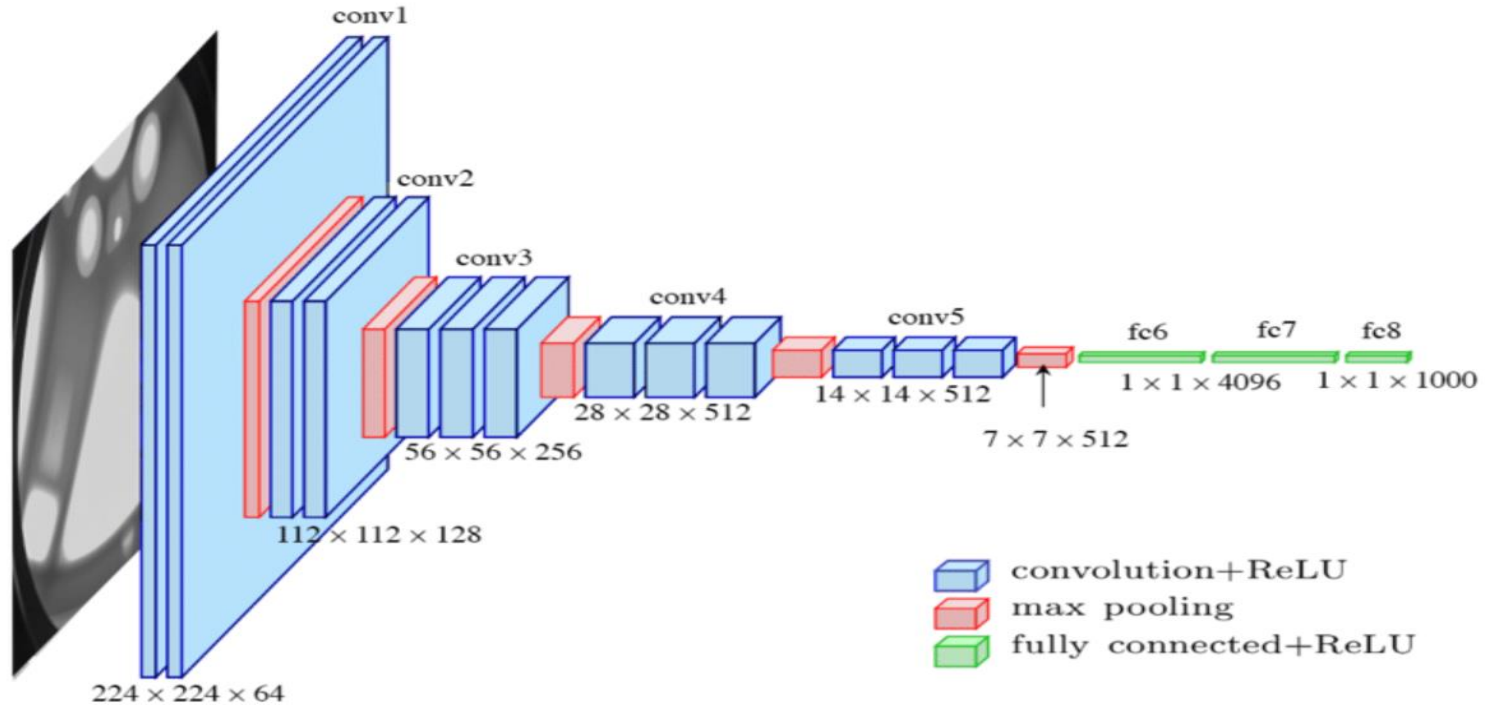
CNN Convolutional Neural Networks

- Although the field is wide, NN could potentially solve "70%" of your problems in CV.
- Similarly CNN, or relative small modification of it, is a first tool of choice for "70%" of your problems
- So, roughly, almost half of the vision problems could be addressed with this technique or not so dramatic derivation of it.

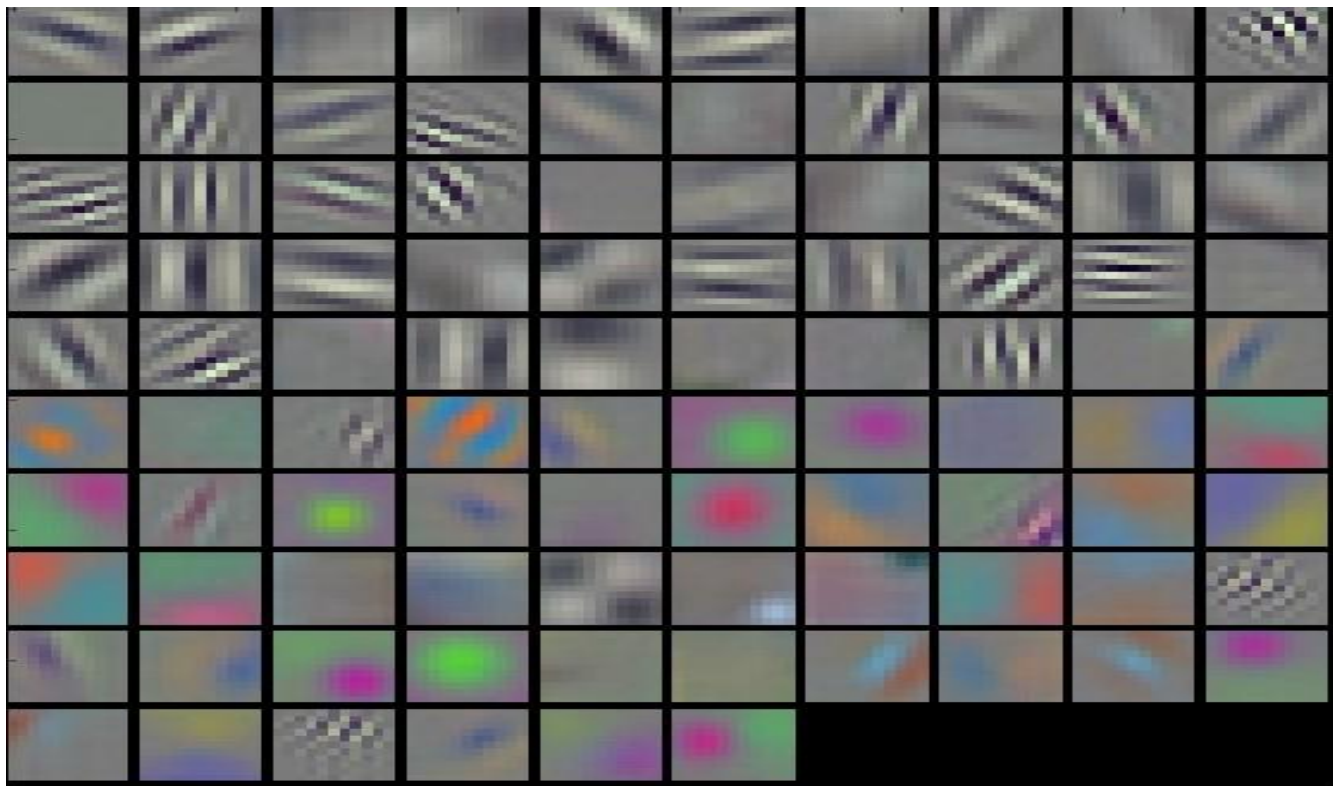
A CNN architecture



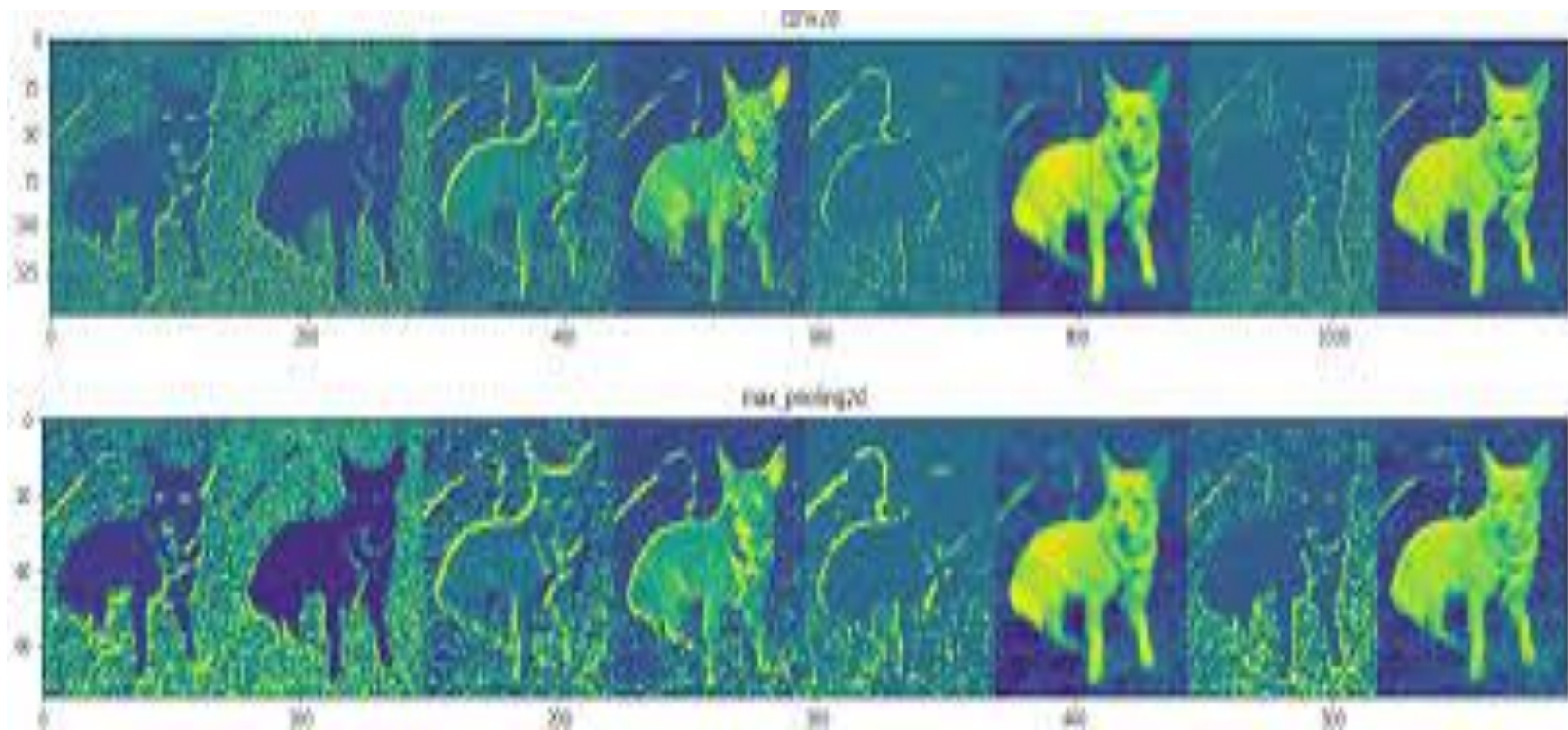
CNN architecture



NN, Filter visualisation



Activation maps



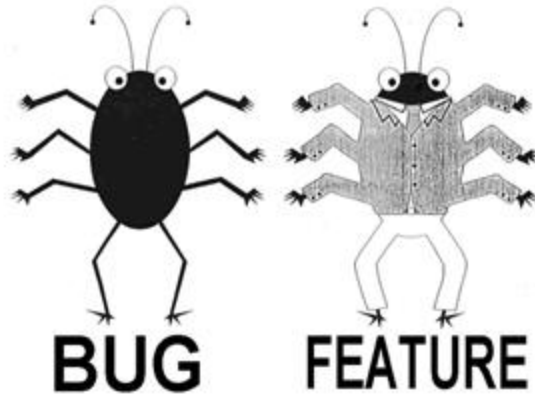
Mathematical formulation

- Quite easy?
- They are quite lot of details that are hidden, nonetheless that is an explanation that could suffice depending on your audience.
- Can you see what is missing?!

Good introductory book

- <http://neuralnetworksanddeeplearning.com/> from Mickael Nielson
- Try to read first chapter, it provides all the explanation and the code to understand and implement in Python a MLP with backprop from scratch.
- Of course your frameworks (Keras, Tensorflow, Pythorch) not only do this, but also deals with parallelisation to be able to use GPUs.

Tutorial 4



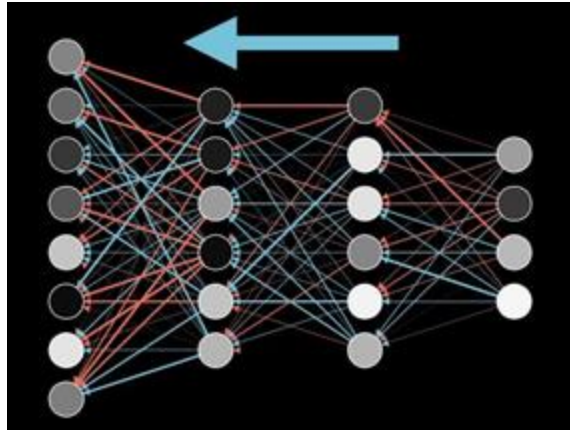


RECAP

Summary

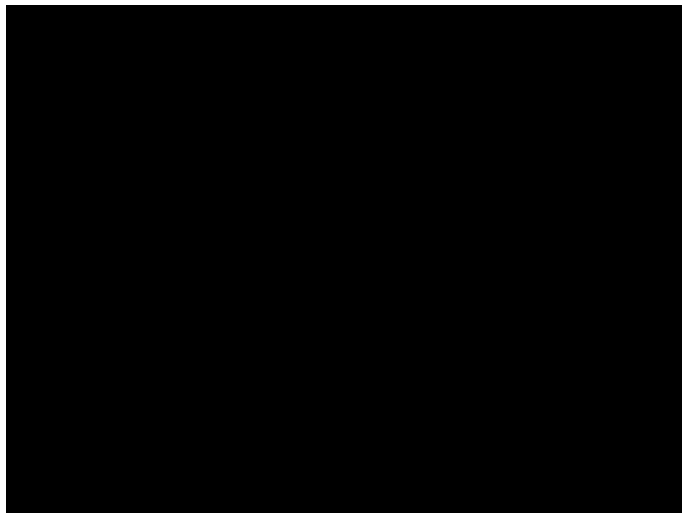
1. History
2. Classic CV
3. ML Basics
4. From a neuron to a neural net
- 5. Backpropagation**
6. Activation functions
7. Batch normalization
8. DNN, zoo

Backpropagation



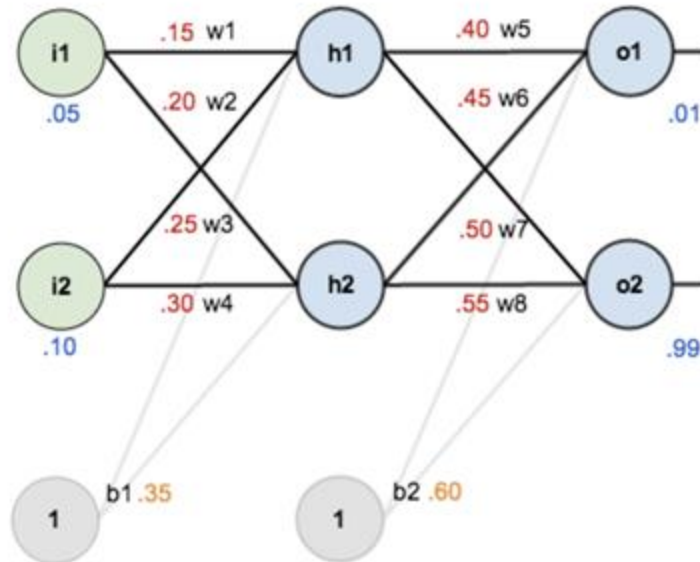
Backpropagation, resources

- Blog calculus: [A Step by Step Backpropagation Example](#)
- [Video calculus](#)
- [Video animation](#) for intuition



Backpropagation, a canonical example

In order to have some numbers to work with, here are the **initial weights**, the **biases**, and **training inputs/outputs**:



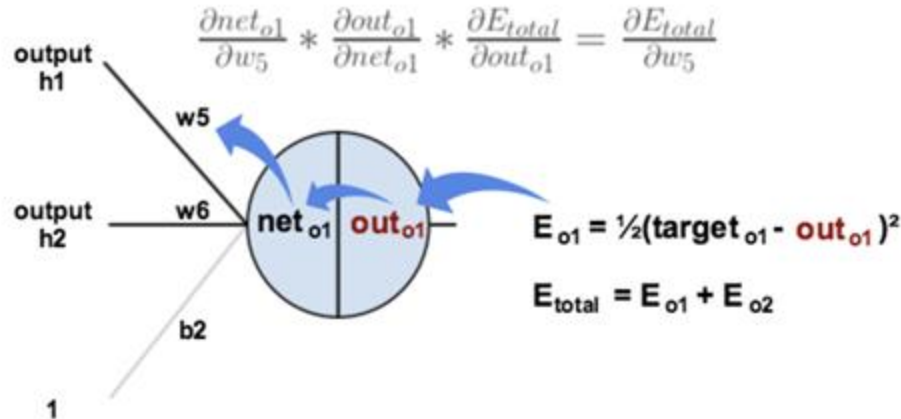


Backpropagation, chain rule (f o g)'

By applying the [chain rule](#) we know that:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Visually, here's what we're doing:



Chain rule

$$h(x) = f(g(x))$$

$$h'(x) = f'(g(x))g'(x).$$

Chain rule

$$\begin{aligned}(f \circ g \circ h)'(a) &= f'((g \circ h)(a)) \cdot (g \circ h)'(a) \\ &= f'((g \circ h)(a)) \cdot g'(h(a)) \cdot h'(a) = (f' \circ g \circ h)(a) \cdot (g' \circ h)(a) \cdot h'(a).\end{aligned}$$

$$\frac{dy}{dx} = \left. \frac{dy}{du} \right|_{u=g(h(a))} \cdot \left. \frac{du}{dv} \right|_{v=h(a)} \cdot \left. \frac{dv}{dx} \right|_{x=a},$$

Chain rule, why does it matter

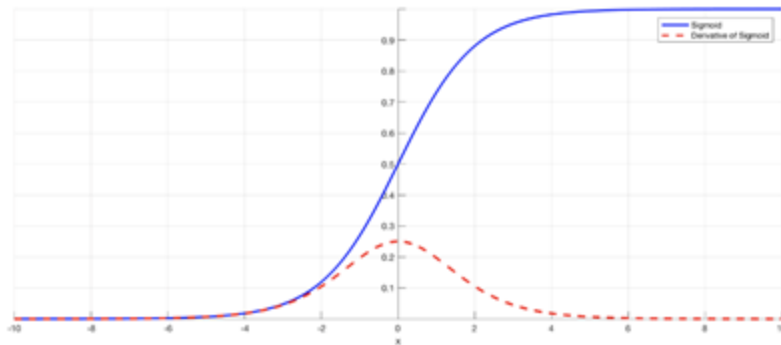
- $L = (f(W_2 * f(W_1 * X + B_1) + B_2) - y)^2$
 - $X = A_1$
 - $f(W_1 * X + B_1) = A_2$
 - $f(W_2 * f(W_1 * X + B_1) + B_2) = A_3 = f(Z_3)$
 - $Z_3 = W_2 * f(W_1 * X + B_1) + B_2 = W_2 * A_2 + B_2$
 - $dL/dW_2 = \dots$
 - $dL/dB_2 = \dots$
-
- We start at $dL/dA_3 (2 * A_3 - y)$ the value is stored, we move backward we evaluate dA_3/dZ_3 (which is f'), and then dZ_3/dW_2 or dZ_3/dB_2 comes quite easily and provide the evaluation of dL/dW_2 and dL/dB_2

Backpropagation, sigmoid derivative

The standard logistic function has an easily calculated [derivative](#). The derivative is known as the density of the [logistic distribution](#):

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

$$\frac{d}{dx} f(x) = \frac{e^x \cdot (1 + e^x) - e^x \cdot e^x}{(1 + e^x)^2} = \frac{e^x}{(1 + e^x)^2} = f(x)(1 - f(x))$$

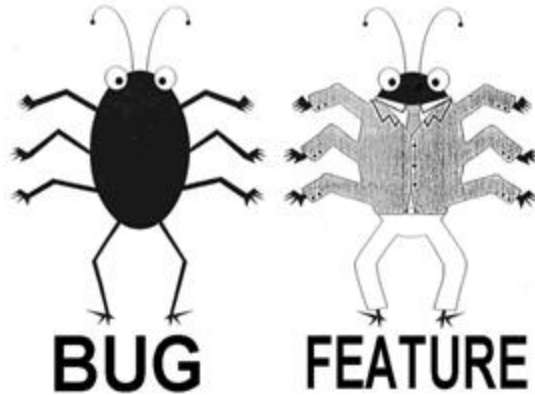


Graph of Sigmoid and the derivative of the Sigmoid function

Homework

- <http://cs231n.stanford.edu/2019/> lecture 6 training neural networks (initialisation and vanishing gradient issue)
- <http://cs231n.stanford.edu/2019/> lecture 4 introduction to neural networks if you are not completely clear about the backpropagation algorithm else you may look at lecture 3 if you have some time.

Tutorial 5





RECAP

How to improve a neural network?

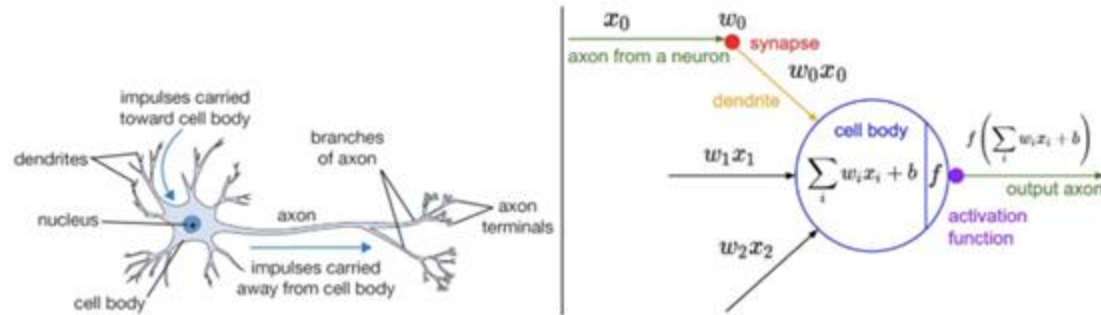
How to improve the performance of your model

- Data augmentation (mirror image, rotations, blur, noise)
- Transfer learning
- Hyper-parameter (why are they call like this, 2 categories)
- Better architecture

Summary

1. History
2. Classic CV
3. ML Basics
4. From a neuron to a neural net
5. Backpropagation
- 6. Activation functions**
7. Batch normalization
8. DNN, zoo

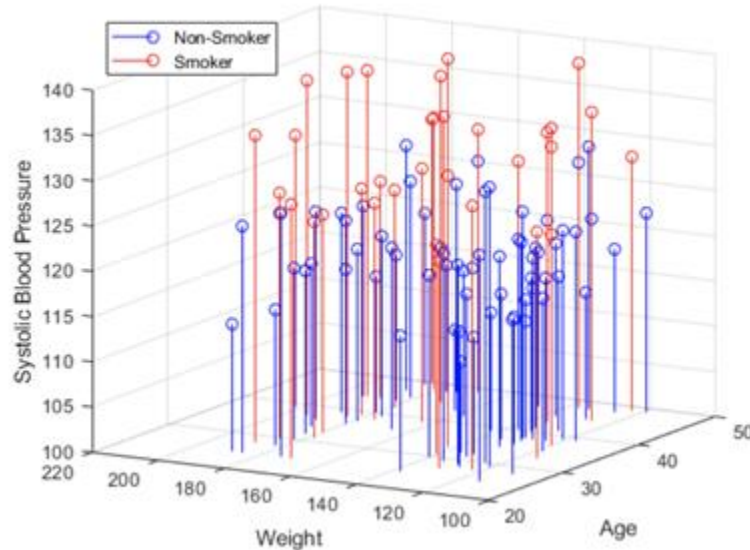
Activation functions



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

Activation function

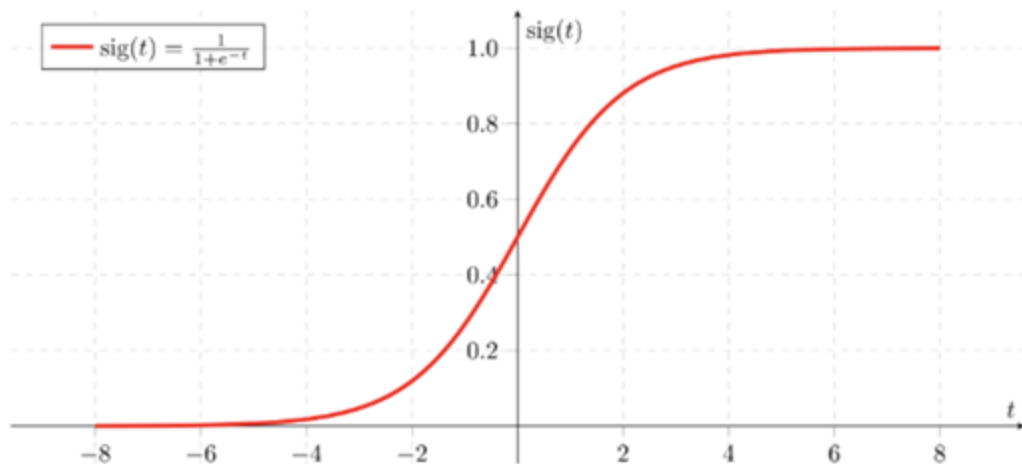
1. biological similarity
2. constraint output values of a neuron to a certain limit
- 3. add non-linearity into a neural network**



Activation function

1. **Vanishing Gradient problem:** we want our activation function to not shift the gradient towards zero
2. **Zero-Centered:** Output of the activation function should be symmetrical at zero so that the gradients do not shift to a particular direction, convergence of the network is faster (cf homework lecture)
3. **Computational Expense:** Activation functions are calculated millions of times. Hence, they should be computationally inexpensive.
4. **Differentiable:** This is a necessary requirement for a function to work as activation function layer.

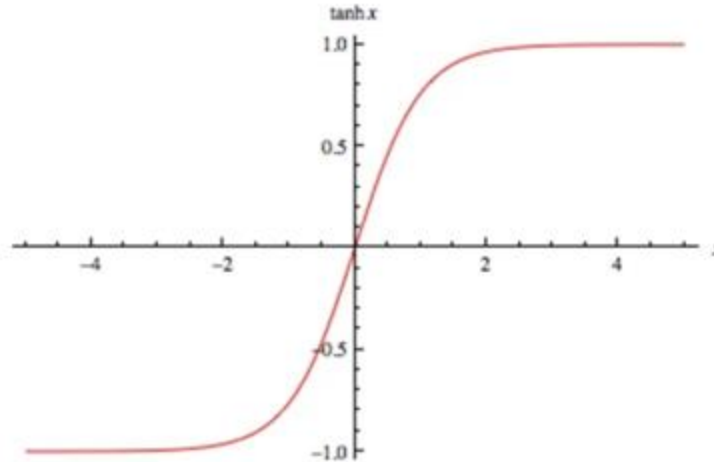
Activation function



This activation function is here only for historical reasons and never used in real models. It is computationally expensive, causes vanishing gradient problem and not zero-centred. This method is generally used for binary classification problems.

Activation function

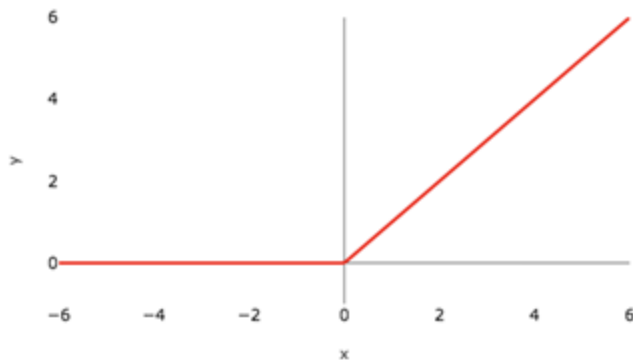
- **Tanh:** The tanh is defined as:



If you compare it to sigmoid, it solves just one problem of being zero-centred.

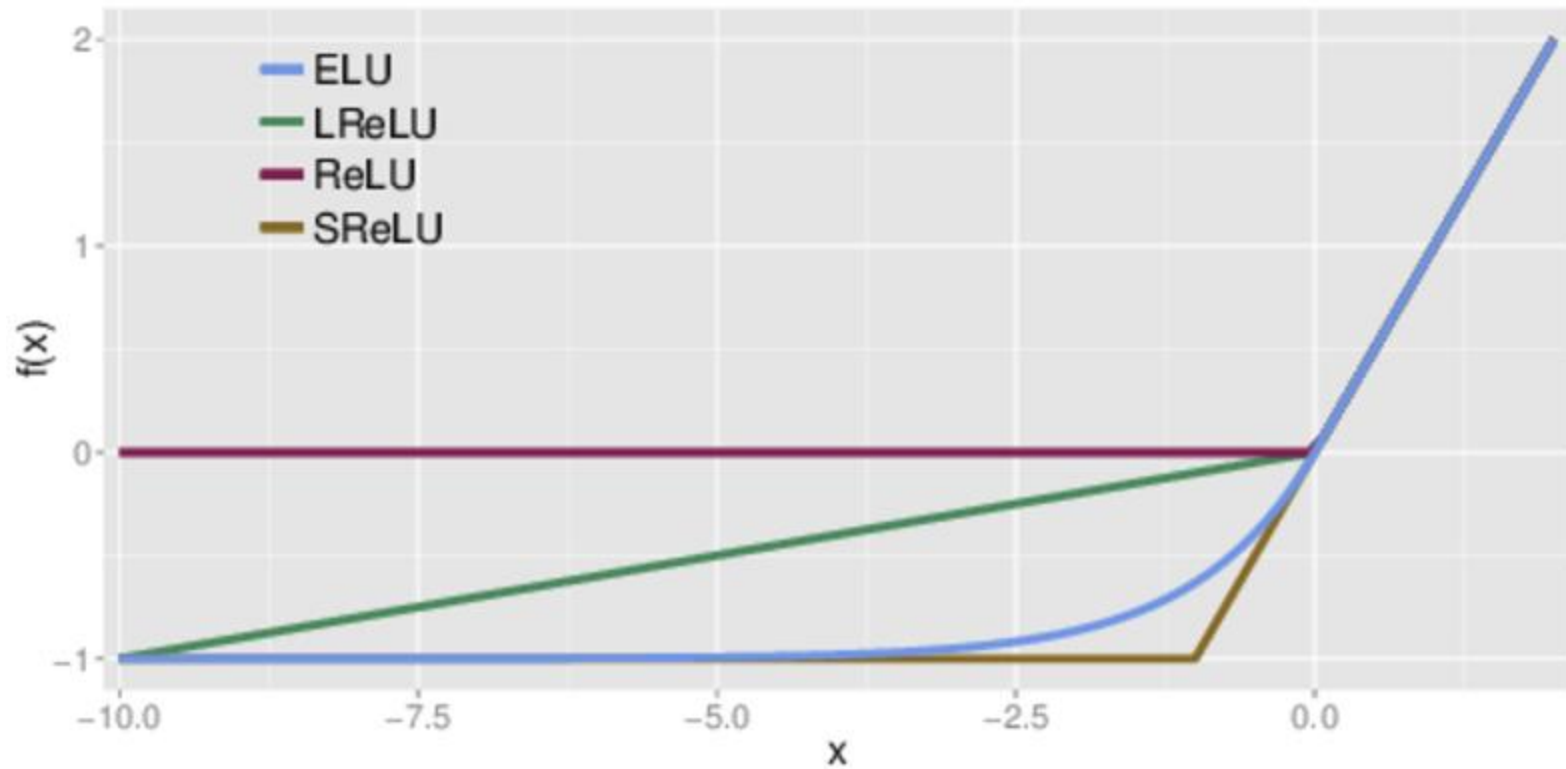
Activation function

- ReLU: ReLU (Rectified Linear Unit) is defined as $f(x) = \max(0, x)$:



This is a widely used activation function, especially with Convolutional Neural networks. It is easy to compute and does not saturate and does not cause the Vanishing Gradient Problem. It has just one issue of not being zero centred. It suffers from “**dying ReLU**” problem. Since the output is zero for all negative inputs. It causes some nodes to completely die and not learn anything.

Activation function



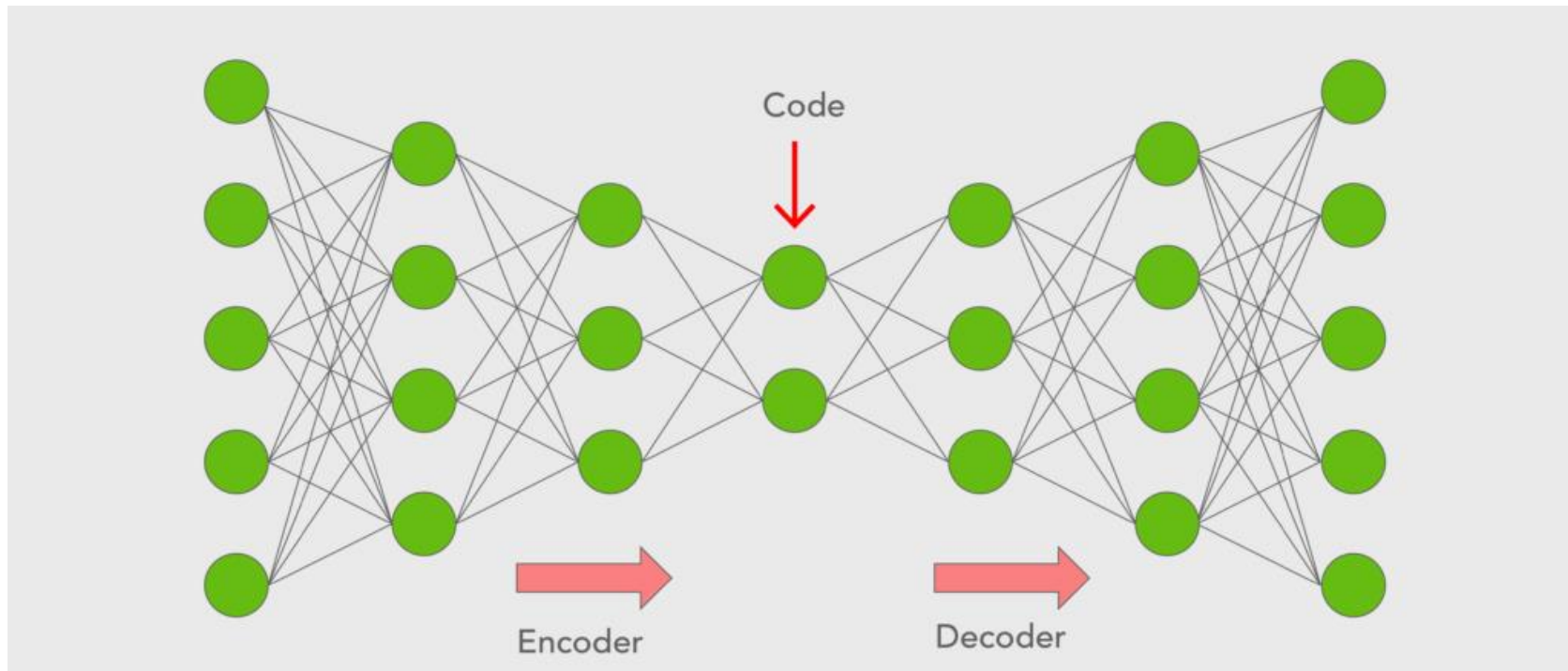
Data is key

- Hidden in most project as you usually start with data. Huge dataset have allowed DL to take off (Imagenet)
- Also, there is the annotation challenge (example with the ADNI project) say in healthcare
- Data augmentation (transfer learning is another approach to cope with the sparsity of data)
- Data can be weighted

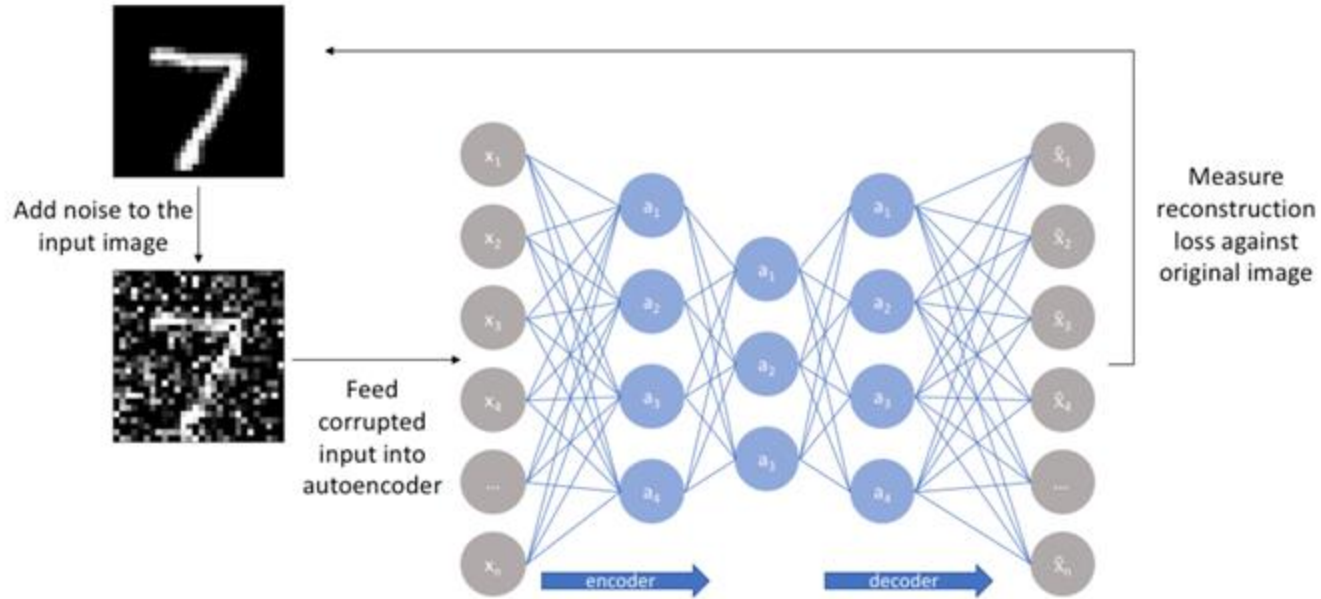
Loss functions (reminder)

- Binary classification
 - Category classification
 - Regression
-
- Unsupervised training...

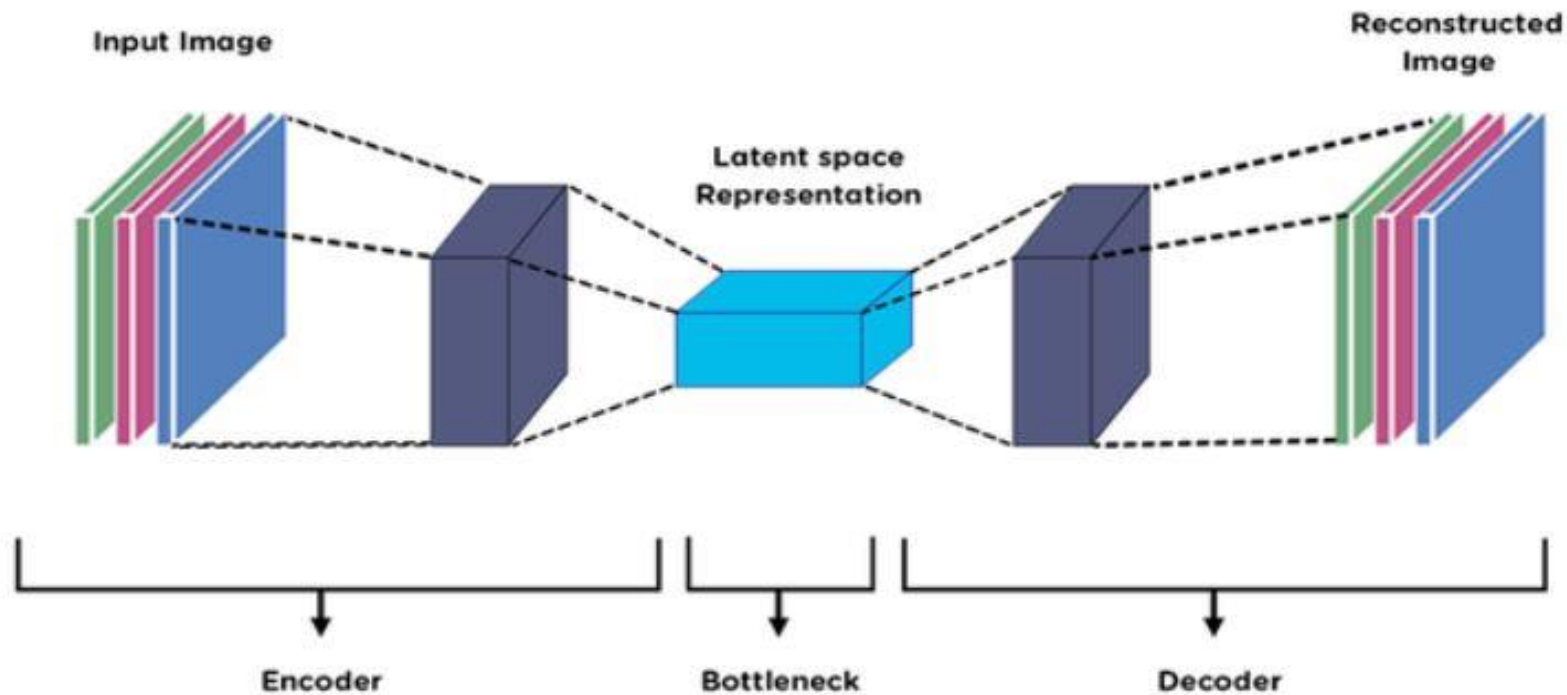
Auto-encoder



Encoder-decoder



Auto-encoder

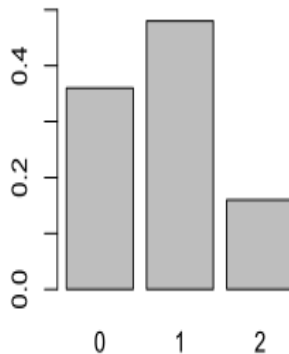


Kullback-Leibler

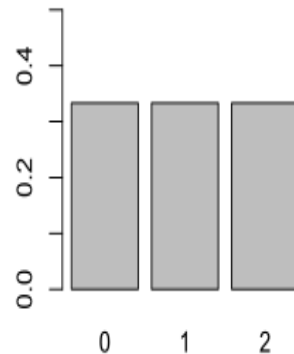
$$D_{\text{KL}}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

$$D_{\text{KL}}(P\|Q) \geq 0$$

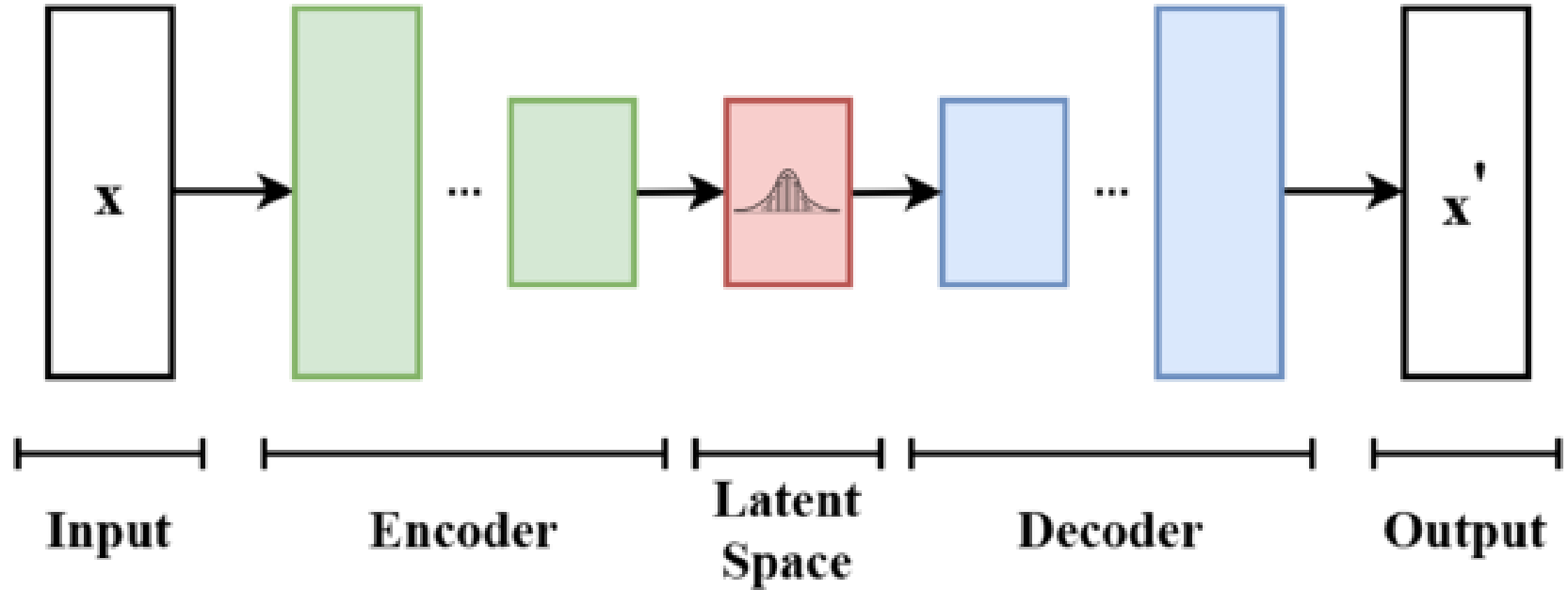
Distribution P
Binomial with $p = 0.4$, $N = 2$



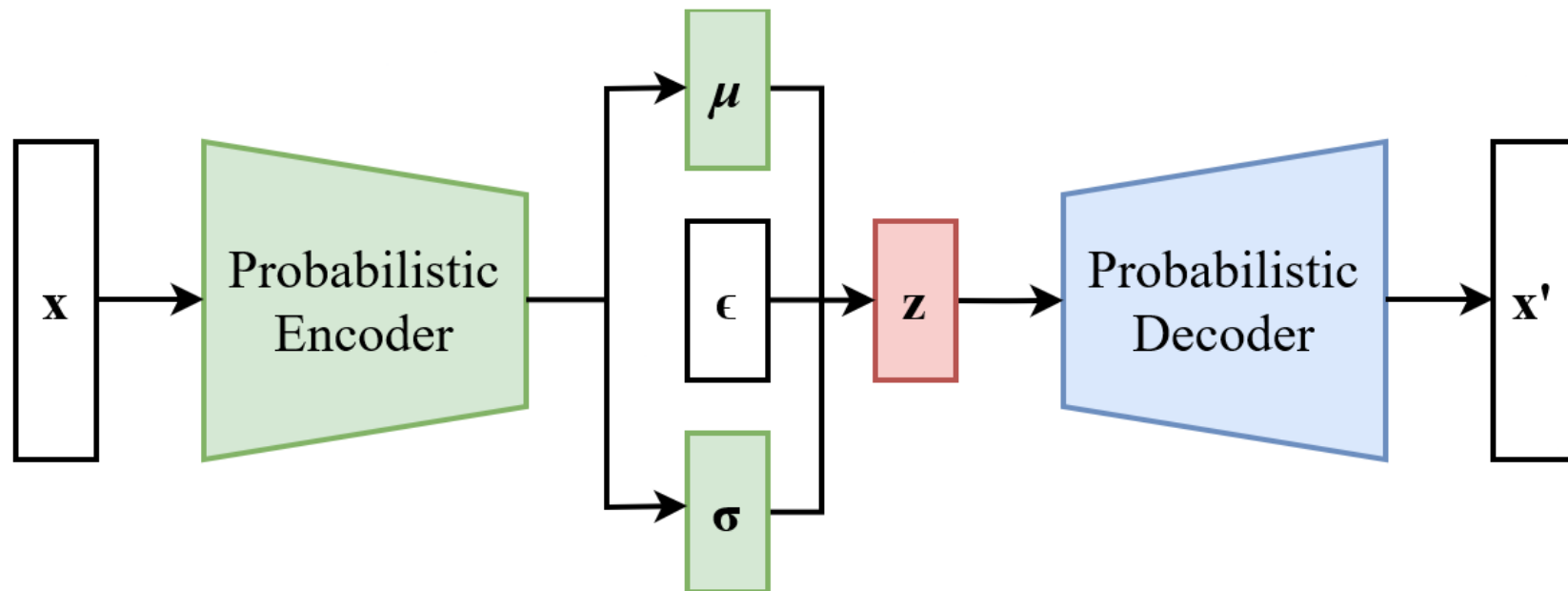
Distribution Q
Uniform with $p = 1/3$



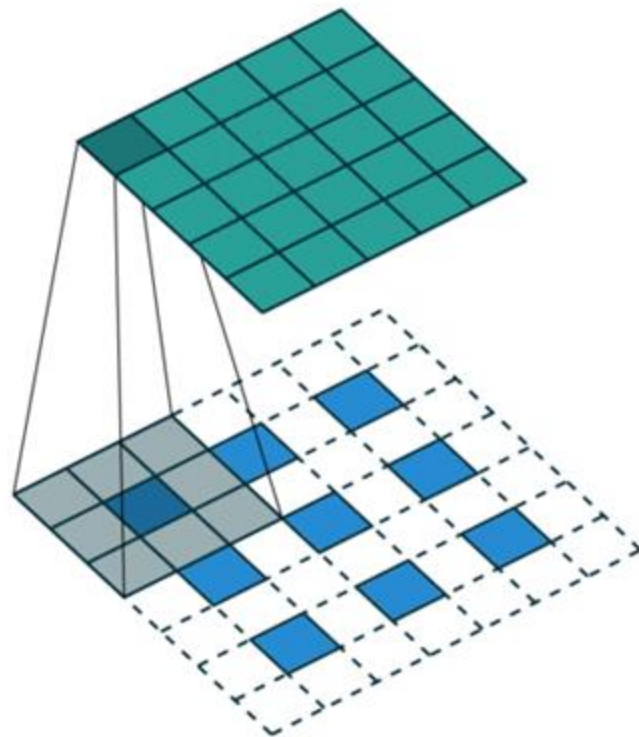
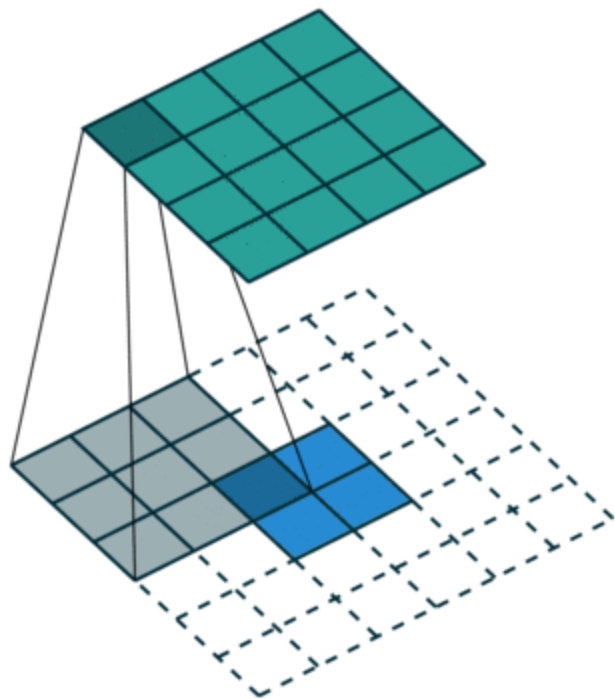
Variational Auto-Encoder (VAE)



VAE



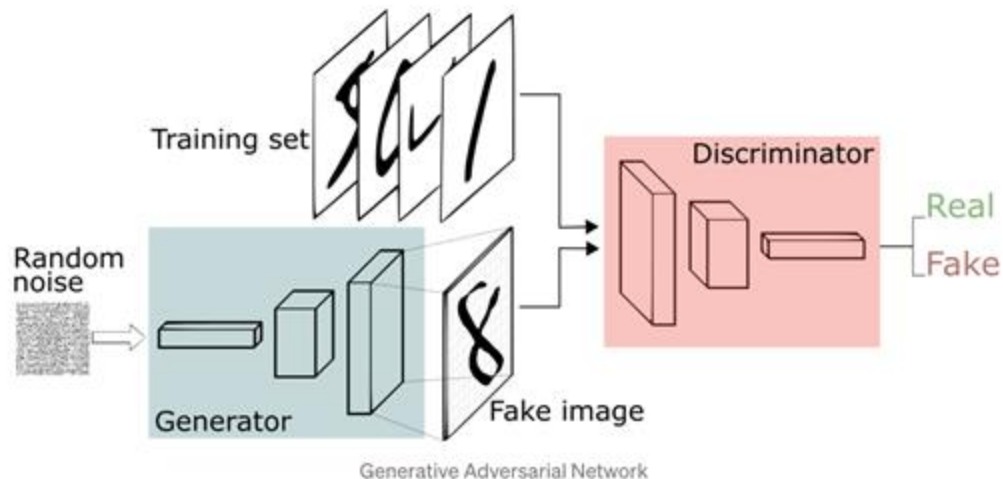
Encoder-decoder



Generative model

- One way to initialize or pre-train a model for a generative model for a GAN

GAN

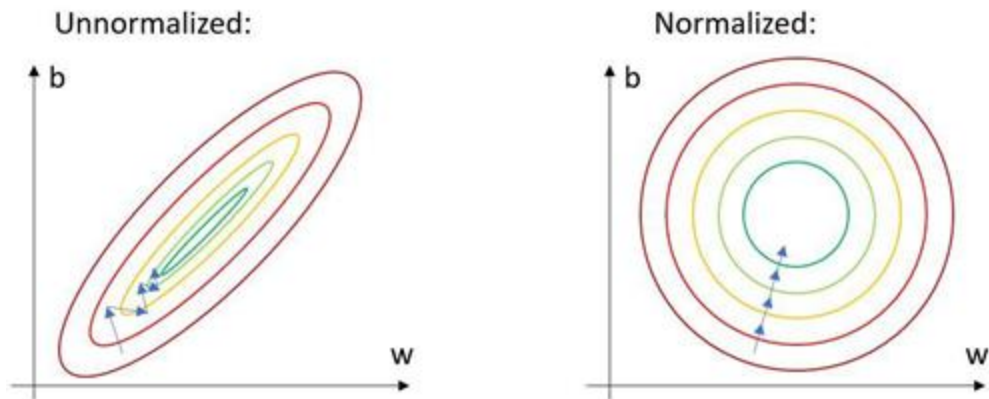


$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$

Summary

1. History
2. Classic CV
3. ML Basics
4. From a neuron to a neural net
5. Backpropagation
6. Activation functions
- 7. Batch normalization**
8. DNN, zoo

Batch normalization



BN, quote

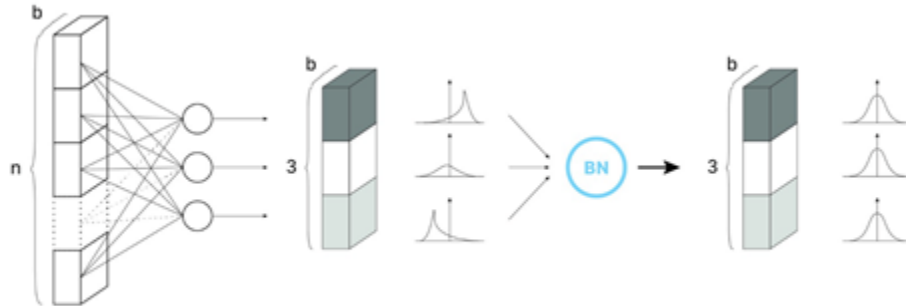
“Even if we don’t understand all the underlying mechanisms of Batch Normalization yet, there’s something everyone agrees on: it works.”

BN, algorithm (2015)

At each hidden layer, Batch Normalization transforms the signal as follow:

$$(1) \mu = \frac{1}{n} \sum_i Z^{(i)} \quad (2) \sigma^2 = \frac{1}{n} \sum_i (Z^{(i)} - \mu)^2$$

$$(3) Z_{norm}^{(i)} = \frac{Z^{(i)} - \mu}{\sqrt{\sigma^2 - \epsilon}} \quad (4) \tilde{Z} = \gamma * Z_{norm}^{(i)} + \beta$$



Batch Normalization first step. Example of a 3-neurons hidden layer, with a batch of size b . Each neuron follows a standard normal distribution. | Credit : author - Design : [Lou HD](#)

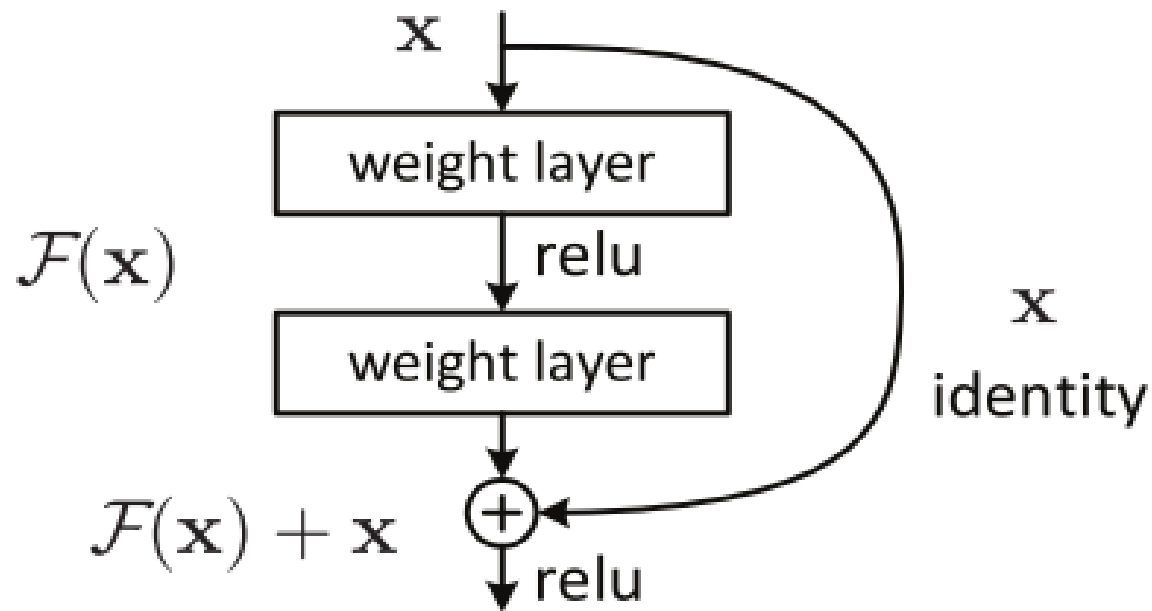
BN (Batch Normalisation)

- *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*
- Better learning by reducing the drift coming from learning and that would affect next layers to take it into account.
- In fact, the explanation was the motivation of the researchers
- Regularisation effect and avoid the vanishing gradient issue when increasing the learning rates
- In practice, the training phase and the inference phase have to be signaled
- To my knowledge, any large neural network use it.

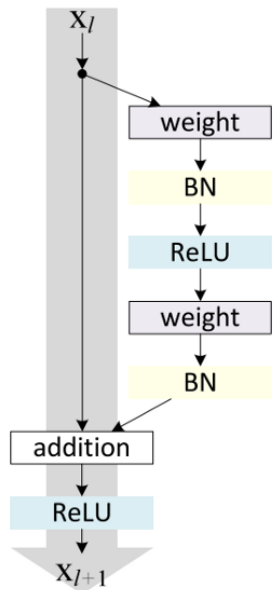
Vanishing (exploding) gradient

- A big issue for a long time, we (the computer scientists, the neural network community) did not know how to train very deep neural networks without having this issue
- Sigmoid to Relu ($\max(0, x)$)
- Initialisation (initialization)
- Resnet blocks
- Batch norm
- In practice if you are using an established architecture, you will not have big issues, however the reasons are quite intricate and results of the many choices and improvements done while the architecture was developed.

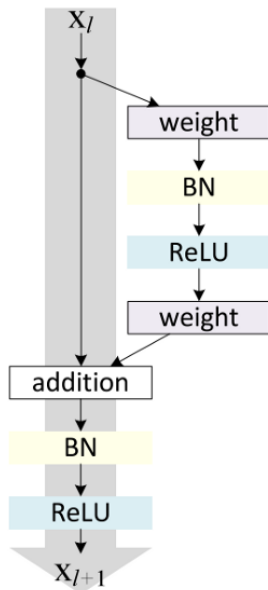
Resnet block



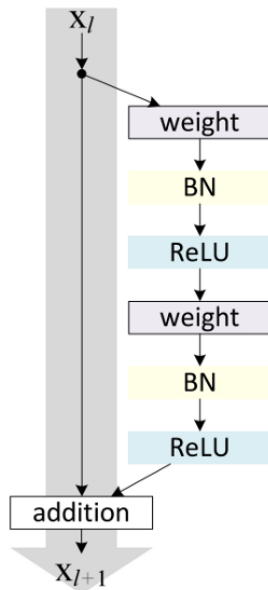
ResNet variants



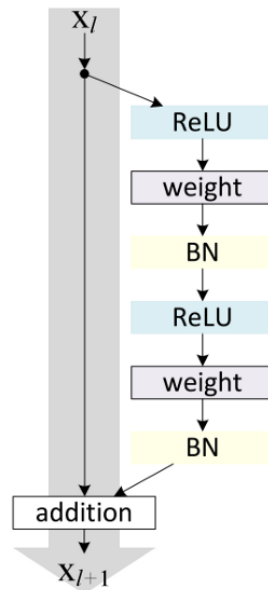
(a) original



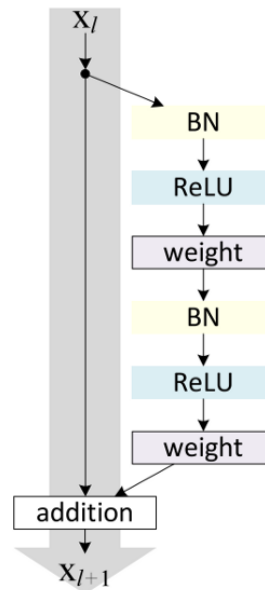
(b) BN after
addition



(c) ReLU before
addition



(d) ReLU-only
pre-activation



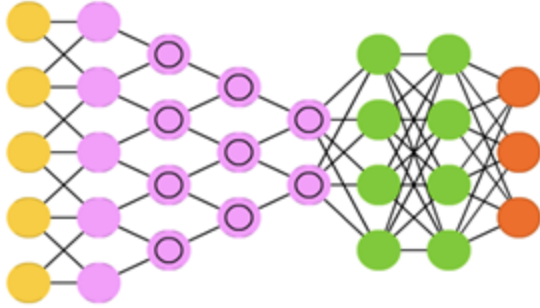
(e) **full pre-activation**

Summary

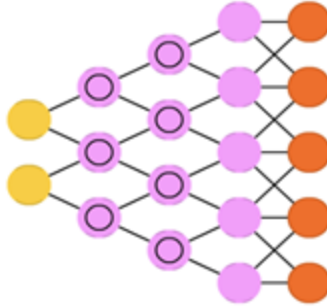
1. History
2. Classic CV
3. ML Basics
4. From a neuron to a neural net
5. Backpropagation
6. Activation functions
7. Batch normalization
- 8. DNN, zoo**

DNN, zoo

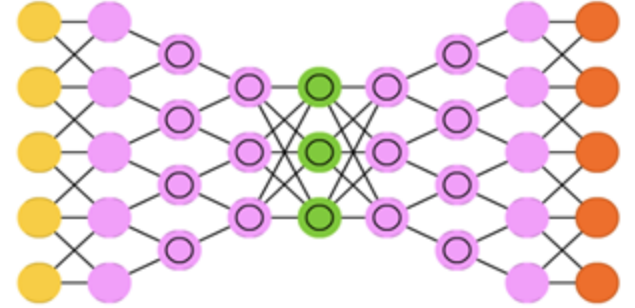
Deep Convolutional Network (DCN)



Deconvolutional Network (DN)



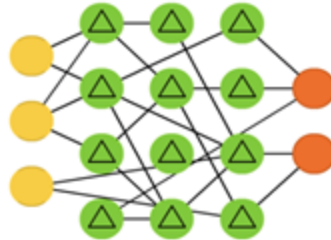
Deep Convolutional Inverse Graphics Network (DCIGN)



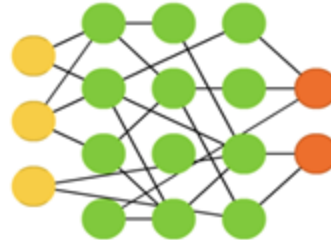
Generative Adversarial Network (GAN)



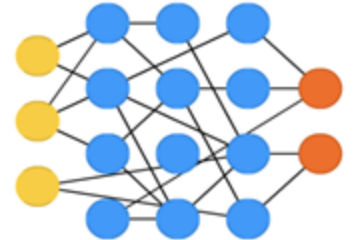
Liquid State Machine (LSM)



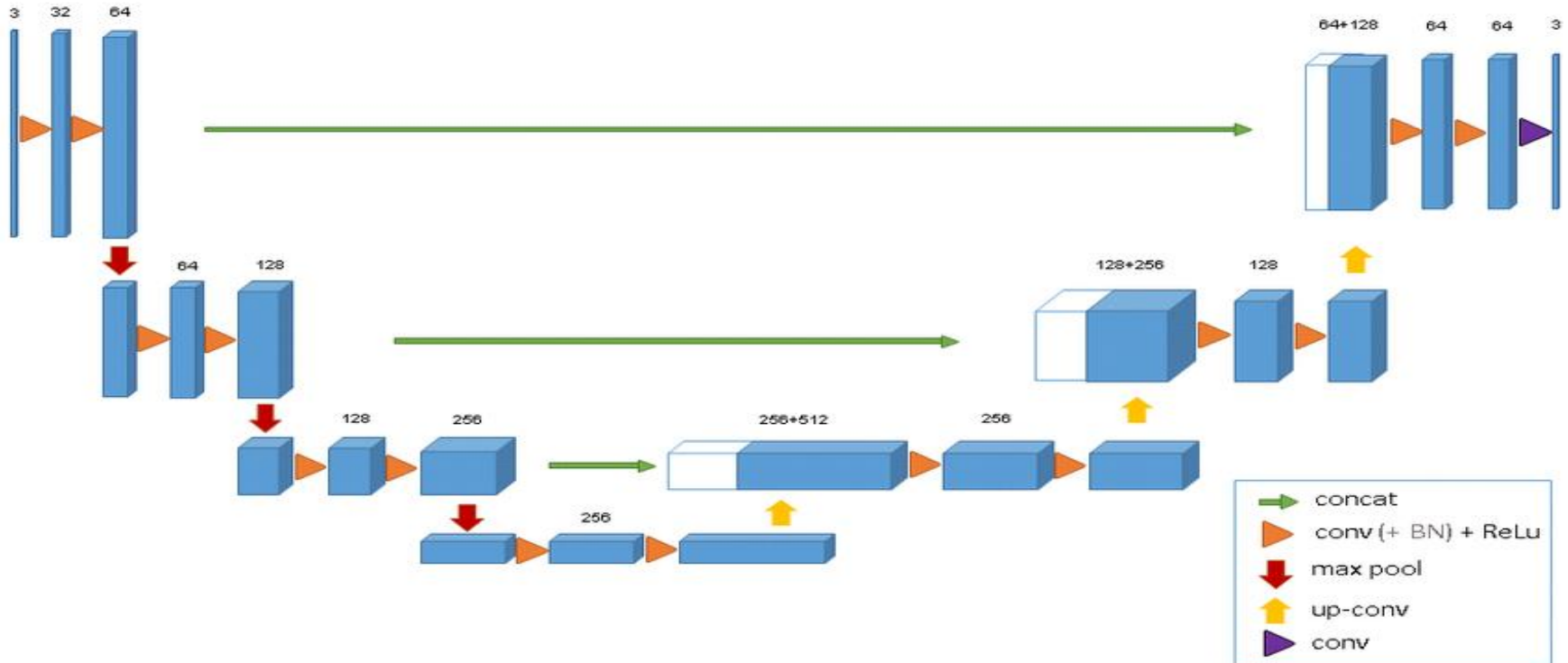
Extreme Learning Machine (ELM)



Echo State Network (ESN)



U-Net for image segmentation (medical application for instance)



Bounding Box YOLO



Final detections

Object Detection, YOLO

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

Comments on the loss function

- 20 classes
- 5 parameters per bounding boxes: x, y, w, h, c (confidence)
- 2 bounding box per cell
- 1_i : I cells in the image, 1_{ij} : j bounding box
- Depending on your interpretation there will be small variation in your implementation, since the code is open source it is possible to check by oneself the details chosen.
- $P_i(c)$ depends on the IoU

The architecture

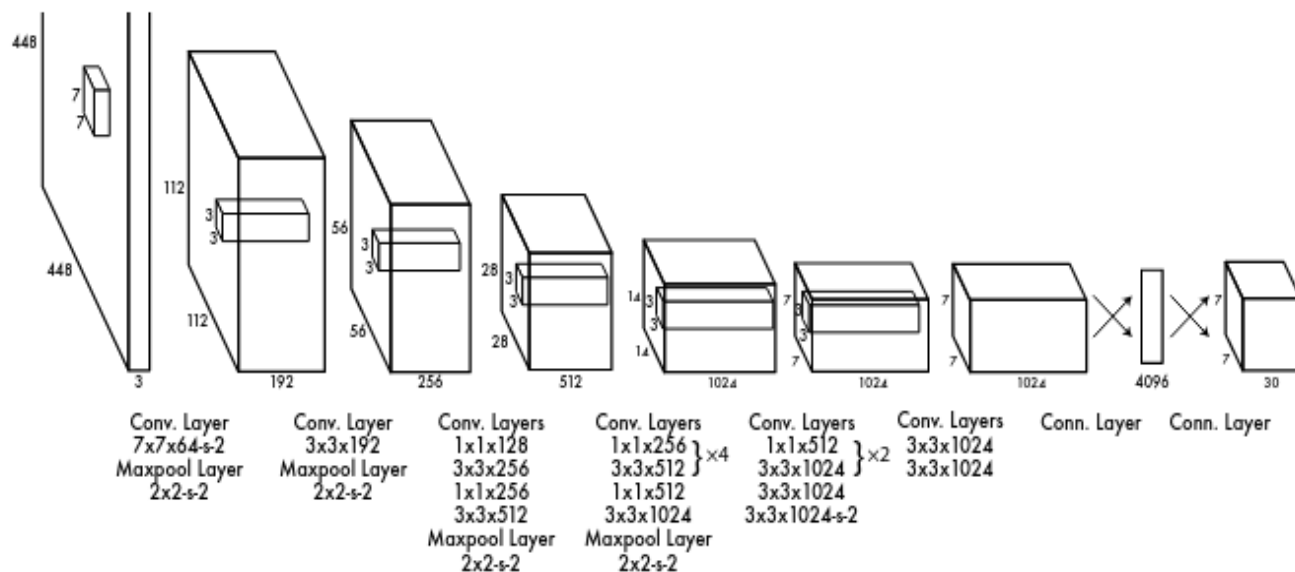


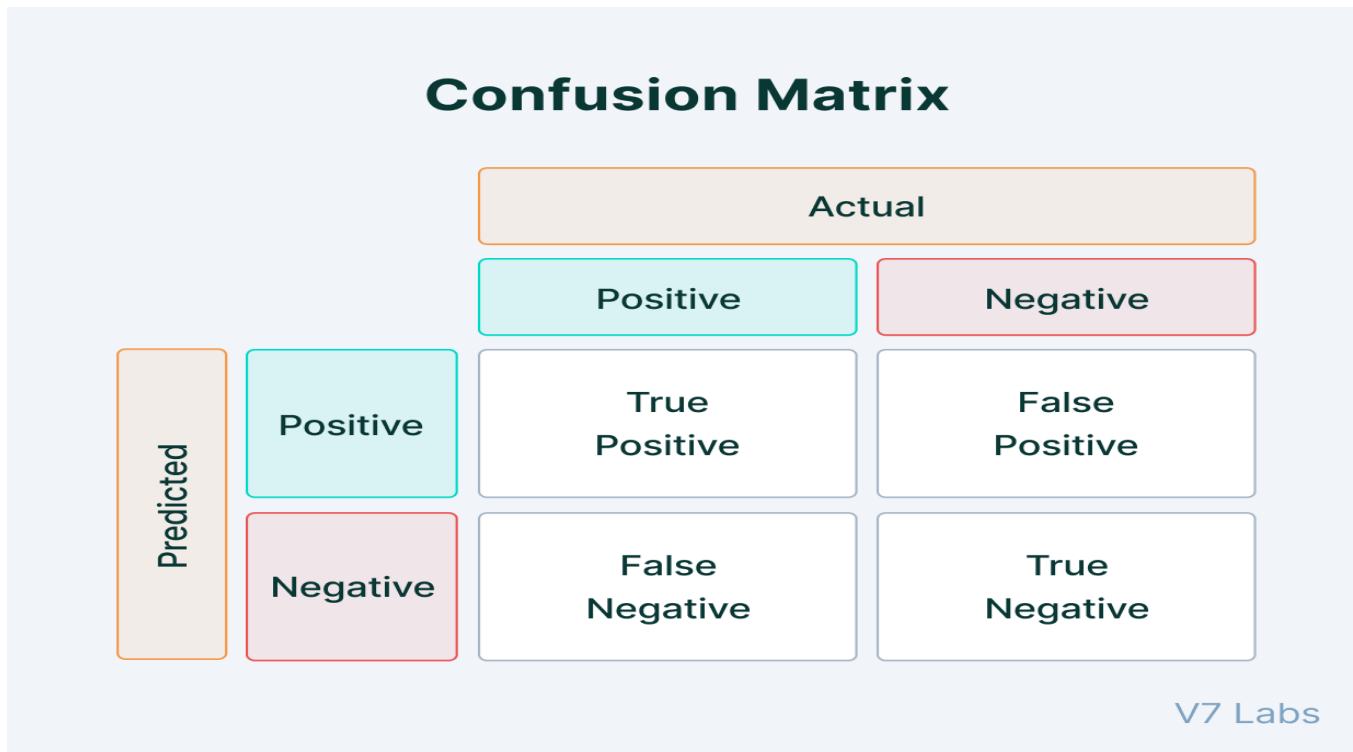
Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

How about the perceptive fields of the cells?

- Conv layers and pooling layers
- In this architecture (version 1) we do not have multiple following conv layers.

How to evaluate the performance of your model?

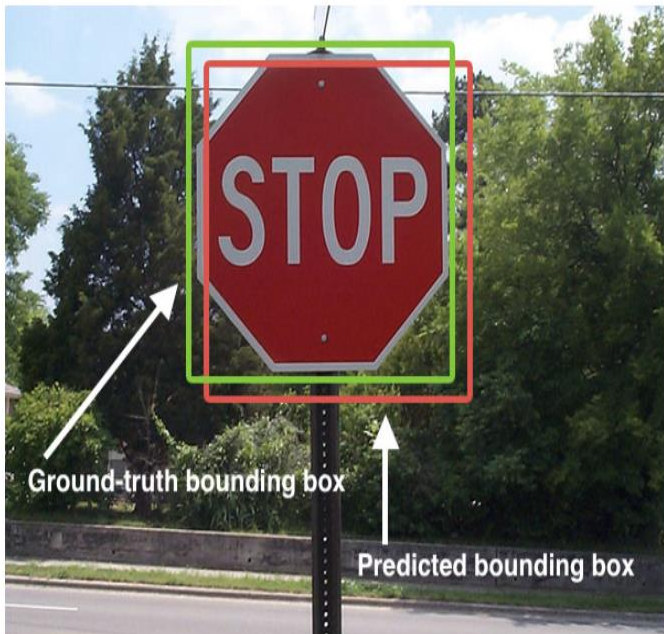
Confusion matrix




Precision Recall (Sensitivity, Specificity)

$$\text{Precision} = \frac{tp}{tp + fp}$$
$$\text{Recall} = \frac{tp}{tp + fn}$$

IoU Intersection over Unions

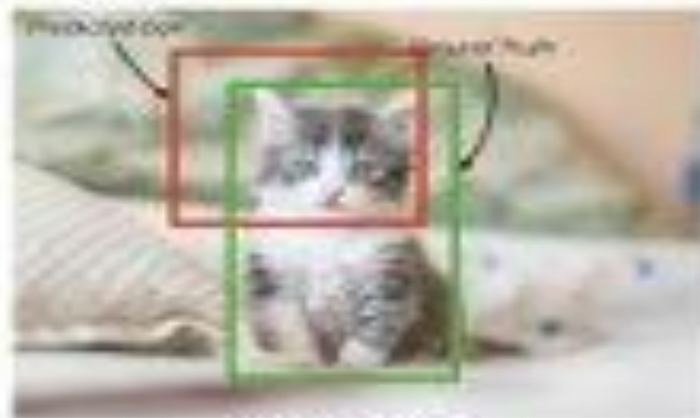


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


Precision and IoU threshold

If IoU threshold = 0.5

False Positive (FP)



$\text{IoU} = 0.3$

True Positive (TP)



$\text{IoU} = 0.7$

MAP (mean average precision)

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k$$

AP_k = the AP of class k

n = the number of classes

Bonus

- GAN: colab with code available [here](#)
- VAE: colab with code available (generative, self supervised) [here](#)
- ViT examples with pytorch available, (from NLP) [here](#)
- UNet (segmentation)
- Diffusion models
- Any other architecture you'd like to know more about?

An interpretation of the negative log likelihood from KL

5.8 Cross-entropy loss

In this chapter, we developed loss functions based on minimizing negative log-likelihood. However, it is common in the literature to refer to *cross-entropy* losses. In this section, we'll introduce the cross-entropy loss and show that it is equivalent to using negative log-likelihood.

ix B.1.4
ergence

The cross-entropy loss is based on the idea of finding parameters θ that minimize the distance between the empirical distribution $q(y)$ of the observed data y and a model distribution $Pr(y|\theta)$ (figure 5.11). The ‘distance’ between two probability distributions $q(z)$ and $p(z)$ can be evaluated using the **Kullback-Leibler (KL) divergence**:

$$\text{KL}[q||p] = \int_{-\infty}^{\infty} q(z) \log[q(z)] dz - \int_{-\infty}^{\infty} q(z) \log[p(z)] dz. \quad (5.28)$$

Now consider that we observe an empirical distribution of data at points $\{y_i\}_{i=1}^I$. We can describe this as a weighted sum of point masses:

And issues of the probabilistic framework

$$q(y) = \frac{1}{I} \sum_{i=1}^I \delta[y - y_i], \quad (5.29)$$

where $\delta[\bullet]$ is the **Dirac delta function**. We want to minimize the KL-divergence between the model distribution $Pr(y|\boldsymbol{\theta})$ and this empirical distribution:

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\int_{-\infty}^{\infty} q(y) \log[q(y)] dy - \int_{-\infty}^{\infty} q(y) \log [Pr(y|\boldsymbol{\theta})] dy \right] \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[- \int_{-\infty}^{\infty} q(y) \log [Pr(y|\boldsymbol{\theta})] dy \right], \end{aligned} \quad (5.30)$$

where the first term disappears as it has no dependence on $\boldsymbol{\theta}$. The remaining second term is known as the *cross-entropy*. It can be interpreted as the amount of uncertainty that remains in one distribution after taking into account what we already know from the other. Now, we substitute in the definition of $q(y)$ from equation 5.29:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[- \int_{-\infty}^{\infty} \frac{1}{I} \sum_{i=1}^I \delta[y - y_i] \log [Pr(y|\boldsymbol{\theta})] dx \right]$$

From a draft book, but often in papers the mathematics are not rigorous like this

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \operatorname{argmin}_{\boldsymbol{\theta}} \left[- \int_{-\infty}^{\infty} \frac{1}{I} \sum_{i=1}^I \delta[y - y_i] \log [Pr(y|\boldsymbol{\theta})] dx \right] \\ &= \operatorname{argmin}_{\boldsymbol{\theta}} \left[- \frac{1}{I} \sum_{i=1}^I \log [Pr(y_i|\boldsymbol{\theta})] \right] \\ &= \operatorname{argmin}_{\boldsymbol{\theta}} \left[- \sum_{i=1}^I \log [Pr(y_i|\boldsymbol{\theta})] \right],\end{aligned}\tag{5.31}$$

where we have eliminated the constant scaling factor $1/I$ in the last line as this does not affect the position of the minimum.

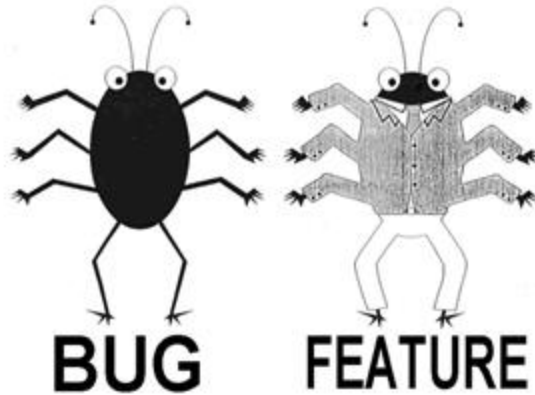
In machine learning, the distribution parameters $\boldsymbol{\theta}$ are computed by the model $\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]$, and so we have:

$$\hat{\boldsymbol{\phi}} = \operatorname{argmin}_{\boldsymbol{\phi}} \left[- \sum_{i=1}^I \log [Pr(y_i|\mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}])] \right].\tag{5.32}$$

This is exactly the negative log-likelihood criterion from the recipe in section 5.2.

It follows that the negative log-likelihood criterion (from maximizing the data likelihood) and the cross-entropy criterion (from minimizing the distance between the model and empirical data distributions) are equivalent.

Tutorial 6



TODO

1. TP: aws
2. TP: kaggle
3. TP: face recognition [article](#) + [code](#)
4. Aive exercice / presentation (bonus music => spleeter)

1. <https://labelerrors.com/>
2. [siamese network](#)
3. [kaggle sota technics](#)
4. create DL lib from scratch [video](#)
5. [pytorch lightning](#) - flash