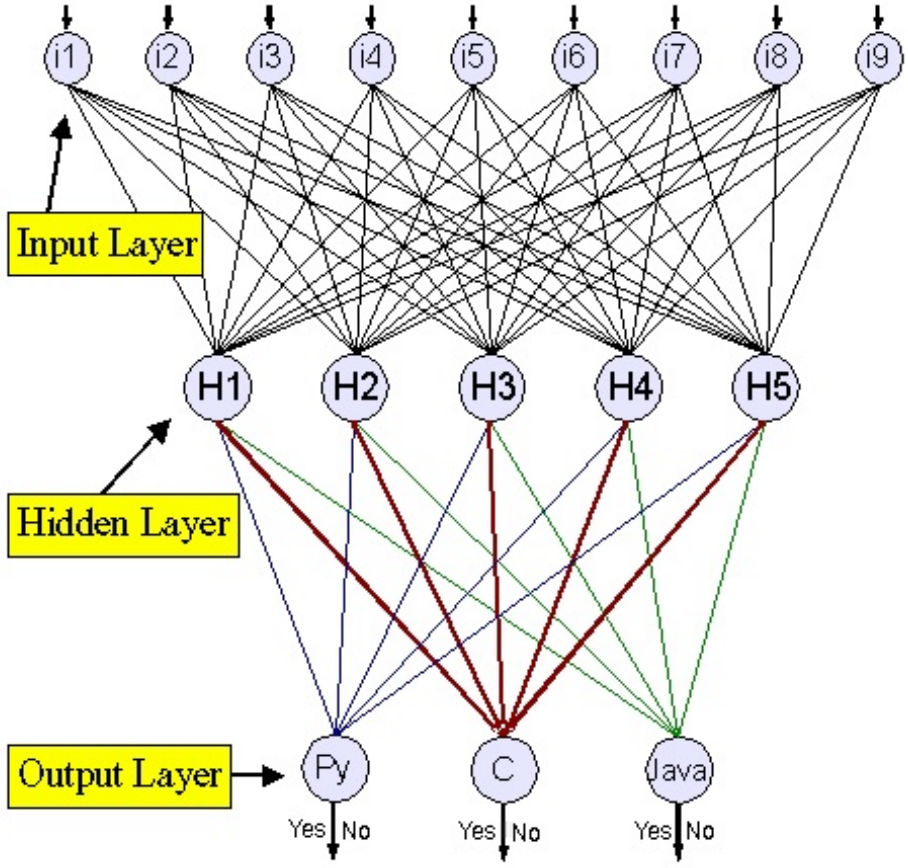
Pattern Recognition :
The Flagship Problem in Artificial Intelligence



Machines with the ability to learn automatically and be smarter than us?

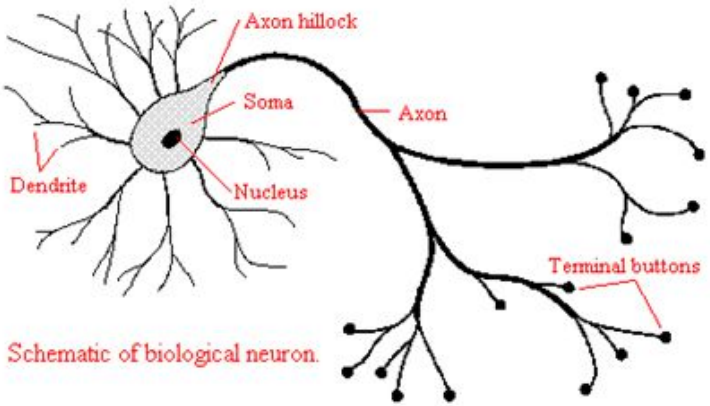
"AlphaGo knocks world champion Lee Se Dol out!"

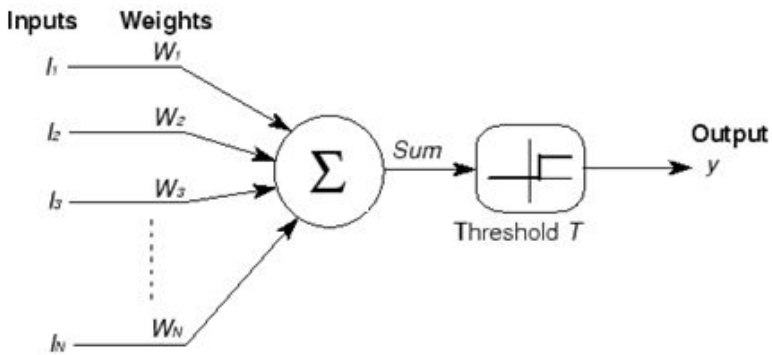
Machines with the ability to learn automatically and be smarter than us?

"Watson wins on 'Jeopardy'!"

Machines that listen or see better than us?

Le neurone

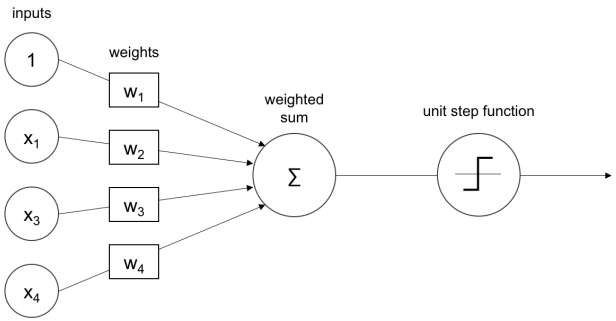




Neurone biologique

Neurone artificiel

The first AI model : artificial neuron



- A mathematical function introduced by McCulloch & Pitts '43
- Mimics the synaptic transmission of the information to a biological neuron
- Given the weight vector w, the function is implemented in a sumplistic fashion :
  1. Compute the inner product $\langle x, w \rangle = {}^t w \cdot x$ (multiplications and summations only)
  2. Activate the neuron if the inner product plus a parameter θ is positive, do not activate it otherwise

$$g(x) = sgn(\langle x, w \rangle + \theta)$$

The first AI model : artificial neuron

- We can collect and store labeled examples $(X_1, Y_1), \ldots, (X_n, Y_n)$
- We need an algorithm to choose θ and the synaptic weights w, so as to reproduce best the examples !
- The first learning algorithm : the Monolayer Perceptron

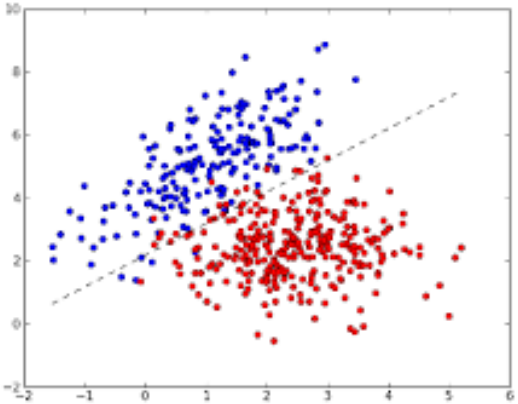
Learning an artificial neuron model
Frank Rosenblatt (1957)

- A probabilistic/statistical view of AI problems
- Motivation : computer vision for aeronautics
- Exploit 'recent' advances in optimization (stochastic approximation for gradient descent, Robbins & Monro '51)
- Exploit the capacities of 'high-speed' calculators in the 50's
- 'On-line' algorithm

The Algorithm - Geometric ideas

- Goal : learn how to split the input space $\mathbb{R}^d$ into two halves, separated by an **affine hyperplane** of eq. $\theta + {}^t w \cdot x = 0$

$$g(x) = sgn({}^t w \cdot X + \theta)$$



Assign positive label to any input x above the hyperplane and negative label when x is below it

- Ideally minimize over (w, θ) in $\mathbb{R} \times \mathbb{R}^d$, the **empirical error**

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{-Y_i({}^t w \cdot X_i + \theta) > 0\}$$

Perceptron Algorithm

- Initialize $w \in \mathbb{R}^{d+1}$
- Until Convergence, repeat:
    - For $i = 1,...,n$
        - If $Y_i \langle w, X_i \rangle \leq 0$ then:

$$w \leftarrow w + Y_i X_i$$

    End For
- Return w

- Equivalent to minimize the Hinge loss with gradient descent.

$$\min_{w \in \mathbb{R}^{d+1}} \frac{1}{n} \sum_{i=1}^{n} \max(0, \ -Y_i h(X_i))$$

with $h(X_i) = \langle w, X_i \rangle$.

The Algorithm

- Observe that it is the statistical counterpart of the probability of error

$$L(g) = \mathbb{P}\{Y \neq g(X)\}$$

based on the training data, when $g(x) = \theta + {}^t w \cdot x$

- Main barrier to the optimization of the empirical error : $u \mapsto \mathbb{I}\{u > 0\}$ is not differentiable !

- Surrogate problem : minimize

$$-\sum_i Y_i({}^t w \cdot X_i + \theta)$$

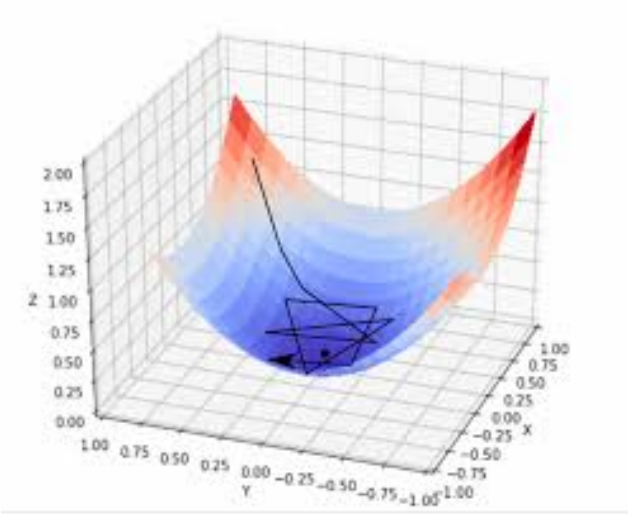
by stochastic gradient descent : $(w, \theta) \mapsto -Y({}^t w \cdot X + \theta)$ is differentiable with gradient (YX, Y)

The Algorithm

- Start with an initial guess for (w, θ)
  1. Choose **at random** a point (X_i, Y_i) misclassified by the current rule (if there is any)
  2. Change the rule parameters (w, θ) by a step into the opposite direction of the gradient computed at (X_i, Y_i) with rate/stepsize ρ

$$\begin{pmatrix} w \\ \theta \end{pmatrix} \leftarrow \begin{pmatrix} w \\ \theta \end{pmatrix} + \rho \begin{pmatrix} Y_i X_i \\ Y_i \end{pmatrix}$$

- Repeat until there is no misclassified observations anymore...



Once the rule is learned, how well does it work ?

- If (w, θ) has been learned by means of the training dataset $D_n = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$, DO NOT rely on the **training error** to evaluate its future performance !

$$\frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\{-Y_i(\theta + {}^t w \cdot X_i) > 0\}$$

- The parameters have been precisely chosen to mimic the input-output pairs (X_i, Y_i), the training error is too optimistic to estimate the **true/theoretical** risk

$$\mathbb{P}\{-Y(\theta + {}^t w \cdot X) > 0\}$$

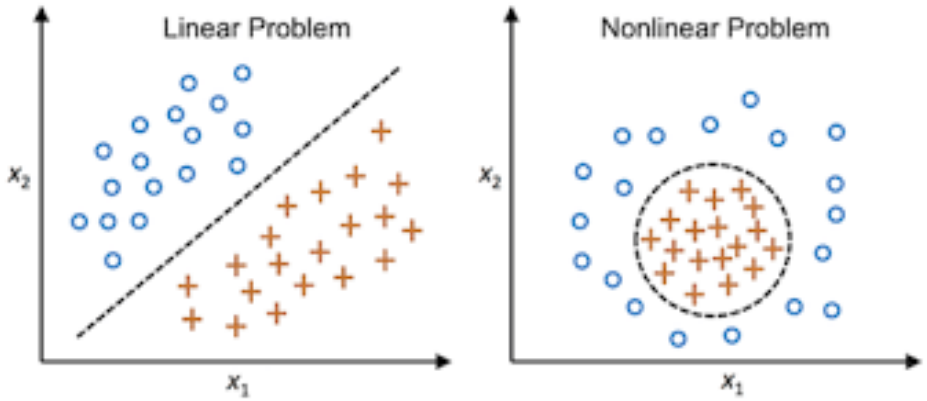
- If possible, use another dataset $\{(X'_1, Y'_1), \ldots, (X'_{n'}, Y'_{n'})\}$, independent from D_n (**test data**) to compute the **test error**

$$\frac{1}{n'} \sum_{i=1}^{n'} \mathbb{I}\{-Y'_i(\theta + {}^t w \cdot X'_i) > 0\},$$

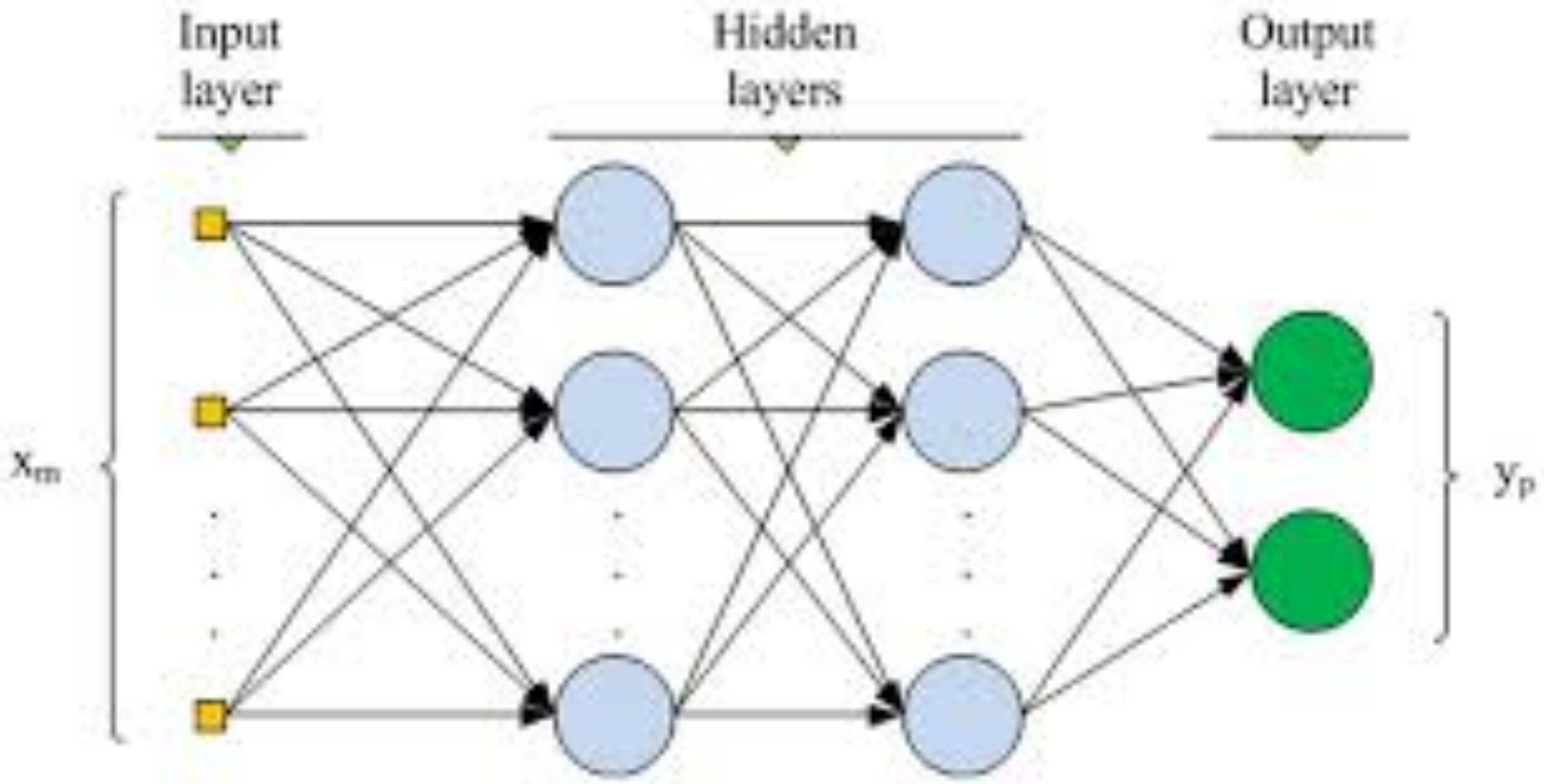
which is a fair (unbiased) estimate of $\mathbb{P}\{-Y(\theta + {}^t w \cdot X) > 0\}$

Only the beginning of the story...

- Easy (and possibly on-line) implementation, that requires no sophisticated library
- If the data are linearly separable, it converges in a finite number of steps
- If not, the hyperplane will oscillate forever...



Network of Artificial Neurons (multilayer perceptron, MLP)



From the Artificial Neuron model to Neural Networks 1/2

- Artificial Neuronl : Mc Cullogh et Pitts, 1943
- Learning the Artificial Neuron model : the Perceptron by Rosenblatt, 1957
- Minsky and Papert : limitation of the Perceptron, 1959
- Learning a multi-layer perceptron by gradient backpropagation, Y. Le Cun, 1985, Hinton and Sejnowski, 1986.
- Multi-Layer Perceptron = a universal approximant, Hornik et al. 1991
- Convolutional networks, 1995, Y. Le Cun and Y. Bengio
- Between 1995 et 2008, the domain is flat (non convexity, computationally demanding, no theory)

From the Artificial Neuron model to Neural Networks 1/2

- Democratization of GPU's (graphical processing units) 2005
- Large image databases : Imagenet, Fei-Fei et al. 2008 (now much more than 10^6 images)
- Deeper and deeper neural networks, learned by means of massive databases
- Initialization with unuspervised learning (autoencoder)
- Word2vec (Mikolov et al. 2013)
- Dropout (Srivastava et al. 2014)

Artificial Neuron

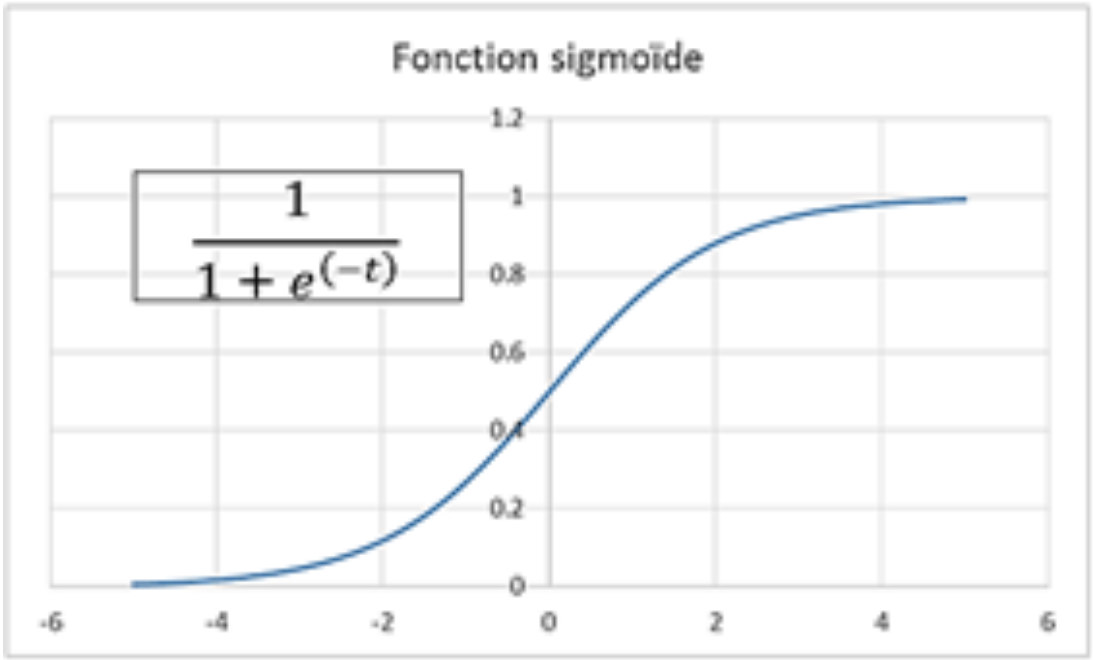
- activation function (e.g. sign)
- weight vector and bias (intercept)

$$f(x) = g(w^T x + b) \tag{1}$$

Choose g differentiable preferably (cf gradient optimization techniques)

Activation function for the Artificial Neuron

For instance :



One also uses hyperbolic tangent *tanh* (values in the range $(-1,\ 1)$).

Add a processing intermediary layer

Now, compute :

$$f(x) = g(\Phi(x)^T w + b)$$

Feature map or latent representation.
Flexibility of neural networks : the feature map Φ is learned from the training data.

Universal Approximation

In 1991, Hornik et al. prove that MLP's with one hidden layer and $p+1$ input is dense in the space of real valued continuous functions on $\mathbb{R}^p$. A MLP with one hidden layer is a **universal approximant**.

Universal Approximation

In 1991, Hornik et al. prove that MLP's with one hidden layer and $p+1$ input is dense in the space of real valued continuous functions on $\mathbb{R}^p$. A MLP with one hidden layer is a **universal approximant**.
Some other flexible/rich classes of decision functions (more next week !) :

- Linear regressor : NO
- SVM with a universal kernel, e.g. Gaussian kernel : YES
- Random Forests : YES
- Boosting stumps : YES

Model Selection

The MLP has several hyperparameters :

- Nb of hidden layers
- Sizes of the hidden layers
- parameter λ
- nb_{cycle}, ε
- $\eta_t = \frac{\gamma}{1+t}$

In general, they are selected by means of CROSS VALIDATION.

Advantages and drawbacks of Neural Networks "feedforward"

- **Pros**
  - Flexibility regarding the output : arbitrary number of classes, etc..
  - Fitting methos known since the mid 80's
  - Stochastic gradient descent is appropriate to deal with BIG DATA
  - GPU architectures can be used
  - PLUG and PLAY : the same paradigm can be used successively
- **Cons**
  - Non convex loss : no global minimum
  - Implementing the gradient descent requires many adjustments in practice
  - No theoretical validity framework
  - *Ad hoc* implementations - The 'art' of neural nets

Learning deep neural networks

From 3 layers, one uses the term "deep learning", this type of network is useful to analyze complex data such as textual data or images.

The why of the use of several hidden layers ?

The fact that a NN with a single hidden layer is a universal approximant does not mean that it provides the best representation/approximation.

Learning deep neural networks

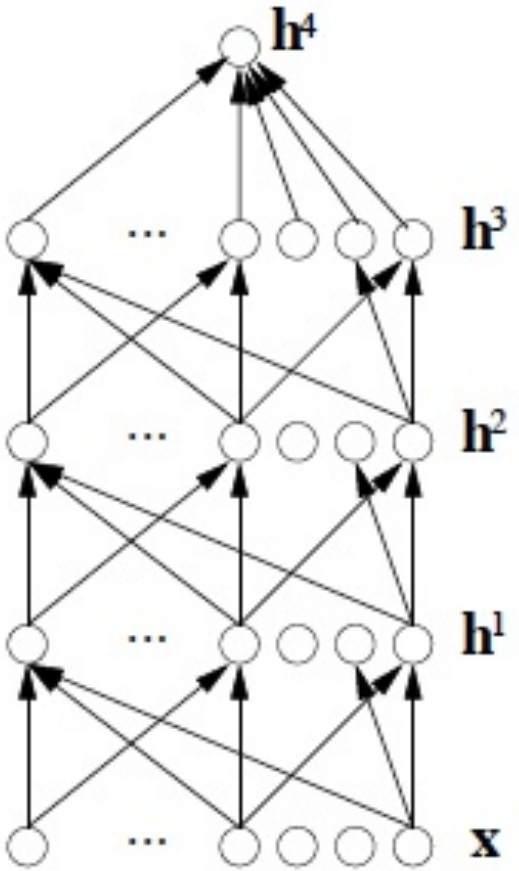
Despite the danger of overfitting,

two good reasons for considering deep neural nets

- advances in memory and computation (GPU)
- availability of massive databases (Imagenet, Fei-Fei, 2008)

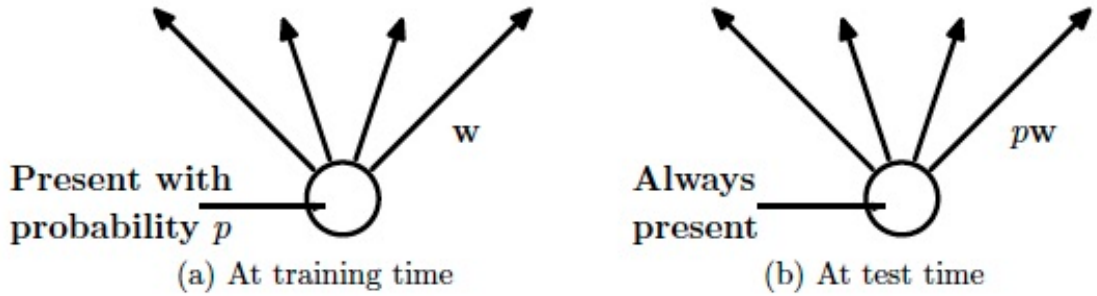
Gradient backpropagation does not work well for deep nets (Bengio et al. 2007 ; Erhan et al. 2009).
In absence of a good initialization, it often outputs bad local minima.

Learning deep neural networks



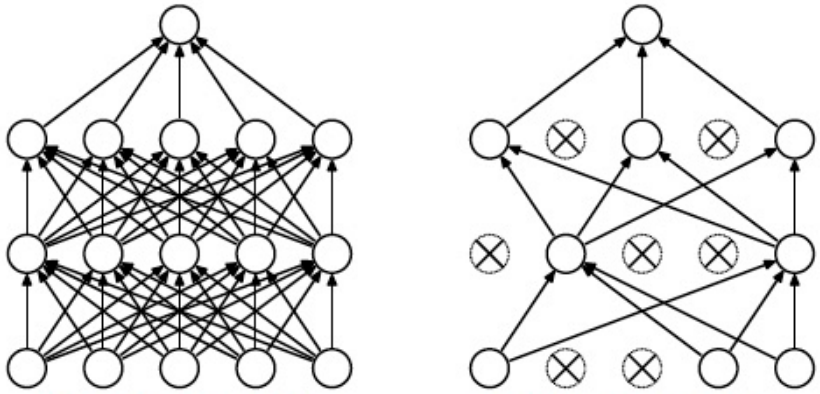
Avoid overfitting deep nets by *dropout 1/3*

For very deep nets (>>2 layers) :



- During the learning stage, at each gradient modification : each unit (neuron) is considered with probability p, meaning that certain neurons are not present and are consequently not corrected systematically.
- During the prediction (test stage), each unit is present with a factor p applied to its weights.

Avoid overfitting deep nets by *dropout 2/3*



(a) Standard Neural Net (b) After applying dropout.

Interpretation :

When m neurons are involved, it is like one learns 2^m sparse neural nets and during the test, they are aggregated to form the neural network used for prediction.

Neurons cannot adjust w.r.t. the others

Avoid overfitting deep nets by *dropout 3/3*

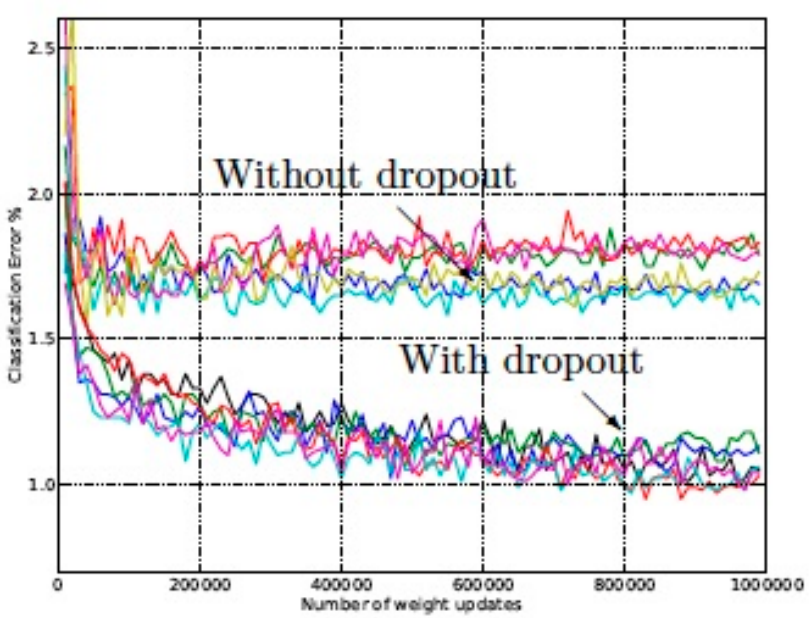


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.