A deeper view of Deep Learning

Mélanie Ducoffe, Frédéric Precioso

Laboratoire I3S - UNS CNRS UMR 7271

06/07/2016

Deep Learning in a nutshell

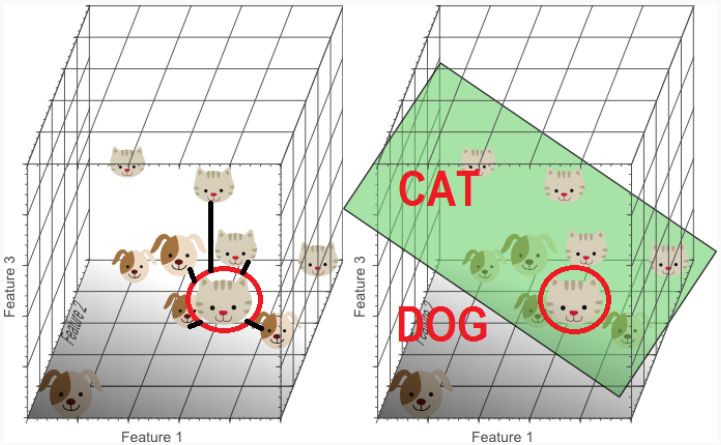
Several kinds of architectures

1. Convolutional Neural Network *(most popular, image)*
2. Auto encoder *(lack of annotated data, unsupervised representation)*
3. Recurrent neural network *(sequence, language processing)*
4. probabilistic models DBN, RBM ... *(speech recognition, music genre classification)*

Naive classification [Mallat, 2014]

- Task: classify images of dogs ( , 0) and cats( , 1)

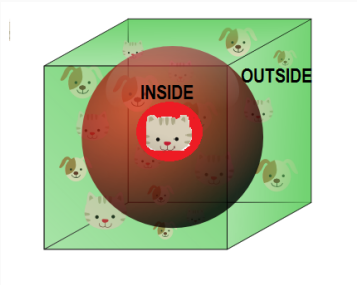
1. pictures are represented by d variables (pixels)
2. one picture is represented by a vector of size d
3. Decision based on the neighbors type
4. Separate the space and look on what side the sample is



The curse of dimensionality [Bellman, 1956]

- Euclidian distance is not relevant in high dimension: d $\geq$10
  1. look at the examples at distance at most r
  2. the hypersphere volume is too small: practically empty of examples

$$\frac{volume\ of\ the\ sphere\ of\ radial\ r}{hypersphere\ of\ 2r\ width} \rightarrow_{d\rightarrow\infty} 0$$



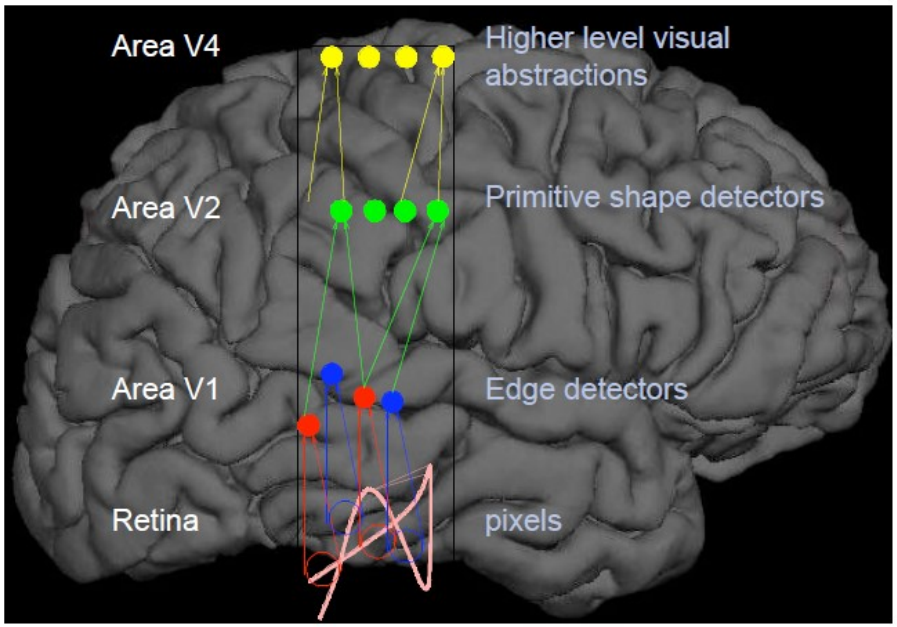
  3. need a number of examples exponential in d

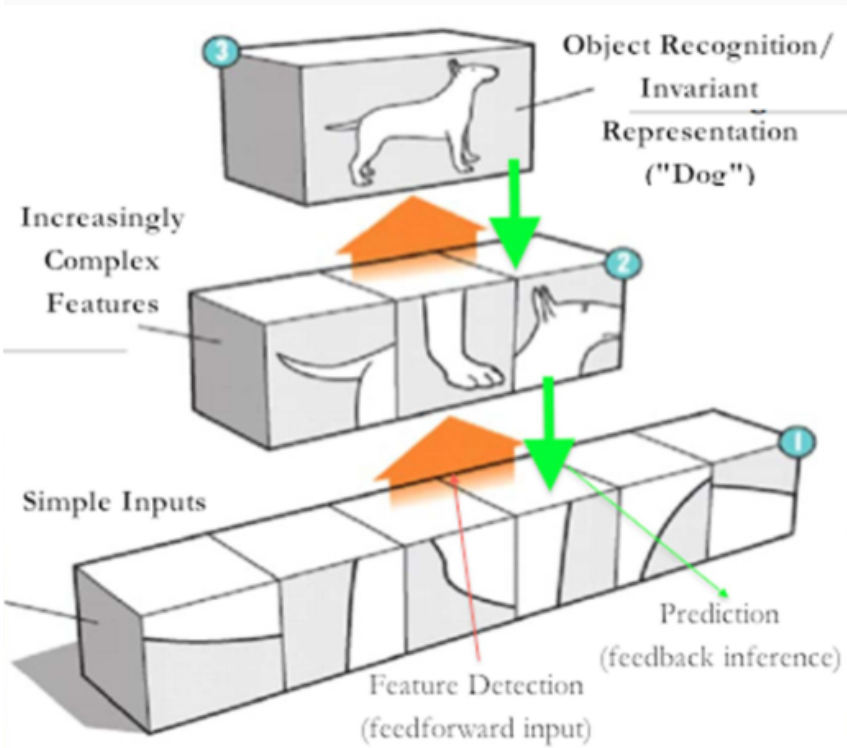
Remark

Specific care for data representation

Deep representation

Deep Architecture in the Brain





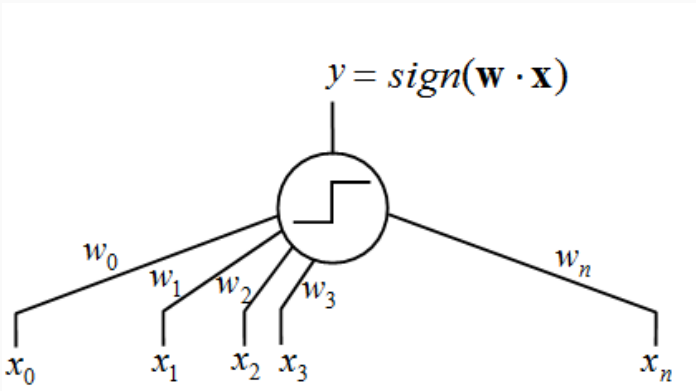
The perceptron property

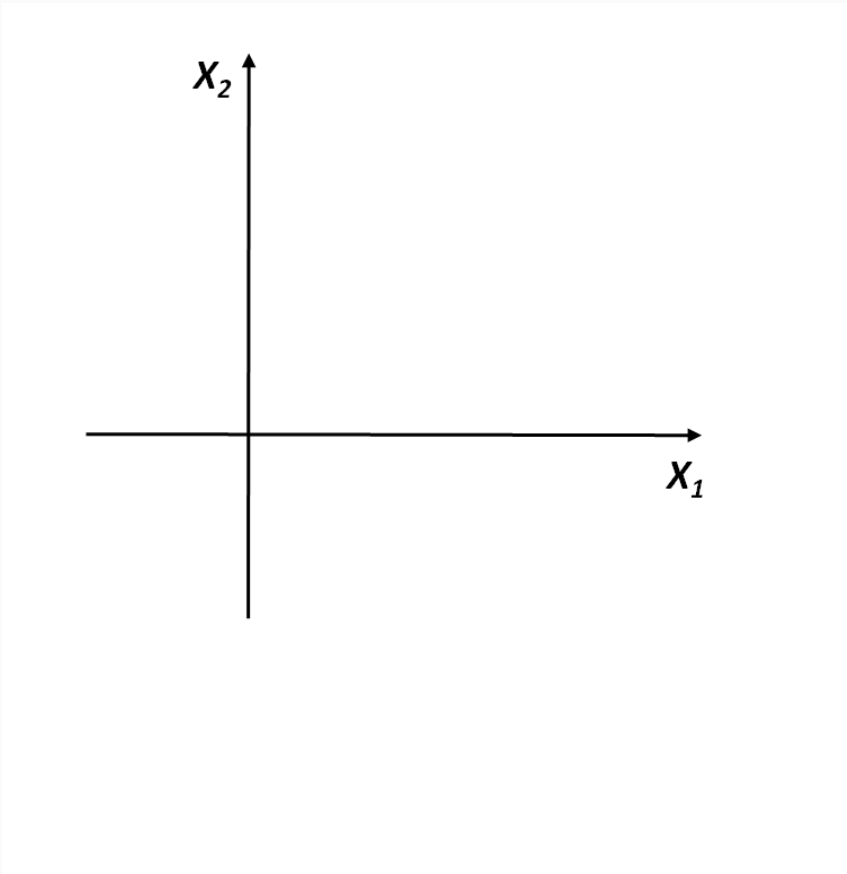
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





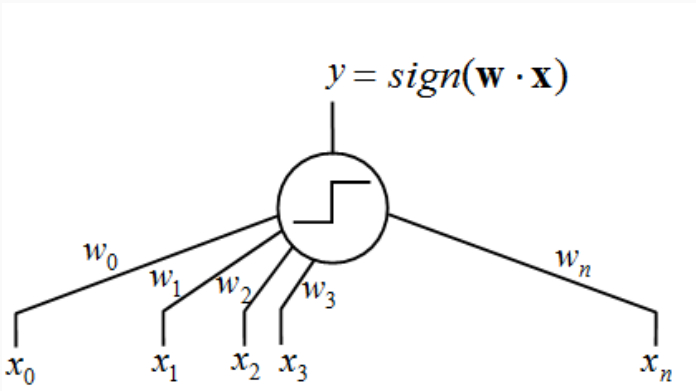
The perceptron property

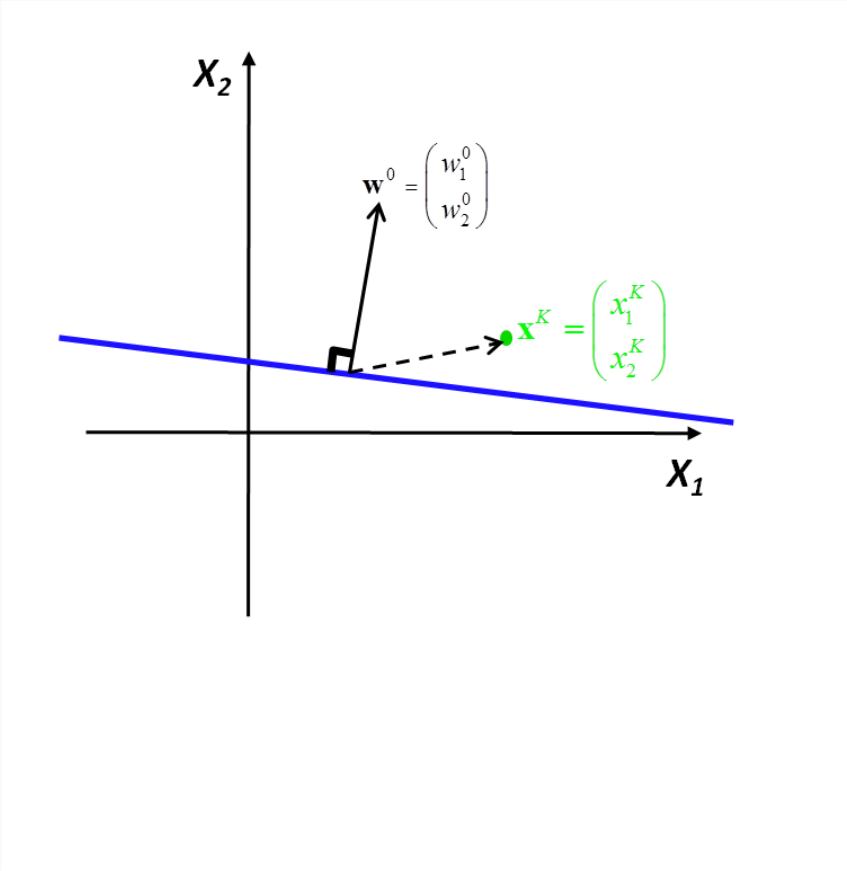
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





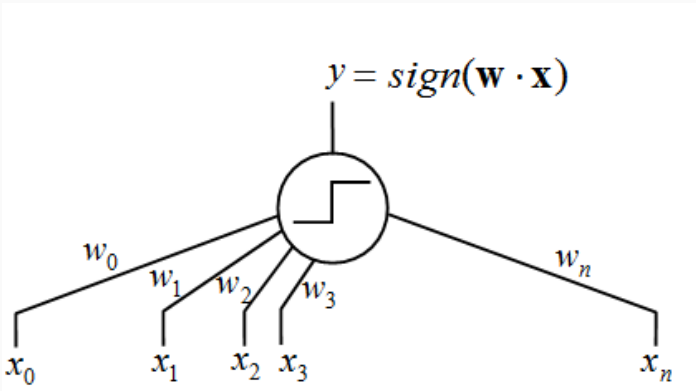
The perceptron property

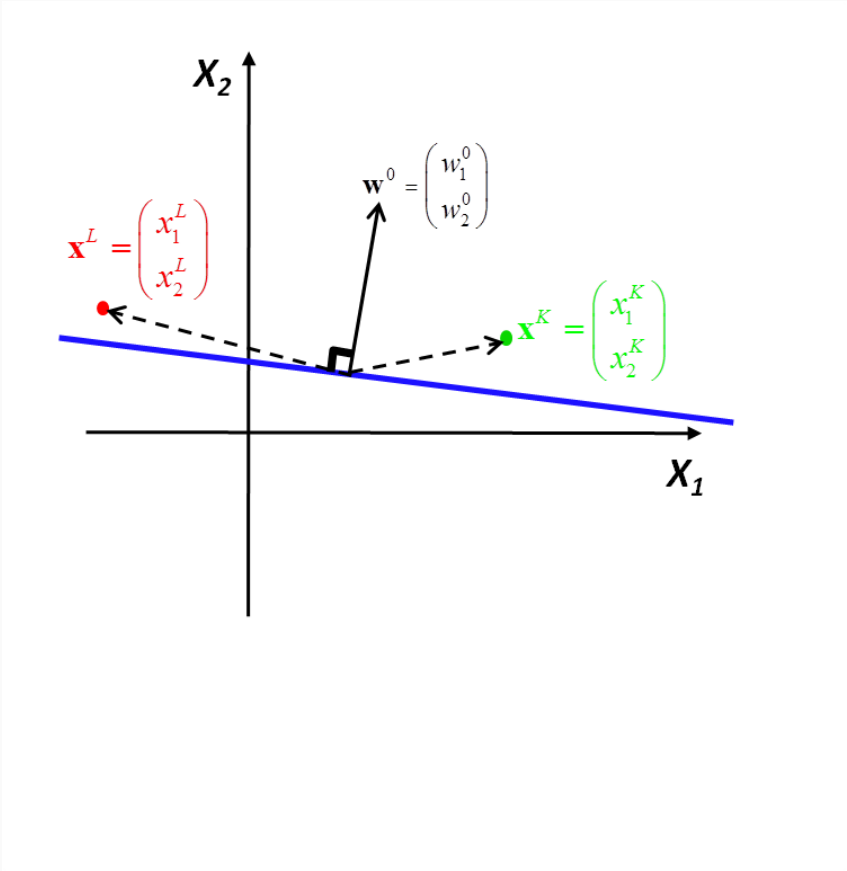
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





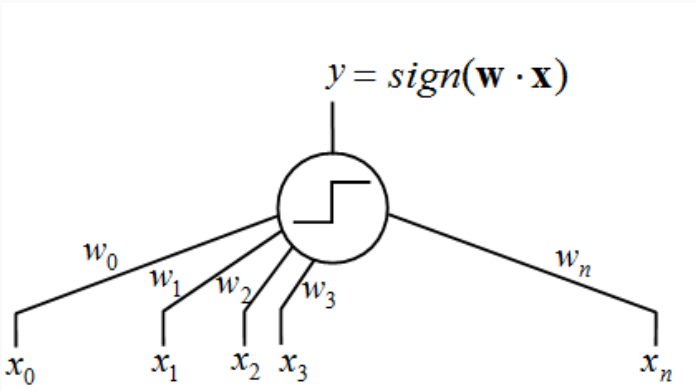
The perceptron property

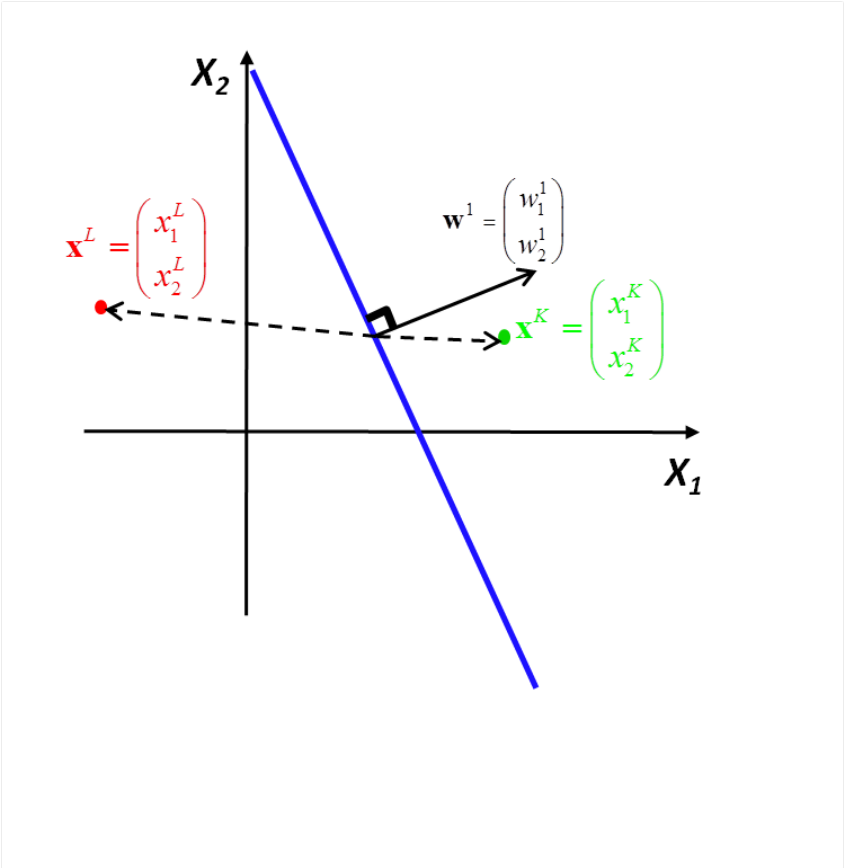
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





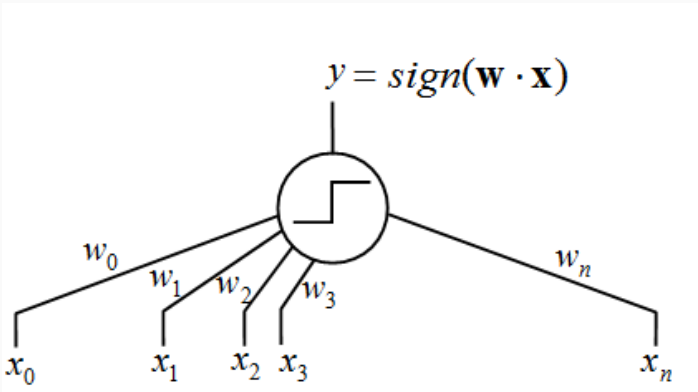
The perceptron property

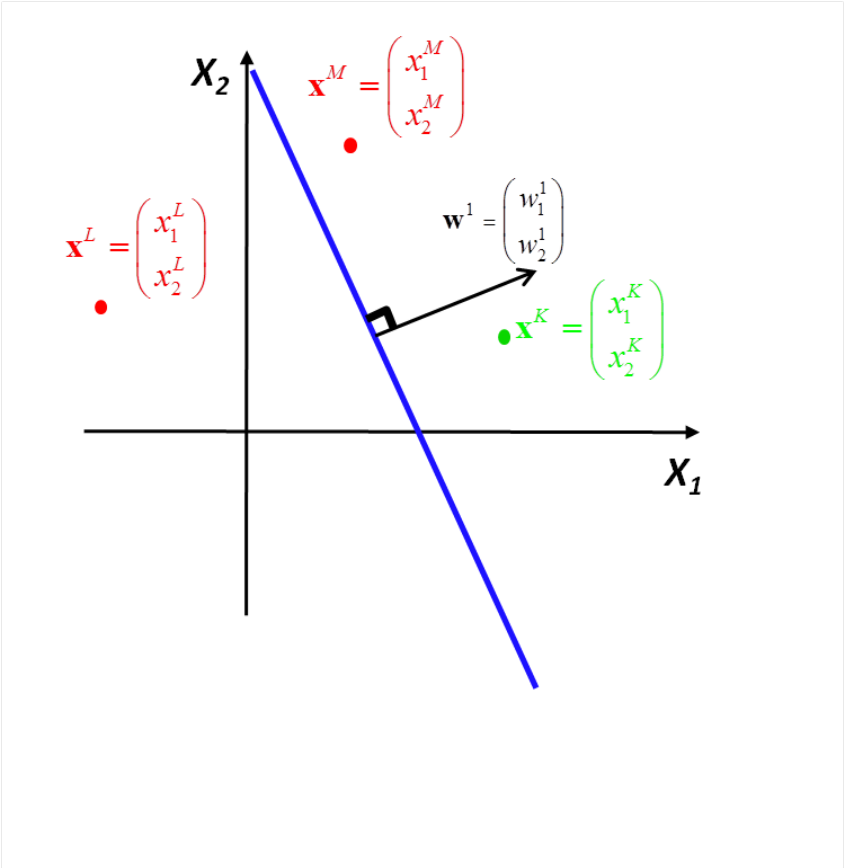
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





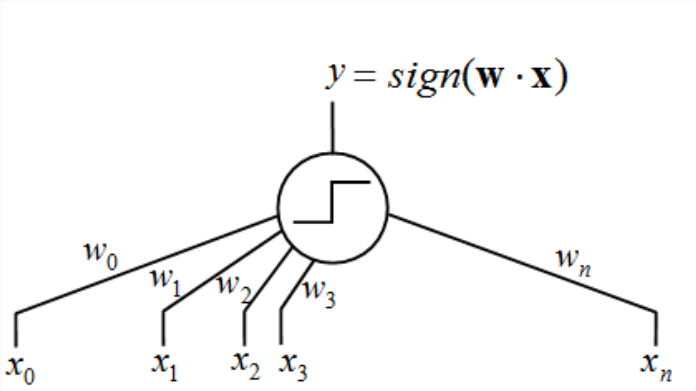
The perceptron property

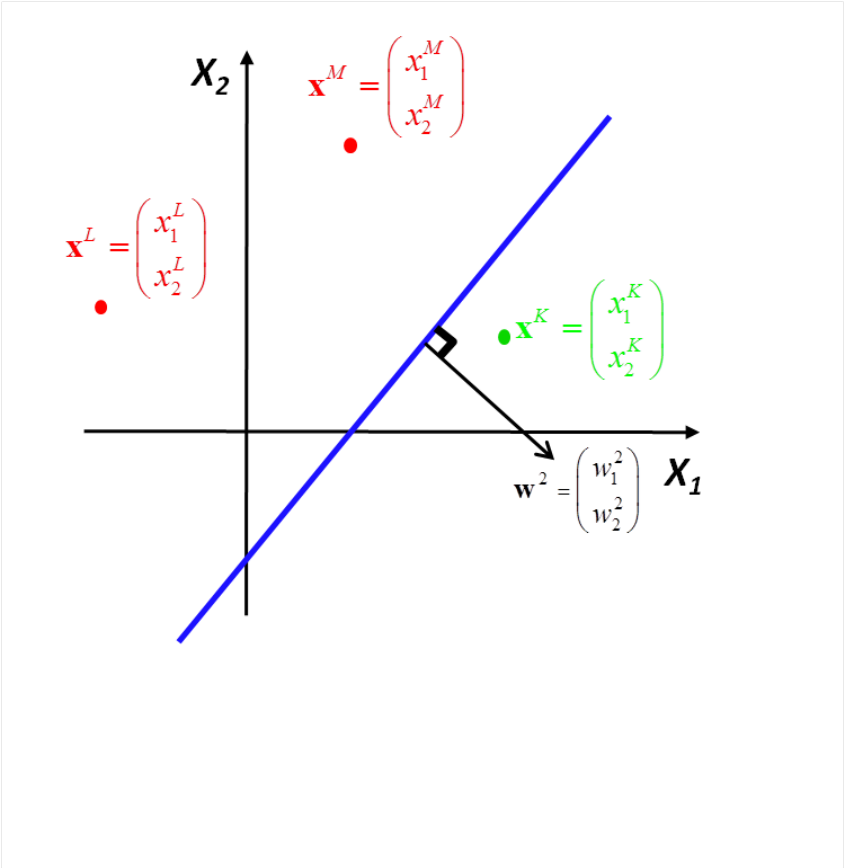
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





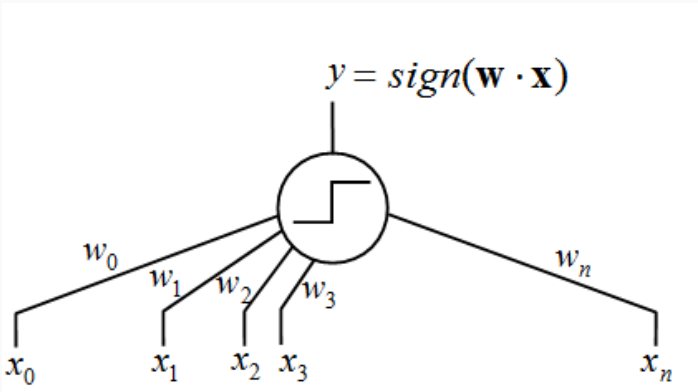
The perceptron property

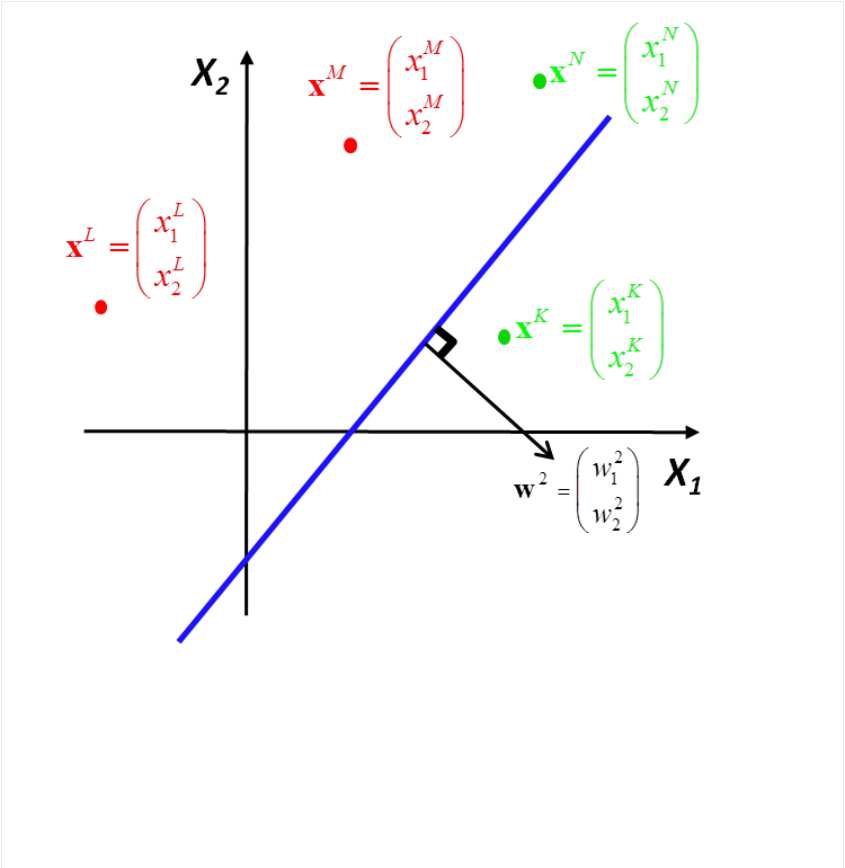
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





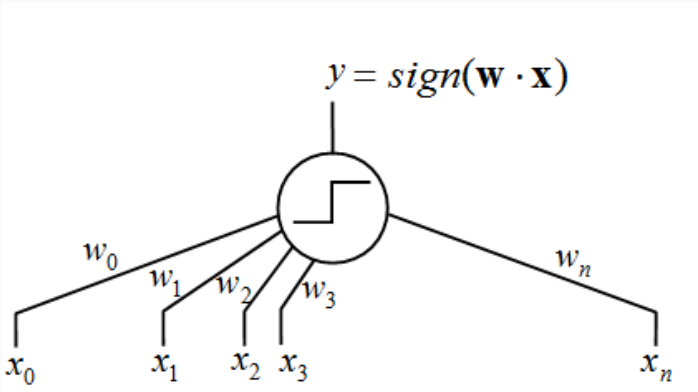
The perceptron property

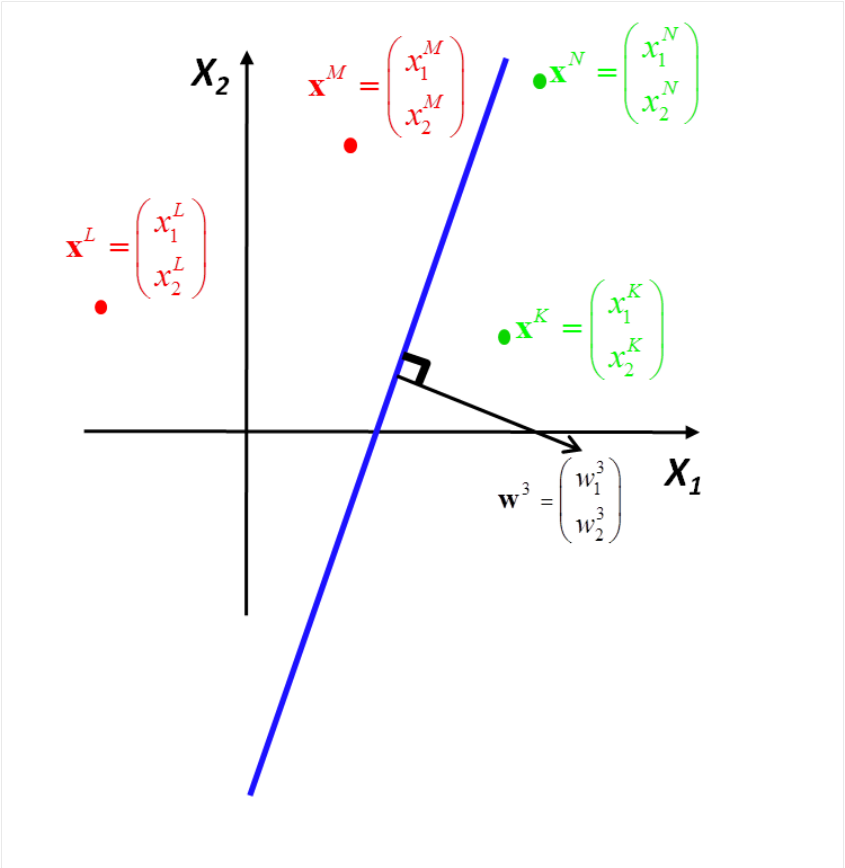
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





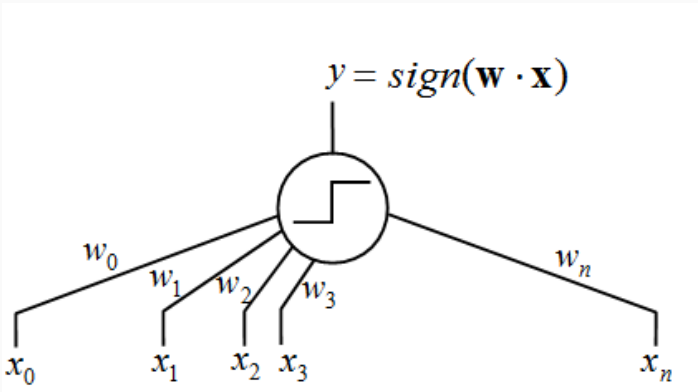
The perceptron property

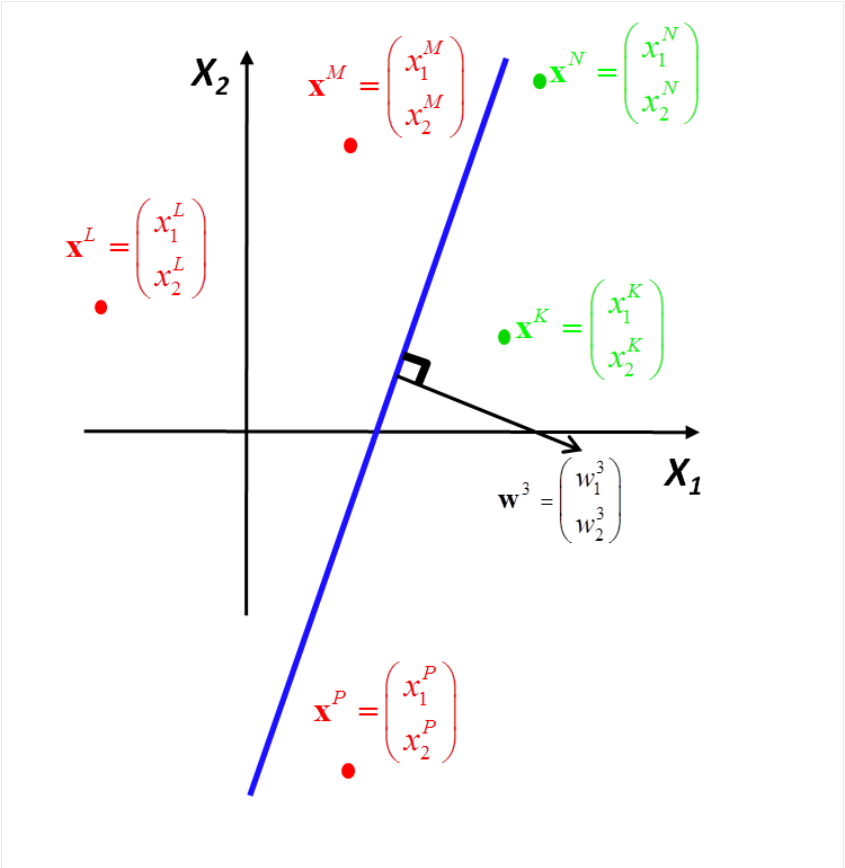
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





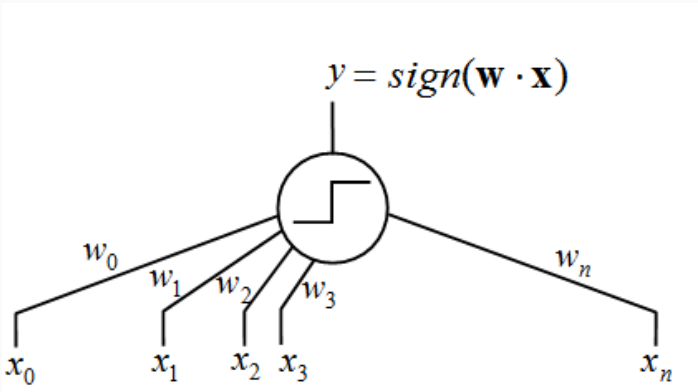
The perceptron property

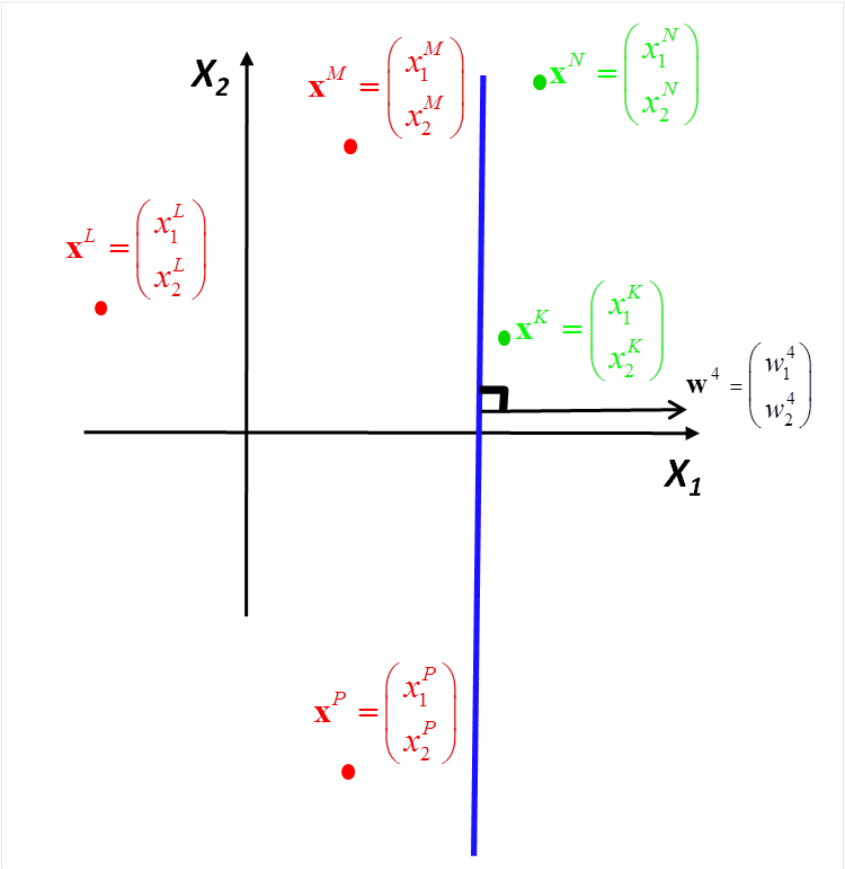
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$





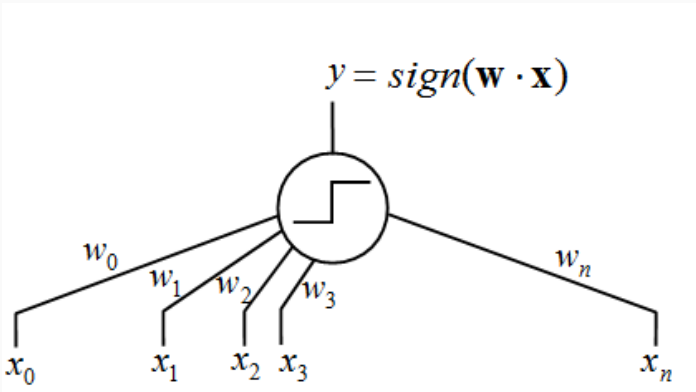
The perceptron property

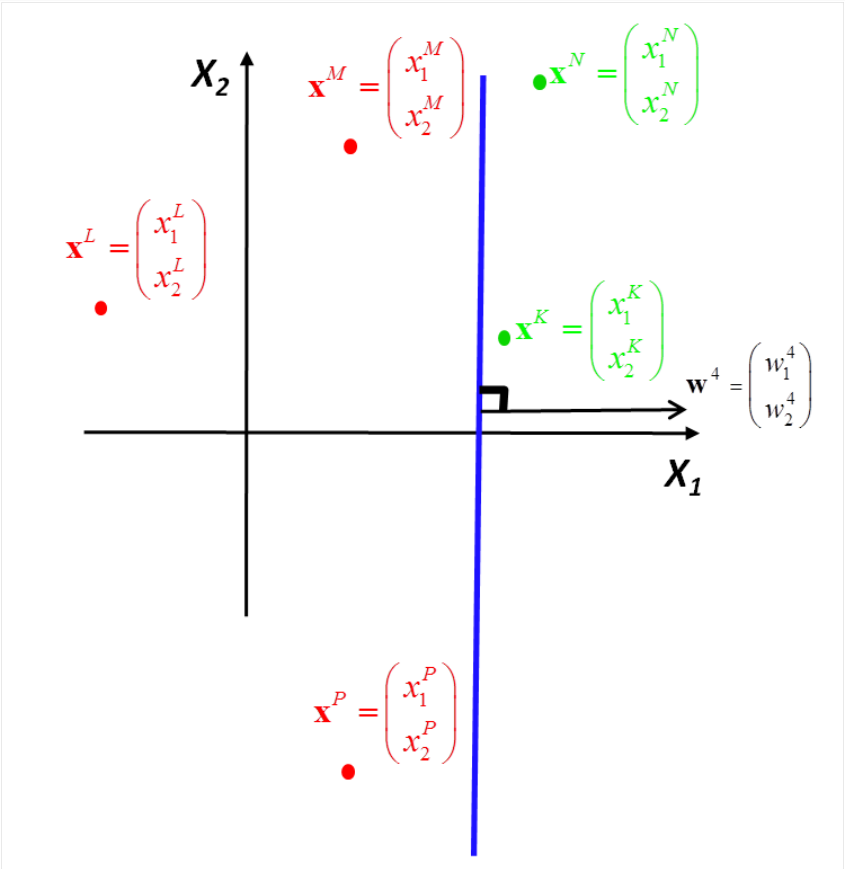
- Perceptron only learns linear function

hyperplane equation: $\mathbf{w}\cdot\mathbf{x}=0$

$\mathbf{w}\cdot\mathbf{x} = \sum_j w_j x_j$

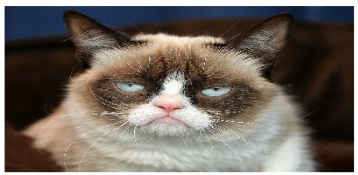
determine the class: $\text{sign}(\mathbf{w}\cdot\mathbf{x})$

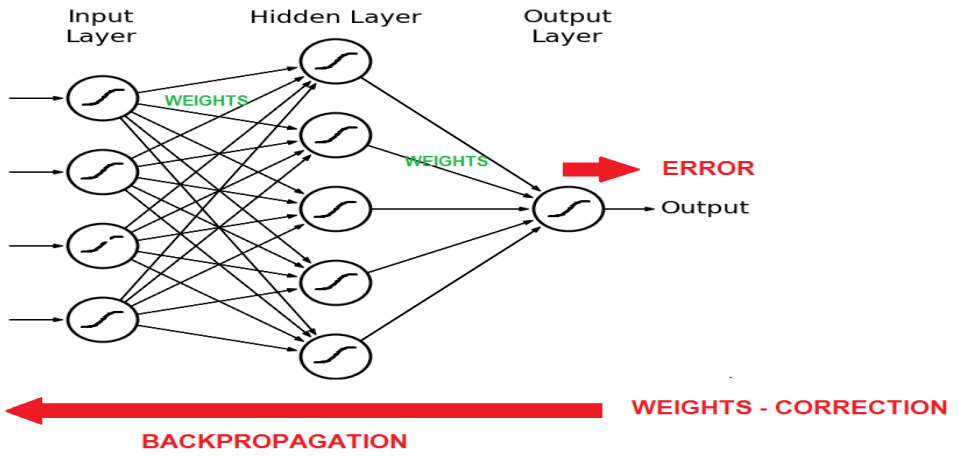




Neural Network [Rumelhart et al., 1985, Lecun, 1985]

- Unknown parameters: weights on the synapses
- **Minimizing** a cost function: $\rightarrow$*some metric between the predicted output and the given output*





Remark

Step function: non continuous function is replaced by a continuous non linear function

Theorem

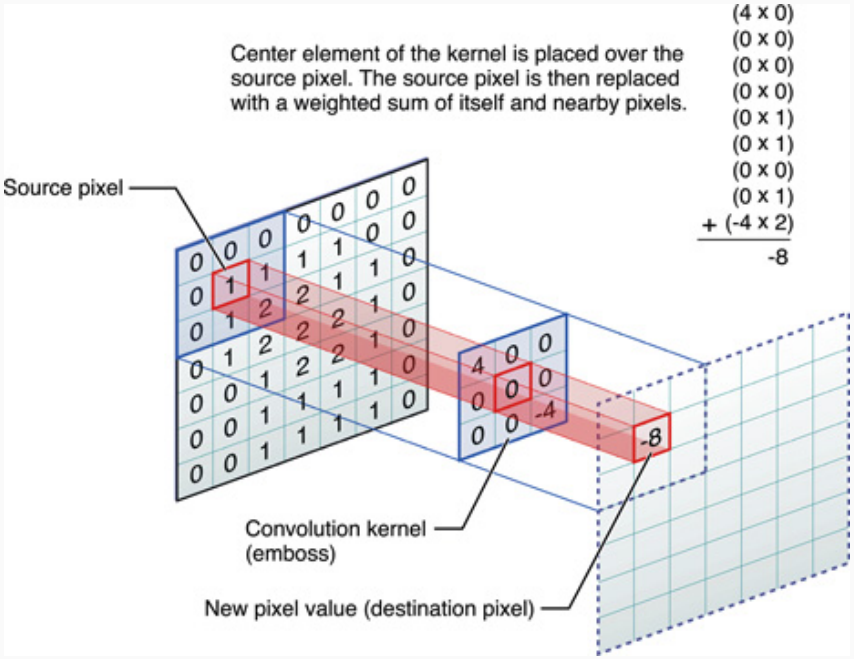
A neural network with one single hidden layer is a universal approximator: it can represent any continuous function on compact subsets of $\mathbb{R}^n$ [Cybenko, 1989]

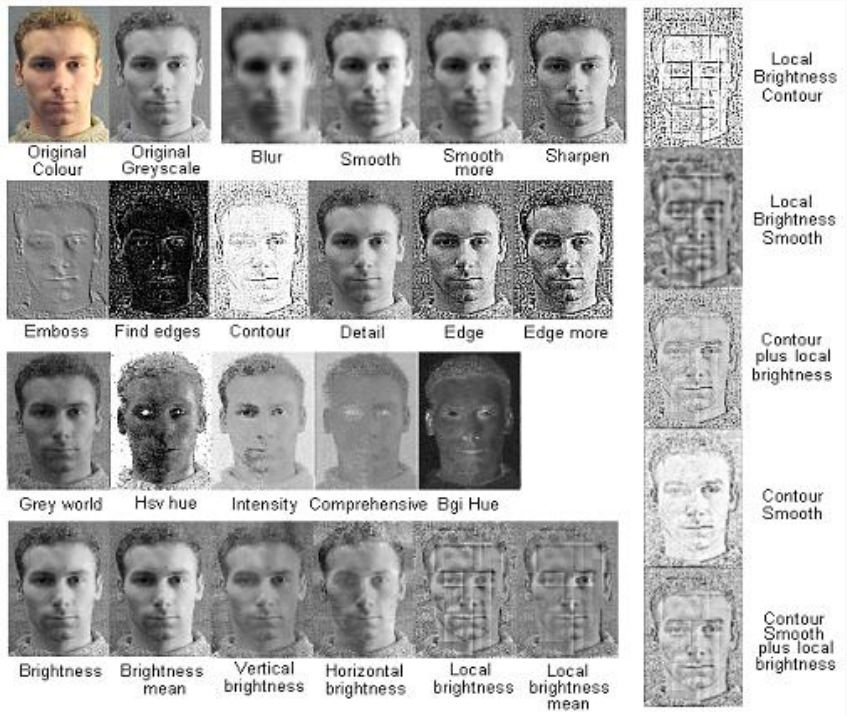
Theorem

Functions representable compactly with k layers may require exponentially size with $k-1$ layers
[Hastad, 1986, Bengio et al., 2007]

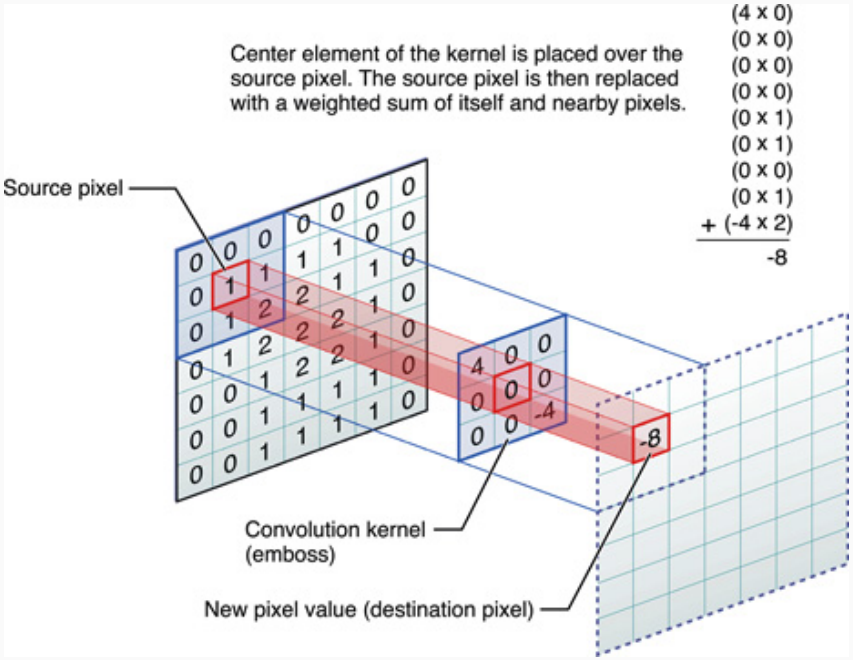
Building a CNN

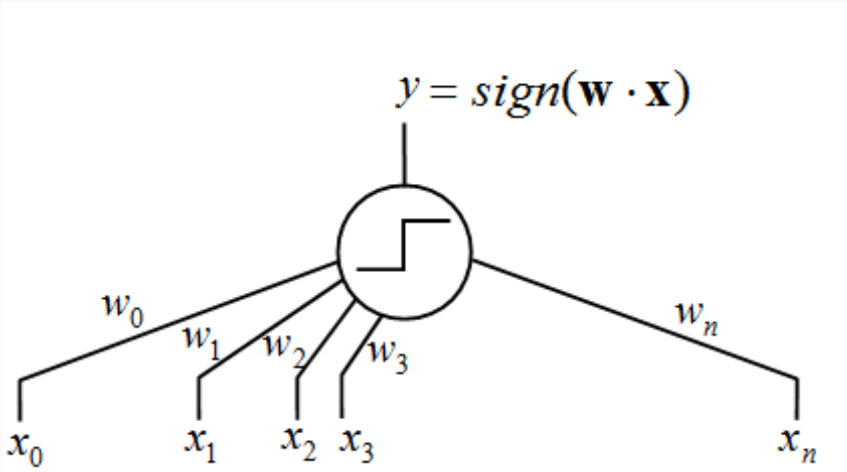
Image filtering





Filtering ⇔ Train perceptron

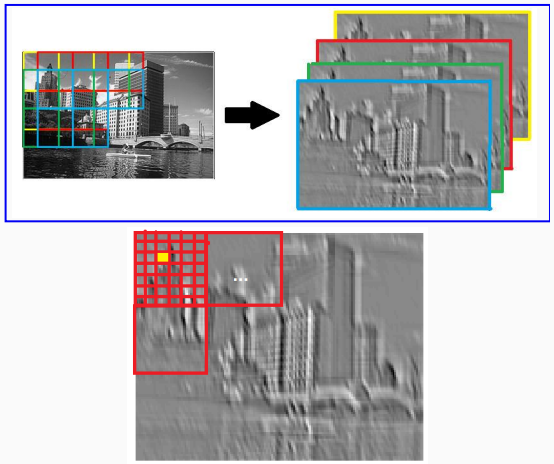


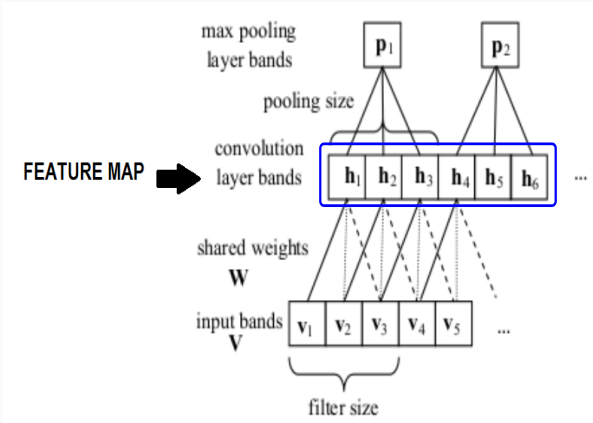


Convolutional neural network (CNN) units

- Yann Lecun, [LeCun et al., 1998]

1. subpart of the field of vision and translation invariant
2. convolution with filters
3. max pooling

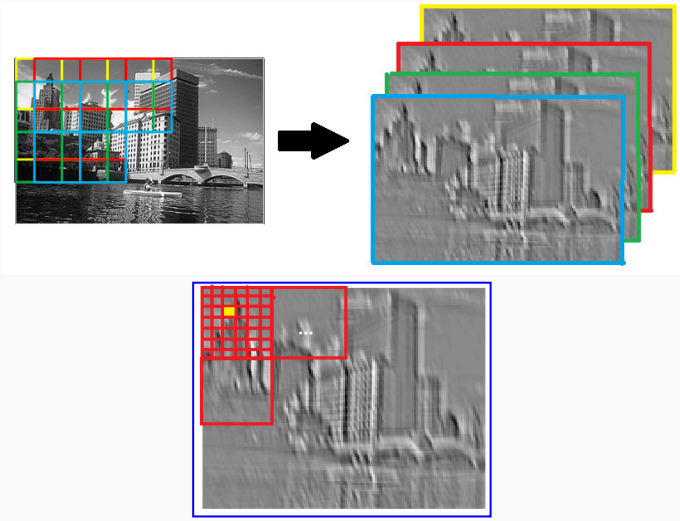


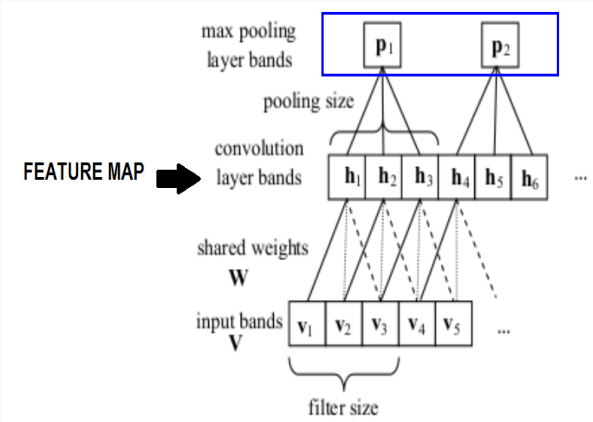


Convolutional neural network (CNN) units

- Yann Lecun, [LeCun et al., 1998]

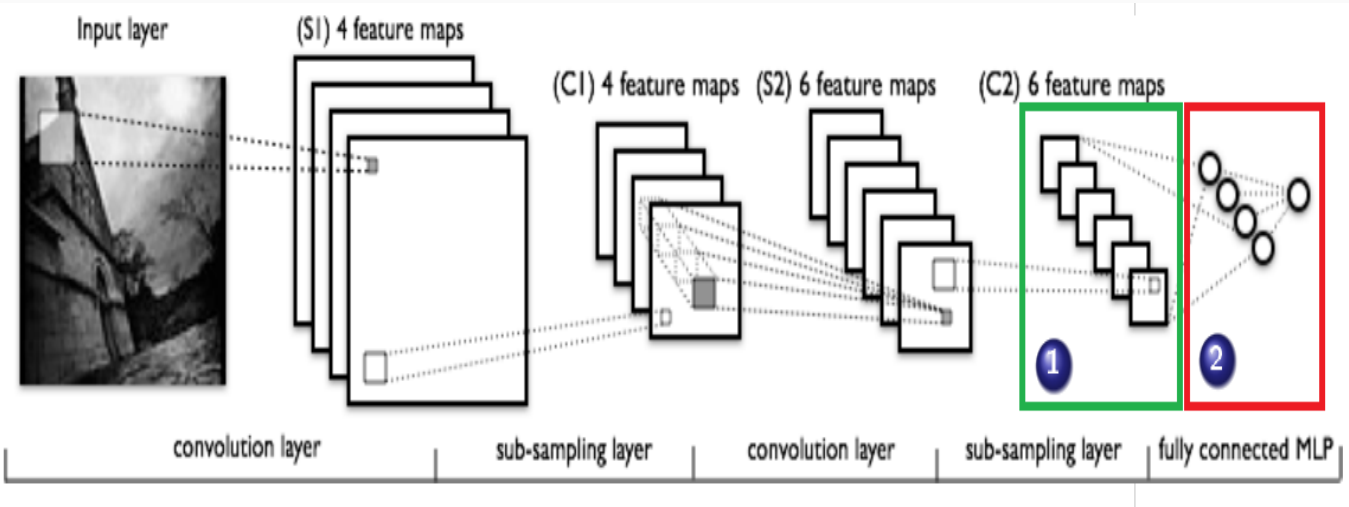
1. subpart of the field of vision and translation invariant
2. convolution with filters
3. max pooling





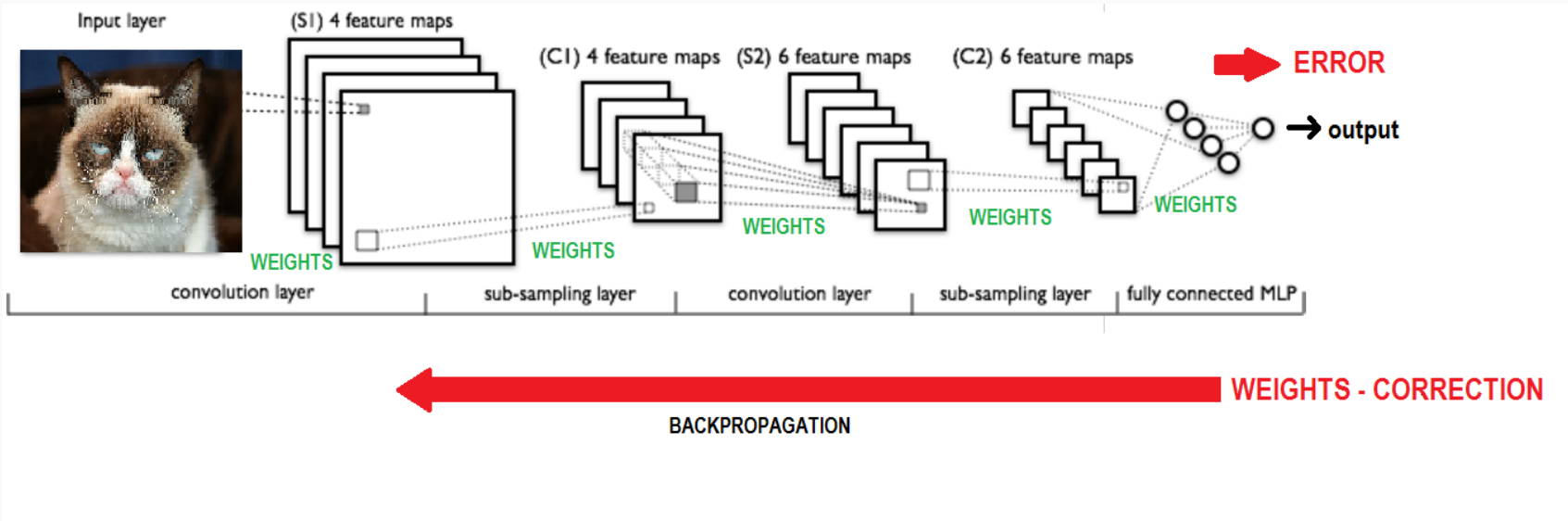
Full architecture of a CNN

- feature map = result of the convolution
- convolution with a filter extract characteristics (*edge detectors*)
- extract parallelised characteristics at each layer



1. final representation of our data
2. classifier (MLP)

supervised training



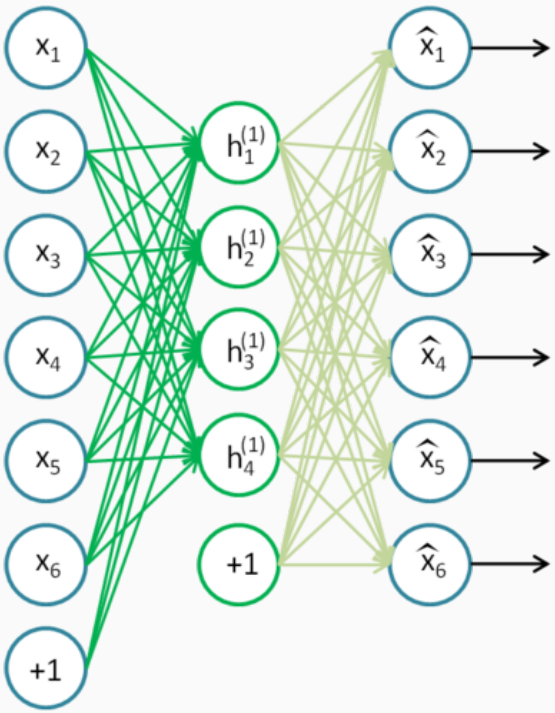
Building an AutoEncoder

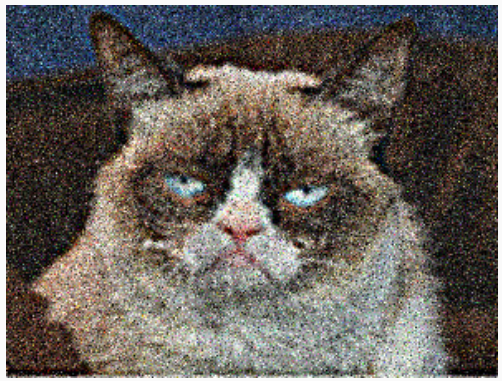
AutoEncoder: a piece of unsupervised

- Learning a compact representation of the data

(**no classification**)

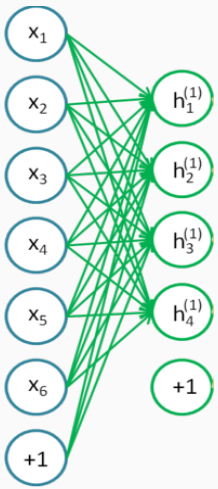
First we train an AutoEncoder layer 1.

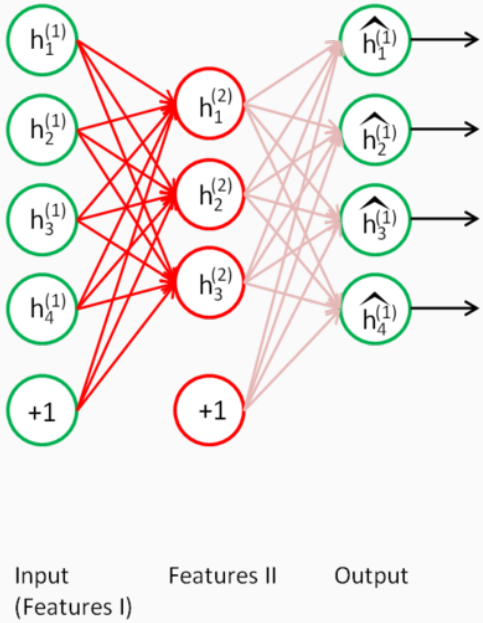




AutoEncoder: a piece of unsupervised

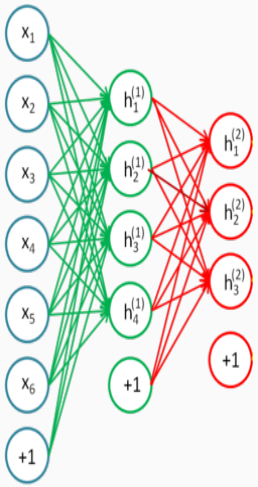
Second we train an AutoEncoder layer 2.

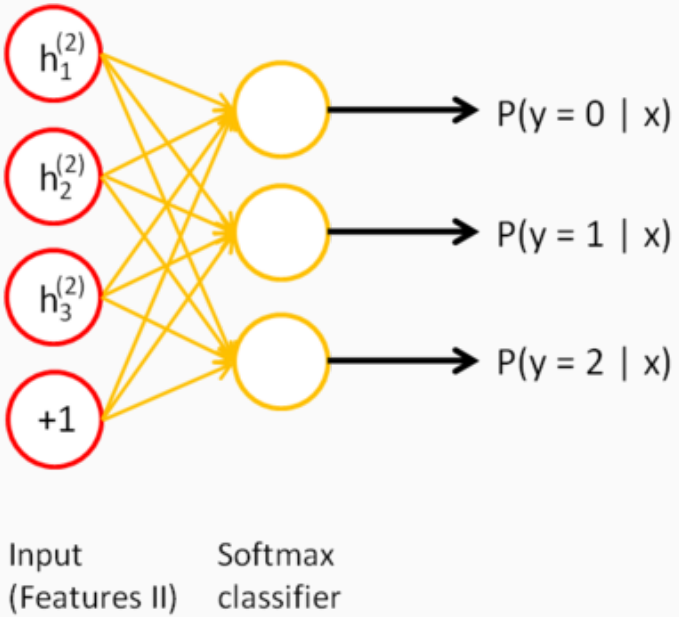




AutoEncoder: a piece of unsupervised

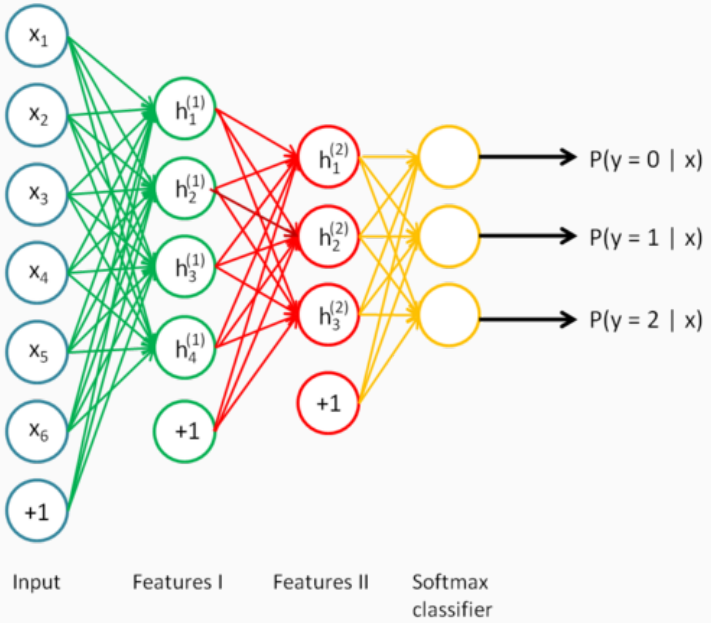
Then we train an output layer of non linearities based on softmax.





AutoEncoder: a piece of unsupervised

Finaly, we fine tune the whole network in a supervised way.

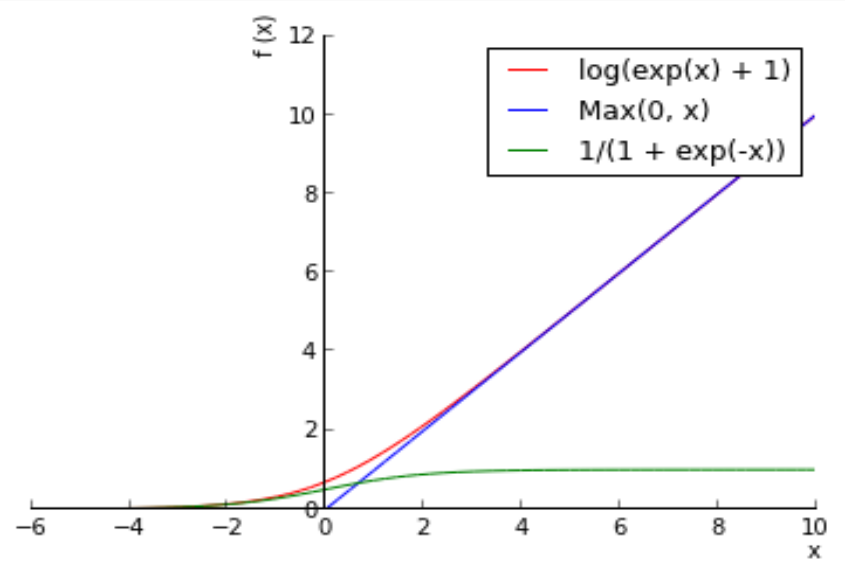


Tips and Tricks: implementation details to build and train your own deep networks

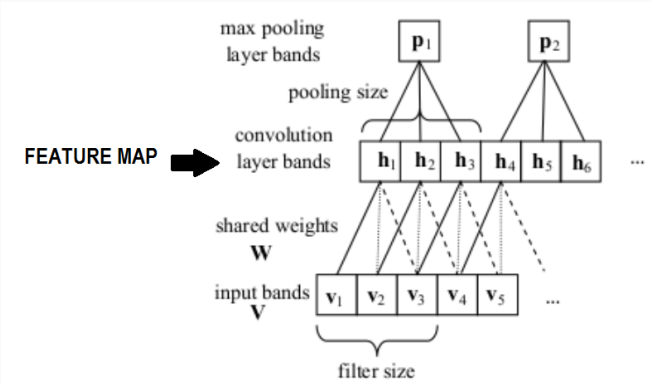
Which activation function in a neuron ?

Rectified Linear Unit (ReLU) [Glorot et al., 2011]:

- Vanishing gradient : corrections are disappearing along backward pass
- Hyperbolic function mimic better biological behaviour
- Rectifier more robust to vanishing gradient



Vanishing gradients (Part 1)



How to initialize the weights?

1. no symmetry
2. gaussian or uniform distribution [Daniely et al., 2016]
3. some rules to prevent vanishing gradients

Vanishing gradient (Part 2)

how to calibrate the variance

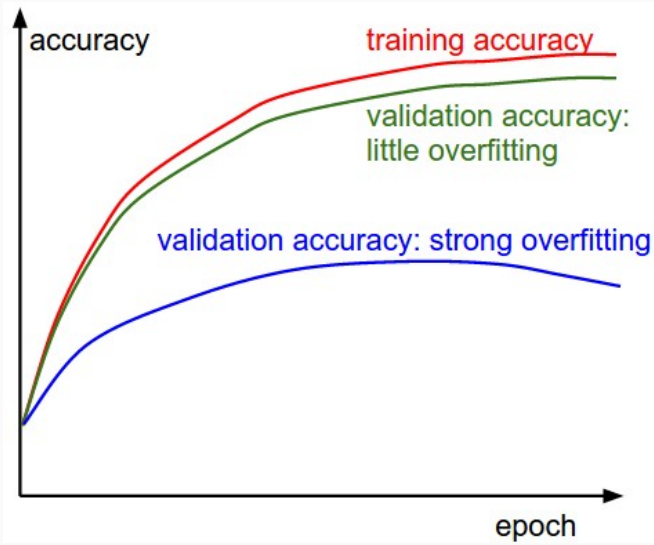
- keep the variance of the input layers balanced (in the forward and backprop pass)
- Glorot for hyperbolic activation [Glorot and Bengio, 2010]:

$$Var(W) = \frac{2}{n_{input} + n_{output}} \tag{1}$$

- He for Rectifier unit [He et al., 2015]: Surpassing Human-Level Performance on ImageNet Classification

$$Var(W) = \frac{2}{n_{i}nput} \tag{2}$$

The stage of learning



babysitting your deep network

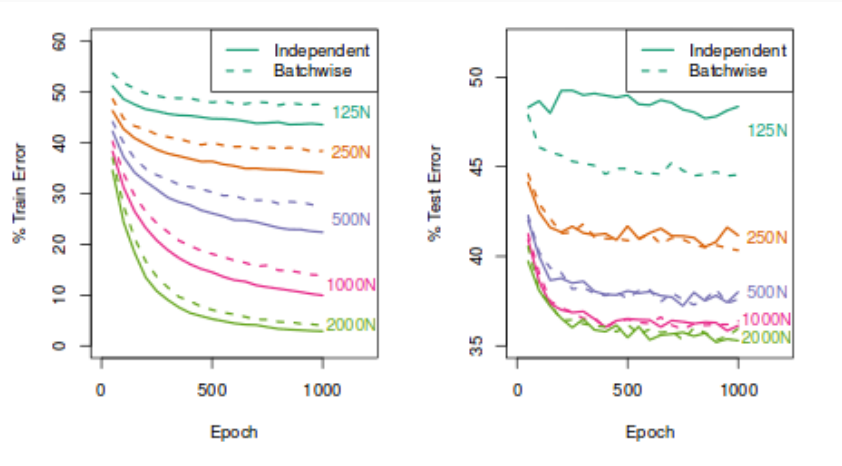
1. overfitting and underfitting
2. check accuracy before training [Saxe et al., 2011]
3. Y. Bengio : "*check if the model is powerful enough to overfit, if not then change model structure or make model larger*"

Regularization

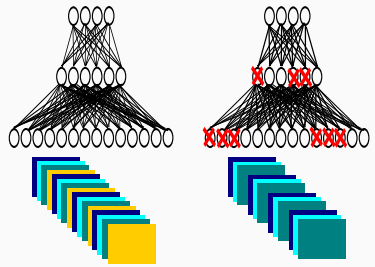
- L2 regularization : smoother weights
- L1 regularization : strict sparsity
- Elastic net regularization : L1 + L2 regularization
- weights clampling

Avoid overfitting : the dropout familly [Srivastava et al., 2014, Wan et al., 2013, Graham et al., 2015]

- prevent co-adaptation by switching off neurons
- dropout rate : 0.5



- DropConnect : remove edges
- Batchwise dropout : dropout + remove filters



Avoid overfitting : Batch Normalization [Ioffe and Szegedy, 2015]

internal covariate shift: a change in the distribution of activations because parameters updates might slow learning

force each layer to follow a normal distribution (in a differentiable manner)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

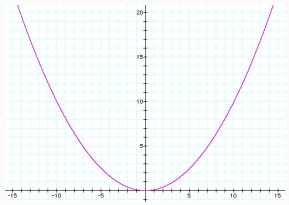
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

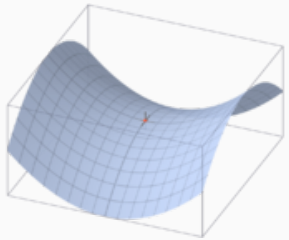
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- faster learning
- increase accuracy
- potentially use a higher learning rate
- prevent against bad initialization

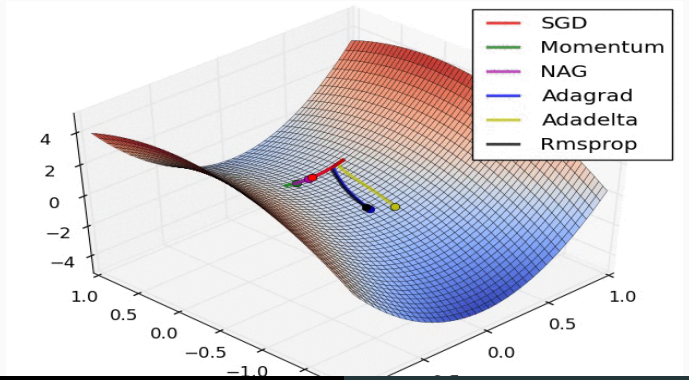
Optimization function





around local minima, more and more saddle points [Choromanska et al., 2015]

- optimizations to escaper from saddle points : Momentum, RMSProp, Adam ... [Hinton et al.,]



Our work in the lab

Deep Learning team into MIND

Acknowledgments of all people collaborating with us on Deep Learning

- **Deep permanents:** Melanie Ducoffe, Geoffrey Portelli, Frederic Precioso
- **Deep permanents collaborators:** Frederic Lavigne, Eric Debreuve, Michel Riveill
- **Deep non permanents:** Tom Bond, Melissa Sanabria Rosas, Fabi Eitel, Joana Iljazi

Deep Learning projects into MIND

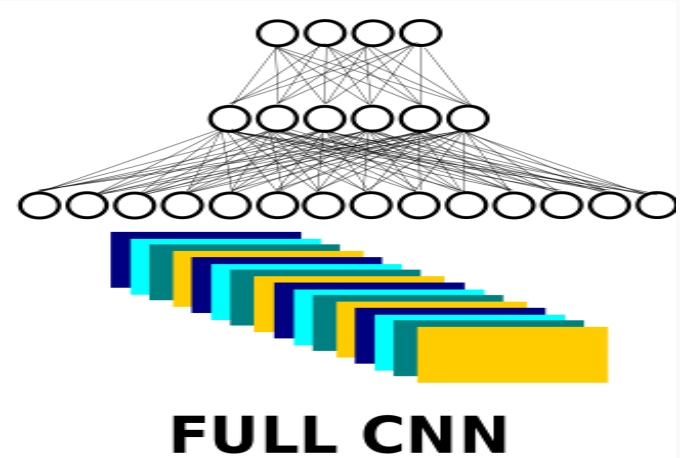
- Theory, fundaments, active learning, heterogeneous data
  (M. Ducoffe, F. Precioso)
- Hyperparameter optimization with bayesian search
  (M. Ducoffe, J. Iljazi, F. Precioso)
- Bio Deep
  (G. Portelli, M. Ducoffe, F. Lavigne, F. Precioso)
- Artistic Style for Neural Network to Sound
  (T. Bond, F. Eitel, M. Ducoffe, F. Precioso)
- Plankton Recognition
  (M. Sanabria Rosas, M. Ducoffe, E. Debreuve, F. Precioso)
- Autonomous RC car
  (Many students !!!, M. Ducoffe, F. Precioso)
- Laryngeal EMG Recognition
  (Many students !!!, M. Ducoffe, F. Precioso)

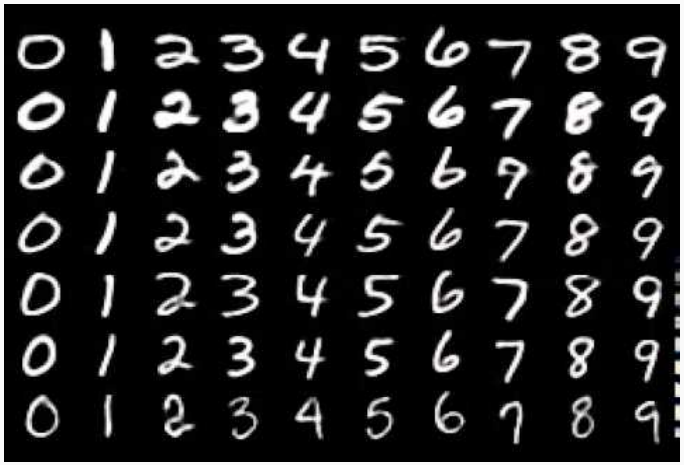
Effective and scalable batch active learning for Deep Learning

For supervised classification : ask the network which sample to label so to grow its annotated training set.

Goal : ask as few data as possible with equivalent accuracy than training with a bigger database

Field of application : medecine, low memory platforms, expensive annotations





Related work : Pool-based active learning

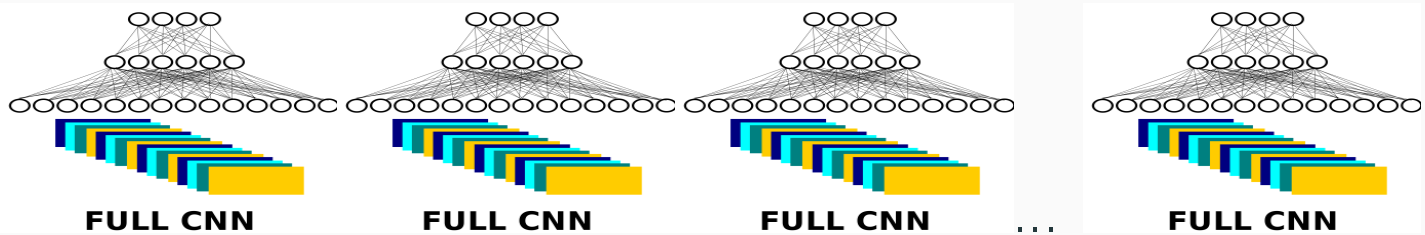
Method	Specificities
A) uncertainty sampling	selection on the least confident prediction
B) expected error reduction	select samples which minimizes the generalization error
C) Query By Committee	pool on a committee whose members are sampled on the current hypothesis space

A) Ok ... ✓ [Zhou et al., 2013]

prone to noise, and focus on a subset of classes. No great success but computationally fast

Related work : Pool-based active learning

Method	Specificities
A) uncertainty sampling	selection on the least confident prediction
B) expected error reduction	select samples which minimizes the generalization error
C) Query By Committee	pool on a committee

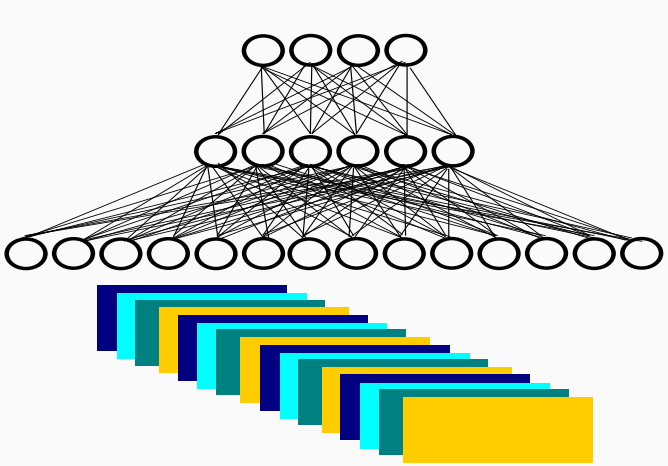


C) Interesting with an efficient and fast sampling scheme ✓ [Seung et al., 1992]

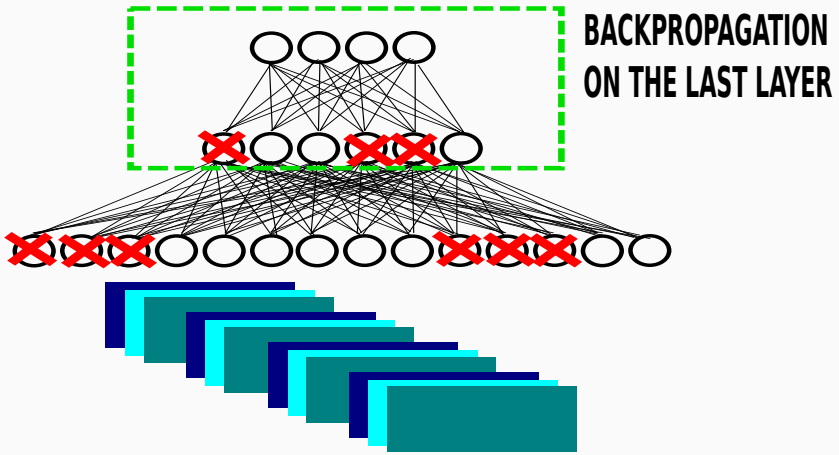
- hypothesis space : all the parameters which give a certain accuracy
- query $\rightarrow$ most disagreement so to reduce the consistent networks

Active decision module: committee of partial CNNs (pCNN)

- Sampling on the hypothesis space by batchwise dropout [Graham et al., 2015]
- Increase the accuracy of partial CNNs by Backpropagation on the last layer only



FULL CNN

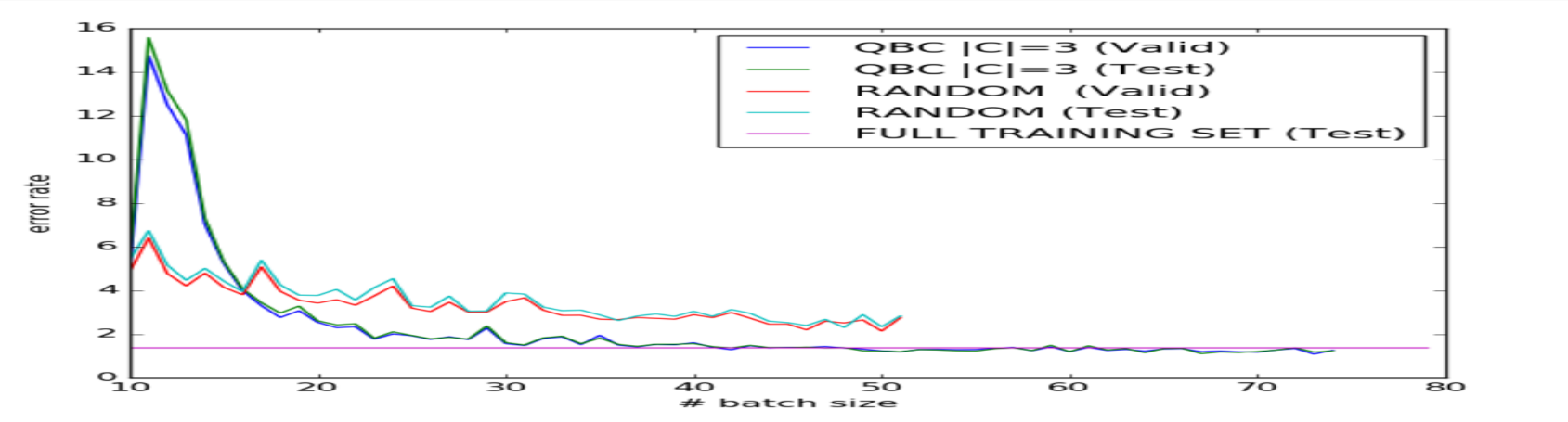


PARTIAL CNN pCNN

Experiment and Applications

METHOD	MEAN error rate	MIN error rate
QBDC	1.10	0.99
RANDOM	2.13	1.78

Average and best error rate on MNIST test set (30%)



Application to intuitive linguistic analysis in NLP

Linguistic analysis of the sample asked by the network (JADT 2016)

Conclusion

Why giving a try to Deep Learning?

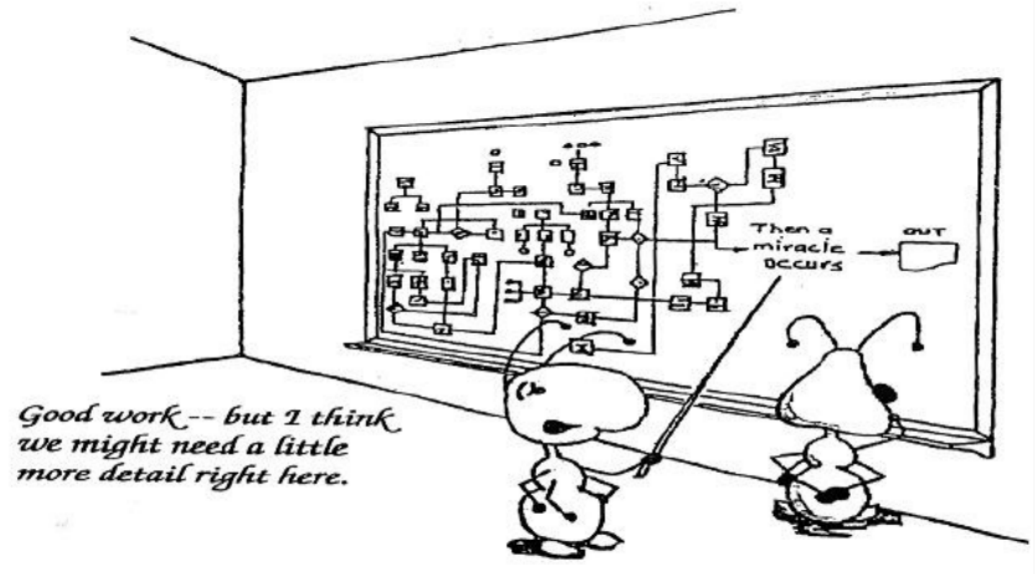
- Because it works to reach outstanding improvements!
- Because the challenges are exciting and it is just the beginning
- Because the room for improvements is wide
- A better understanding of what is going on

Any question ?

To contact us

Melanie Ducoffe : ducoffe@i3s.unice.fr

Frederic Precioso : precioso@i3s.unice.fr




Bellman, R. (1956).
Dynamic programming and lagrange multipliers.
Proceedings of the National Academy of Sciences of the United States of America, 42(10):767.


Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H., et al. (2007).
Greedy layer-wise training of deep networks.
Advances in neural information processing systems, 19:153.


Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. (2015).
The loss surfaces of multilayer networks.
In *AISTATS*.

 Cybenko, G. (1989).
Approximation by superpositions of a sigmoidal function.
Mathematics of control, signals and systems, 2(4):303–314.

 Daniely, A., Frostig, R., and Singer, Y. (2016).
Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity.
arXiv preprint arXiv:1602.05897.

 Glorot, X. and Bengio, Y. (2010).
Understanding the difficulty of training deep feedforward neural networks.
In *Aistats*, volume 9, pages 249–256.

 Glorot, X., Bordes, A., and Bengio, Y. (2011).

Deep sparse rectifier neural networks.

In *Aistats*, volume 15, page 275.

Graham, B., Reizenstein, J., and Robinson, L. (2015).

Efficient batchwise dropout training using submatrices.

arXiv preprint arXiv:1502.02478.

Hastad, J. (1986).

Almost optimal lower bounds for small depth circuits.

In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. ACM.

He, K., Zhang, X., Ren, S., and Sun, J. (2015).

Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034.


Hinton, G., Srivastava, N., and Swersky, K.
Lecture 6a overview of mini–batch gradient descent.
Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online.


Ioffe, S. and Szegedy, C. (2015).
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
arXiv preprint arXiv:1502.03167.


Lecun, Y. (1985).

***Une procedure d'apprentissage pour reseau a seuil asymmetrique (A learning scheme for asymmetric threshold networks)*, pages 599–604.**

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998).
Gradient-based learning applied to document recognition.

Proceedings of the IEEE, 86(11):2278–2324.

Mallat, S. (2014).
Des mathematiques pour l'analyse de donnees massives.
http://www.academie-sciences.fr/video/v180214.htm.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985).
Learning internal representations by error propagation.

Technical report, DTIC Document.


Saxe, A., Koh, P. W., Chen, Z., Bhand, M., Suresh, B., and Ng, A. Y. (2011).
On random weights and unsupervised feature learning.
In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1089–1096.


Seung, H. S., Opper, M., and Sompolinsky, H. (1992).
Query by committee.
In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM.


Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).

Dropout: a simple way to prevent neural networks from overfitting.

Journal of Machine Learning Research, 15(1):1929–1958.

 Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., and Fergus, R. (2013).

Regularization of neural networks using dropconnect.

In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066.

 Zhou, S., Chen, Q., and Wang, X. (2013).

Active deep learning method for semi-supervised sentiment classification.

Neurocomputing, 120:536–546.