Supervised Classification

Vincent Itier (IMT Nord Europe)
John Klein (Univ. Lille)

UV DATA 2021

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.
- On sait que des décisions optimales se prennent si on a accès à $p_{Y|X}$.

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.
- On sait que des décisions optimales se prennent si on a accès à $p_{Y|X}$.
- We saw that Err_{train} is linked to the negative log likelihood of probabilistic models:
  - Generative (ex: Naive Bayes),
  - Discriminative (ex: logistic regression).

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.
- On sait que des décisions optimales se prennent si on a accès à $p_{Y|X}$.
- We saw that Err_{train} is linked to the negative log likelihood of probabilistic models:
  - Generative (ex: Naive Bayes),
  - Discriminative (ex: logistic regression).
- Generative approach requires lots of data and intuition about the distribution $p_{X|Y}$

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.
- On sait que des décisions optimales se prennent si on a accès à $p_{Y|X}$.
- We saw that Err_{train} is linked to the negative log likelihood of probabilistic models:
  - Generative (ex: Naive Bayes),
  - Discriminative (ex: logistic regression).
- Generative approach requires lots of data and intuition about the distribution $p_{X|Y}$
- Logistic regression is a linear model:
  $\rightarrow$ the separating border is a straight line !

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.
- On sait que des décisions optimales se prennent si on a accès à $p_{Y|X}$.
- We saw that Err_{train} is linked to the negative log likelihood of probabilistic models:
  - Generative (ex: Naive Bayes),
  - Discriminative (ex: logistic regression).
- Generative approach requires lots of data and intuition about the distribution $p_{X|Y}$
- Logistic regression is a linear model:
  $\rightarrow$ the separating border is a straight line !
- Structured data.

Supervised learning problem?

- We know that if the data size n, is large then Err_{train} will descrease.
- On sait que des décisions optimales se prennent si on a accès à $p_{Y|X}$.
- We saw that Err_{train} is linked to the negative log likelihood of probabilistic models:
  - Generative (ex: Naive Bayes),
  - Discriminative (ex: logistic regression).
- Generative approach requires lots of data and intuition about the distribution $p_{X|Y}$
- Logistic regression is a linear model:
  $\rightarrow$ the separating border is a straight line !
- Structured data.

Neural networks converge to other kinds of separating boundaries, based on logistic regression.

Outline

Single neuron as a linear classifier:

- A neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space.
- Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

Single neuron as a linear classifier:

- A neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space.
- Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

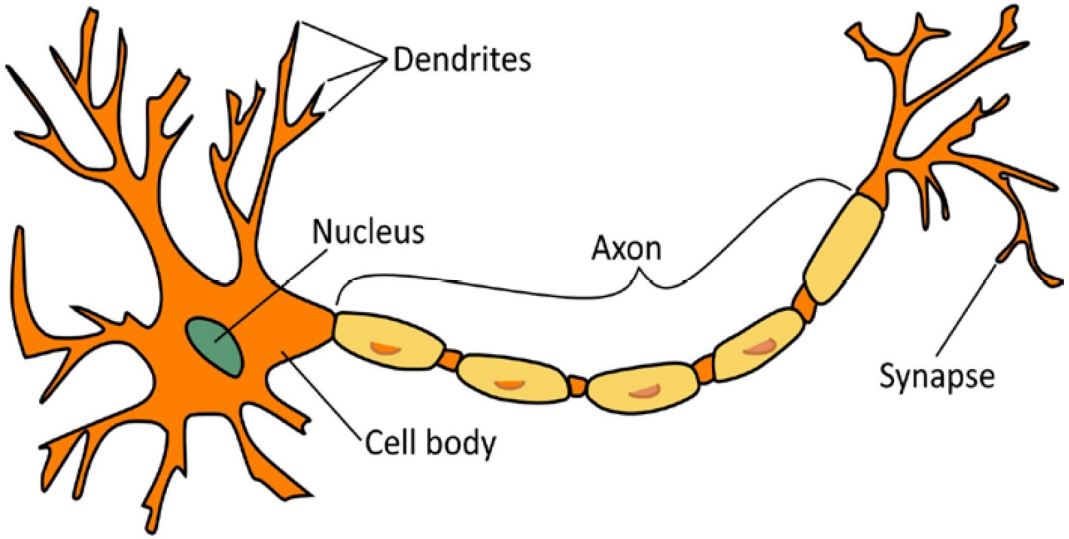
- Binary Softmax classifier:
  We can interpret $\sigma(\sum_i w_i x_i + b)$ as the probability of one of the classes $P(y_i = 1|x_i; x)$. The other class would be $P(y_i = 0|x_i; w) = 1 - P(y_i = 1|x_i; w)$.

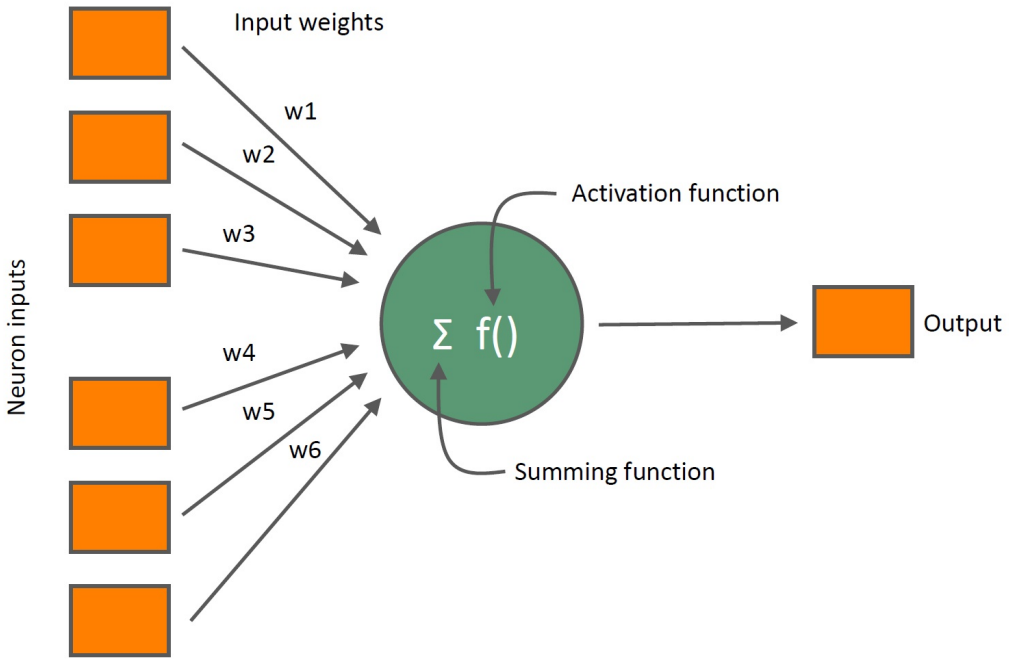
Single neuron as a linear classifier:

- A neuron has the capacity to "like" (activation near one) or "dislike" (activation near zero) certain linear regions of its input space.
- Hence, with an appropriate loss function on the neuron's output, we can turn a single neuron into a linear classifier:

- Binary Softmax classifier:
  We can interpret $\sigma(\sum_i w_i x_i + b)$ as the probability of one of the classes $P(y_i = 1|x_i; x)$. The other class would be $P(y_i = 0|x_i; w) = 1 - P(y_i = 1|x_i; w)$.
- Binary SVM classifier:
  Attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

Biological mimic:

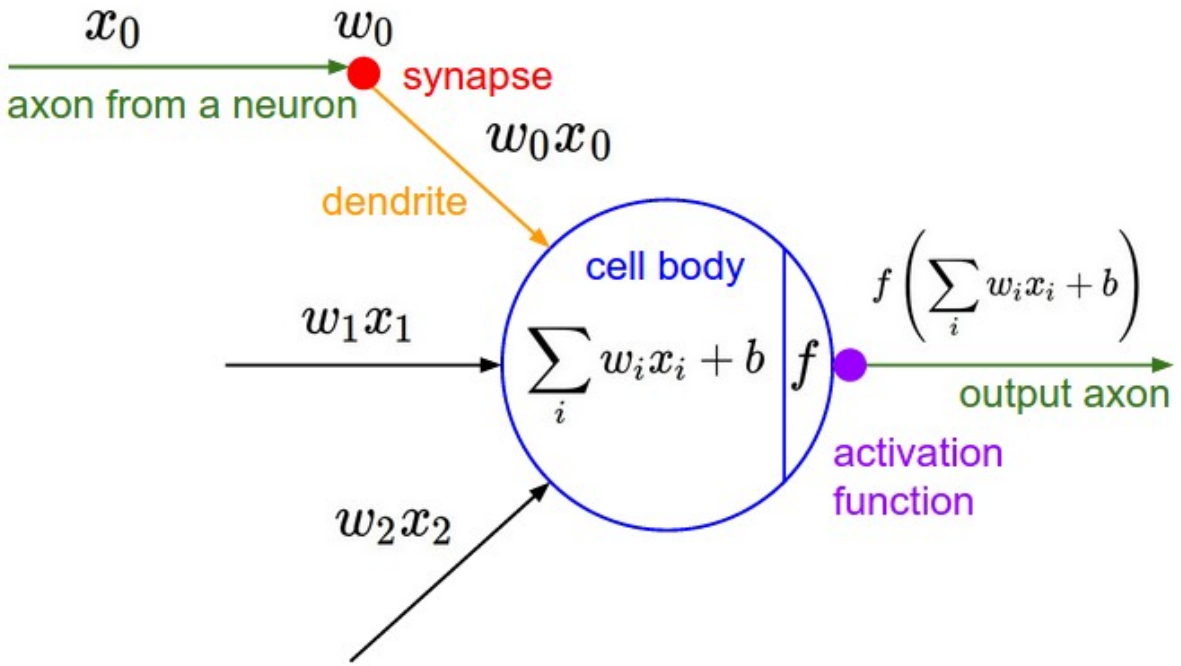




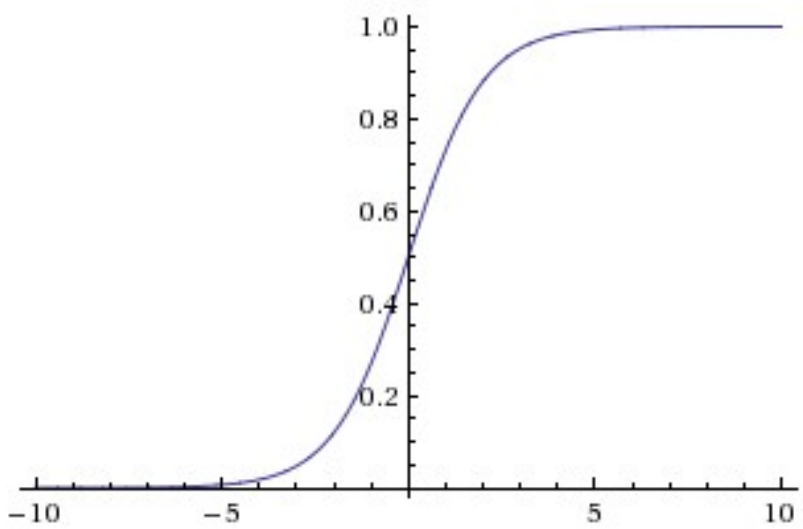
Just an inspiration!

Just an inspiration!

Biological mimic:



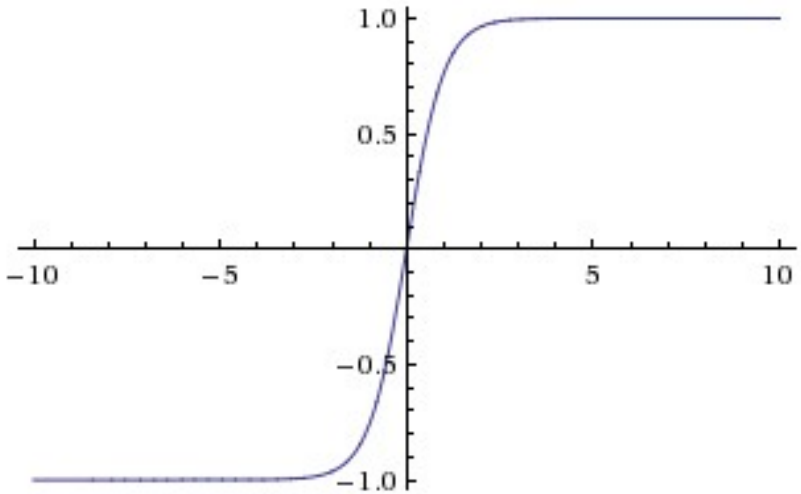
Commonly used activation functions:



The sigmoid non-linearity.

- $\mathbb{R} \mapsto [0, 1]$.
- $\sigma(x) = \frac{1}{1+e^{-x}}$.
- Historically since it has a nice interpretation
- Two major drawbacks:
  - Sigmoids saturate and kill gradients.
  - Sigmoid outputs are not zero-centered.

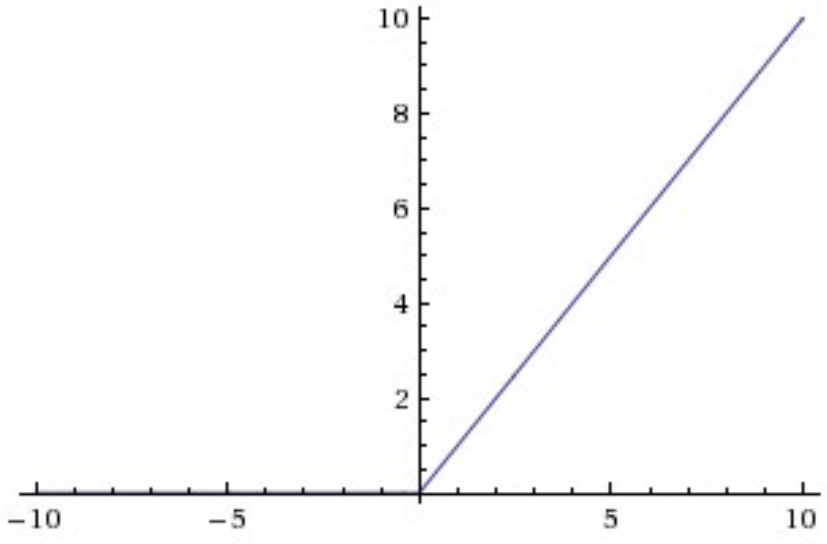
Commonly used activation functions:



The tanh non-linearity.

- $\mathbb{R} \mapsto [-1, 1]$.
- Output is zero-centered!
- But its activations also saturate.
- Tanh neuron is simply a scaled sigmoid neuron.
- $\tanh(x) = 2\sigma(2x) - 1$.

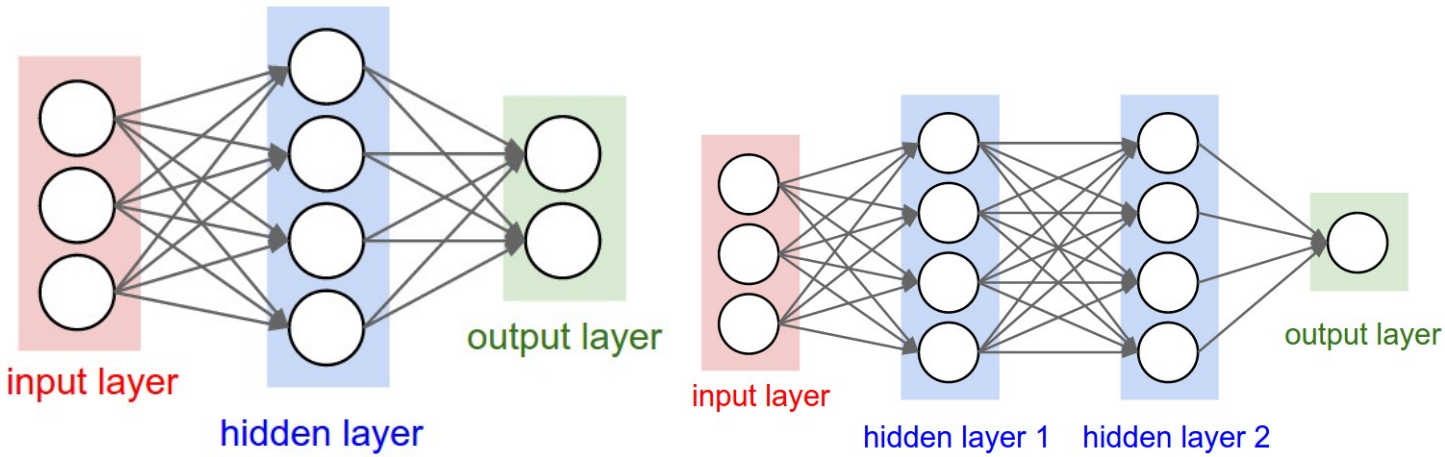
Commonly used activation functions:



Rectified Linear Unit (ReLU).

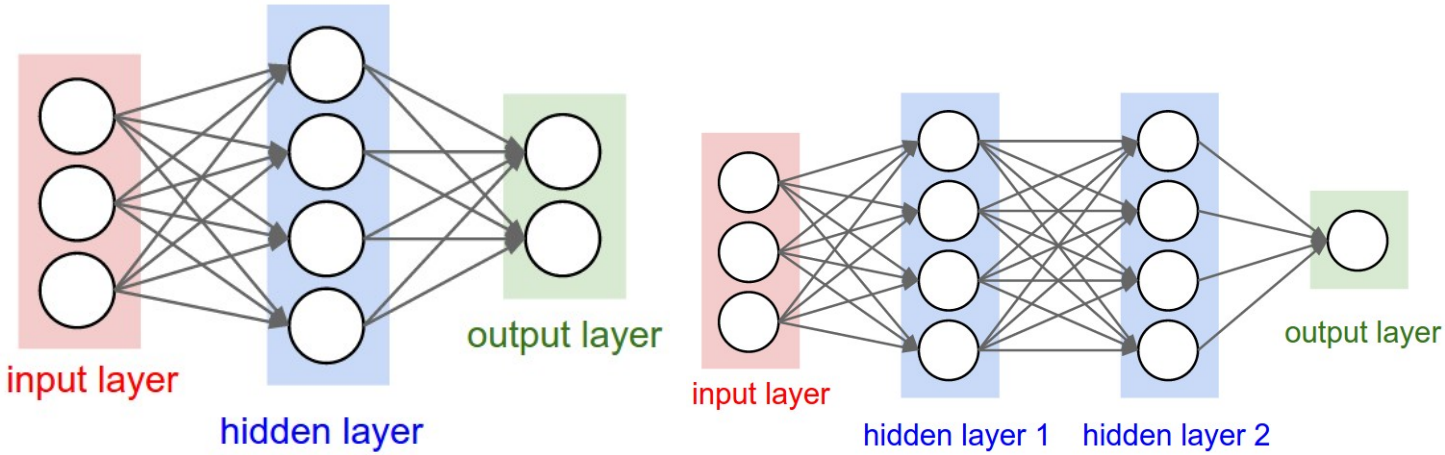
- $f(x) = max(0, x)$.
- Greatly accelerate (*e.g.* a factor of 6 in *Krizhevsky et al.*) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions.
- It does not involve expensive operations.
- Tanh neuron is simply a scaled sigmoid neuron.
- ReLU units can be fragile during training and can "die".

Neural Network / Multi-Layer Perceptrons:



A 2-layer Neural Network and a 3-layer neural network.

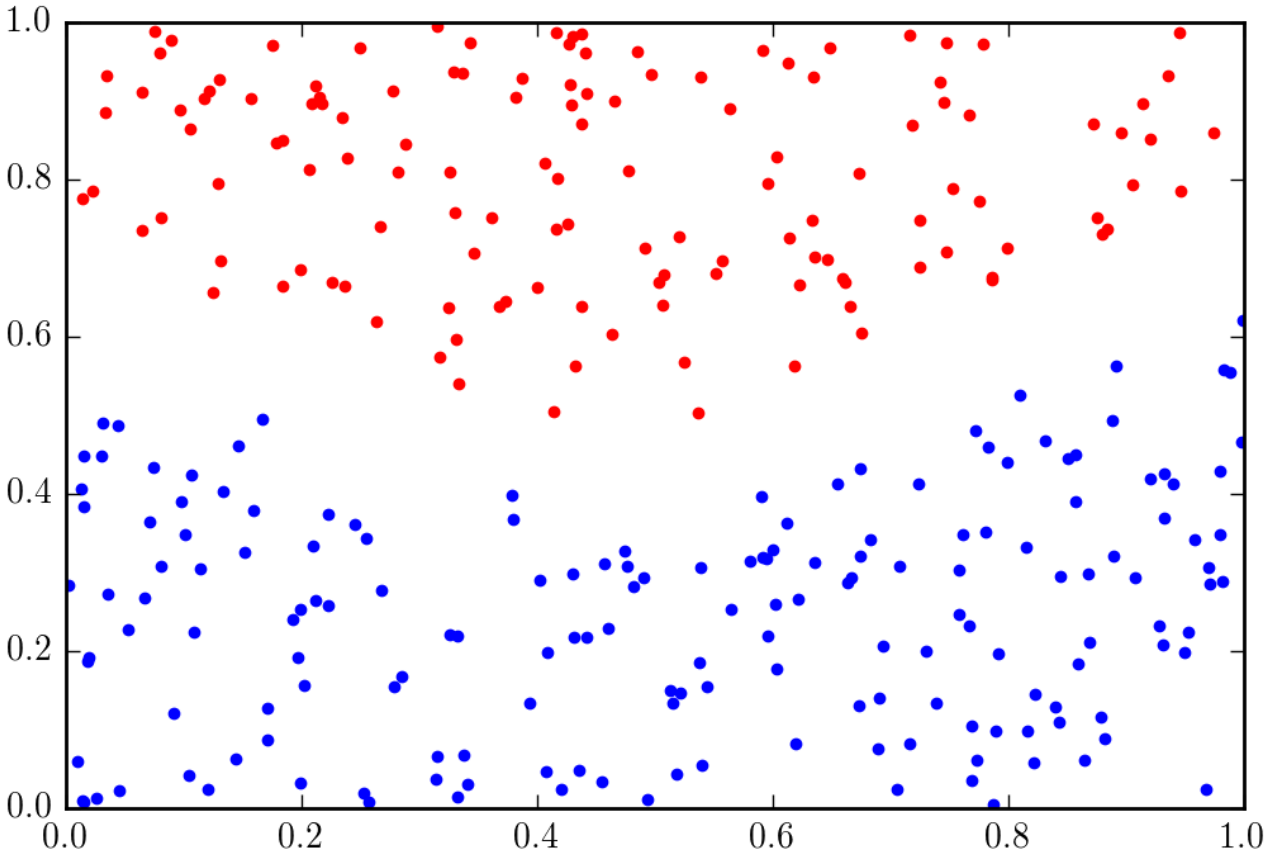
Neural Network / Multi-Layer Perceptrons:



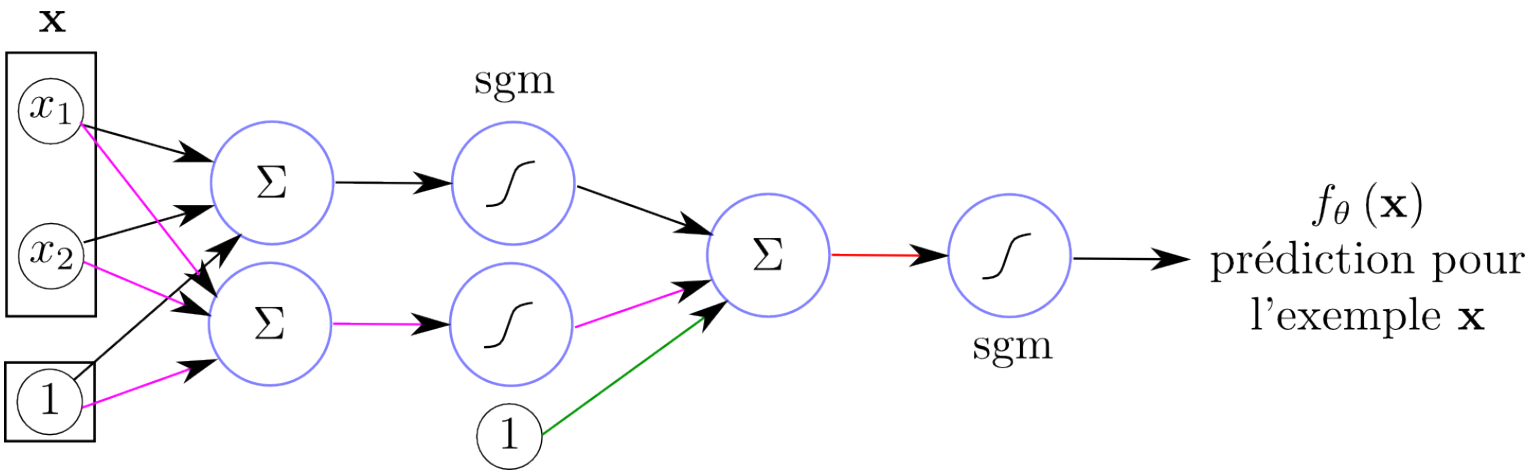
A 2-layer Neural Network and a 3-layer neural network.

Output layer: No activation function, represent the class scores (classification), or real-valued target (regression).

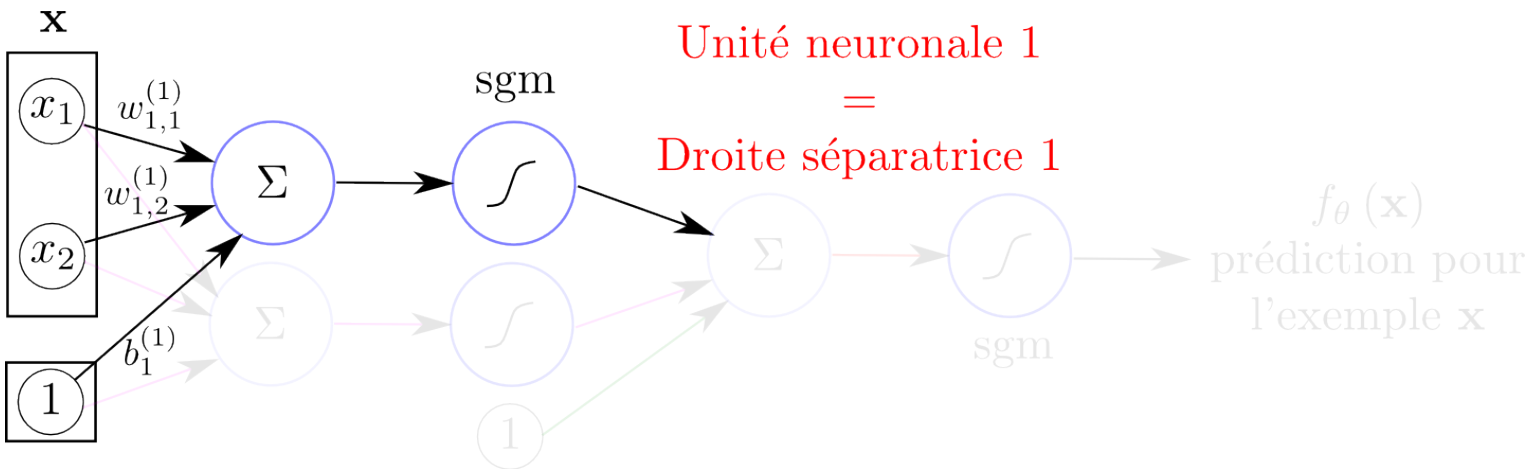
Suppose we want to learn from following data:



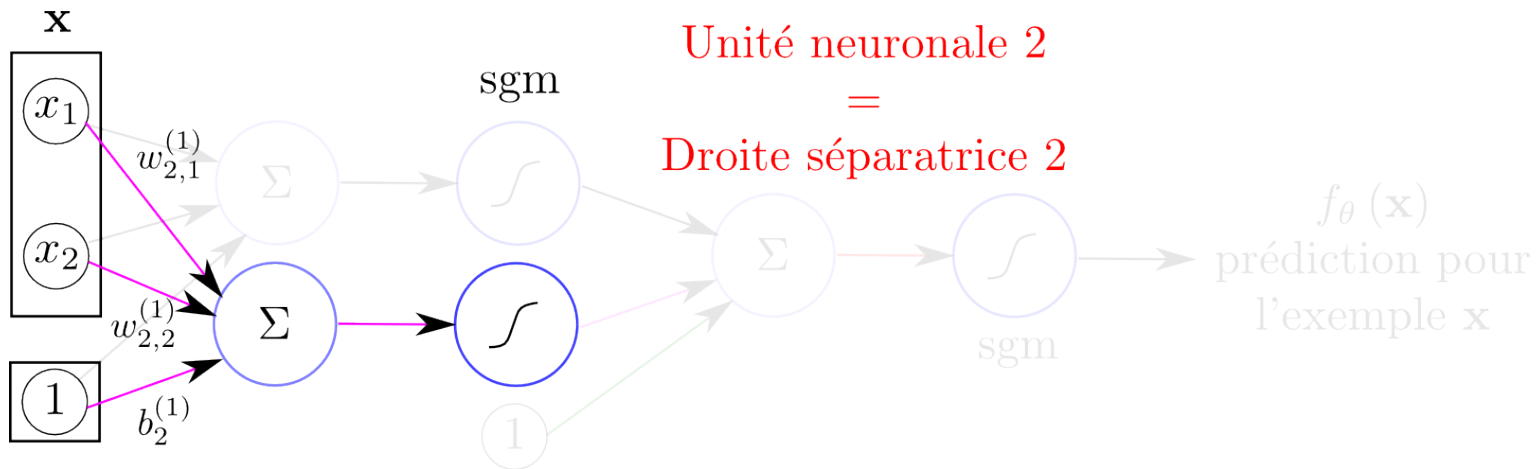
To solve the problem, we need 2 separating lines combined with AND.



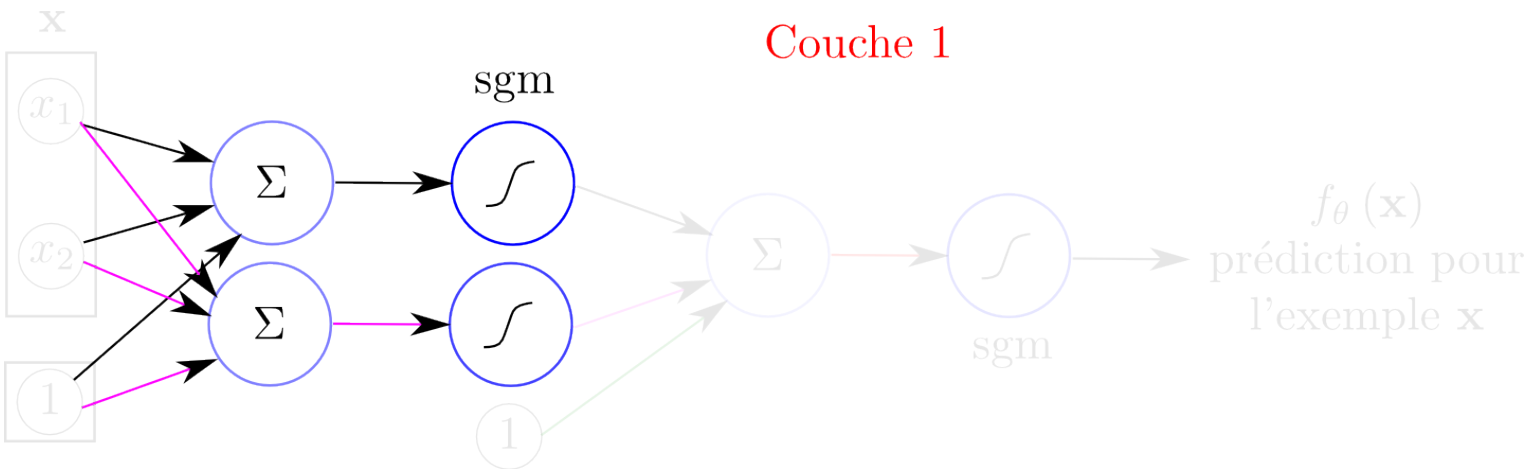
To solve the problem, we need 2 separating lines combined with AND.



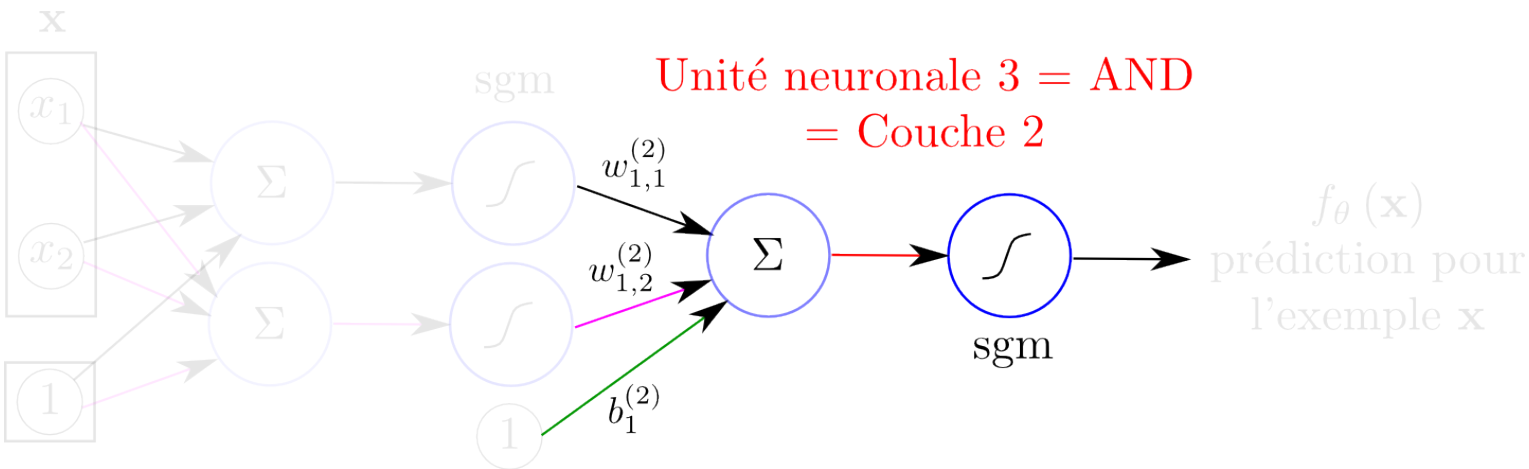
To solve the problem, we need 2 separating lines combined with AND.



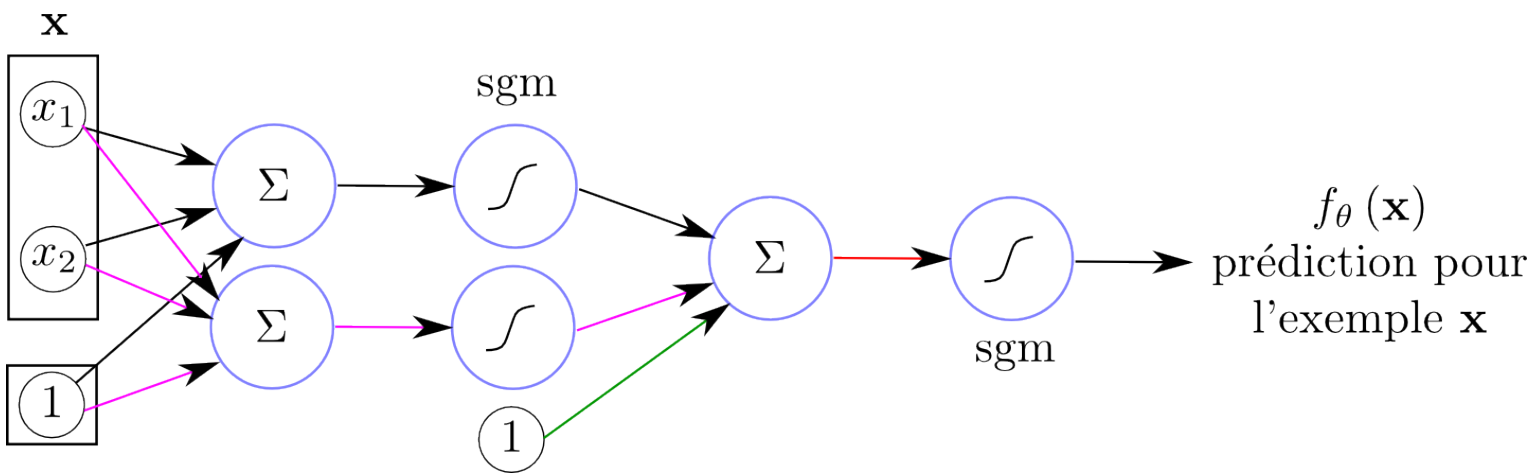
To solve the problem, we need 2 separating lines combined with AND.



To solve the problem, we need 2 separating lines combined with AND.

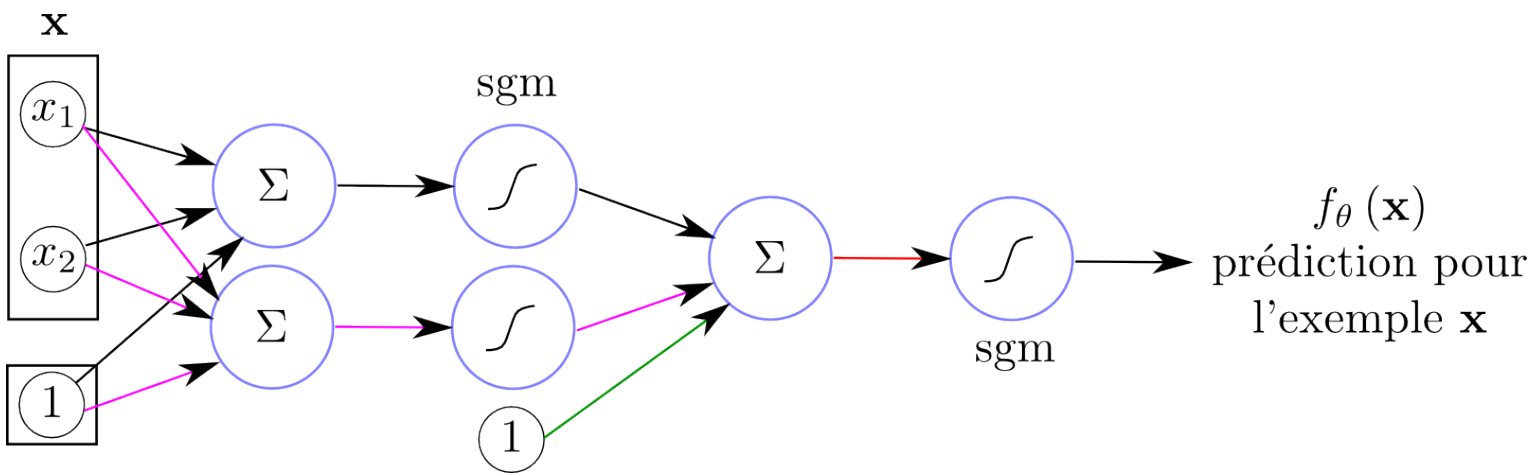


This structure is called *multi-layer perceptron* (MLP).



Parameters are indexed such as: $w_{v,u}^{(k)}$, with:

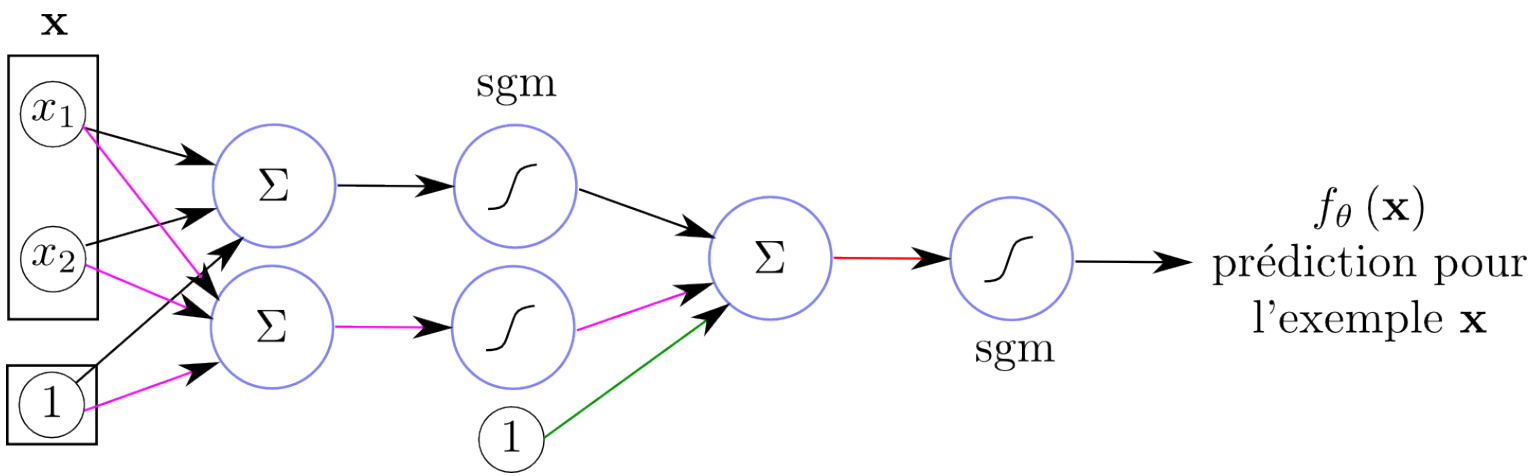
This structure is called *multi-layer perceptron* (MLP).



Parameters are indexed such as: $w_{v,u}^{(k)}$, with:

- k layer index from 1 to K.

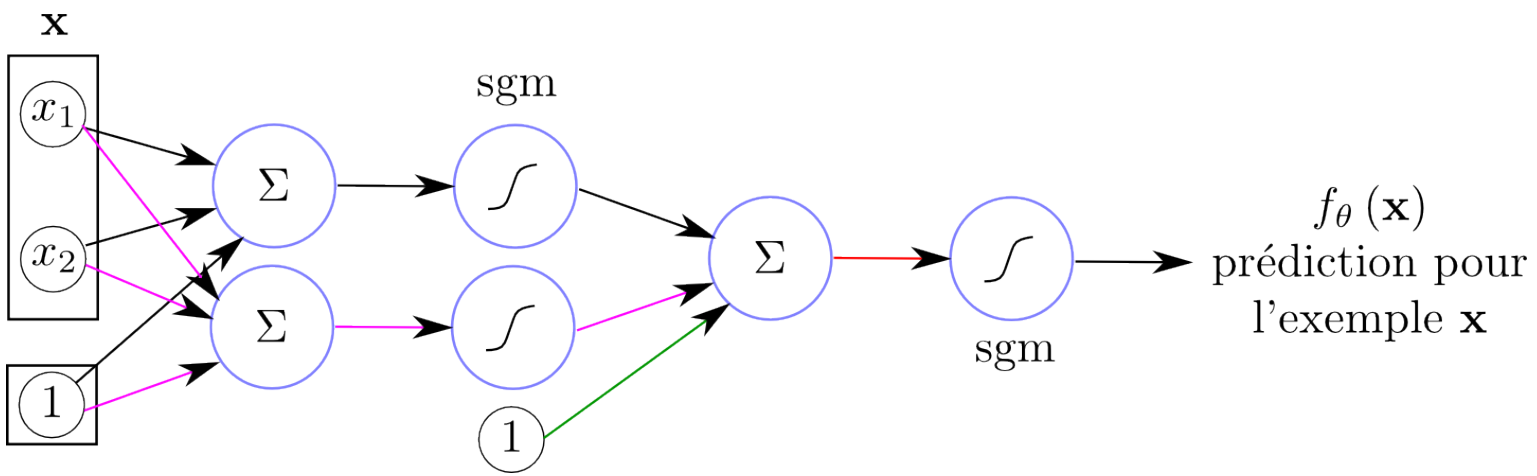
This structure is called *multi-layer perceptron* (MLP).



Parameters are indexed such as: $w_{v,u}^{(k)}$, with:

- k layer index from 1 to K.
- v input layers dimension index K.

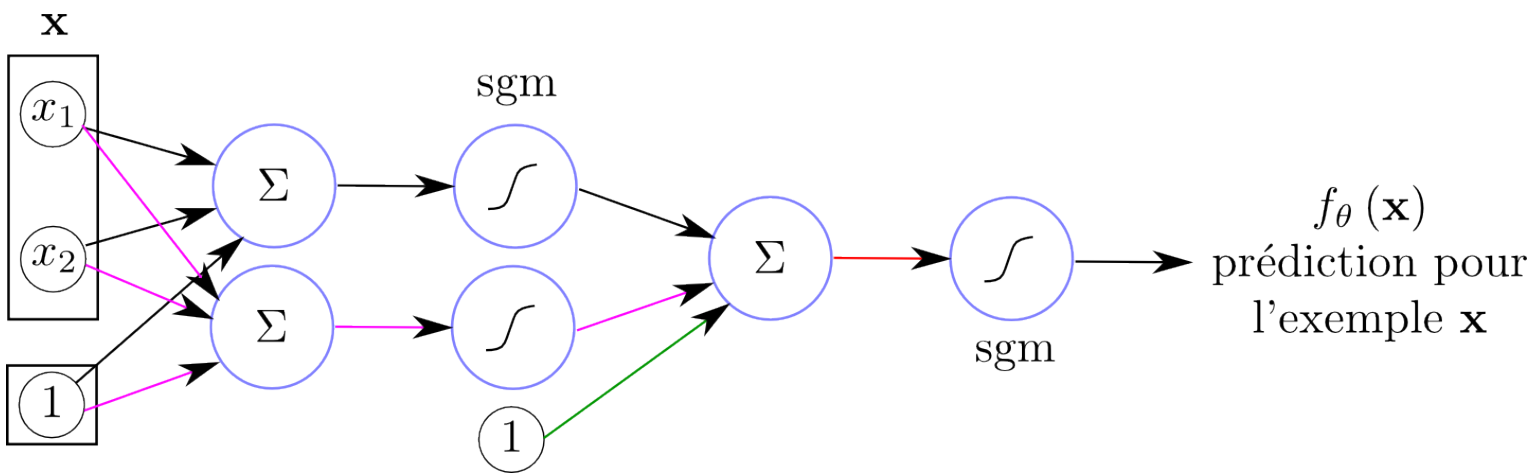
This structure is called *multi-layer perceptron* (MLP).



Parameters are indexed such as: $w_{v,u}^{(k)}$, with:

- k layer index from 1 to K.
- v input layers dimension index K. From 1 to $d^{(k)}+1$ where $d^{(k)}$ is the input vector layer size k.

This structure is called *multi-layer perceptron* (MLP).

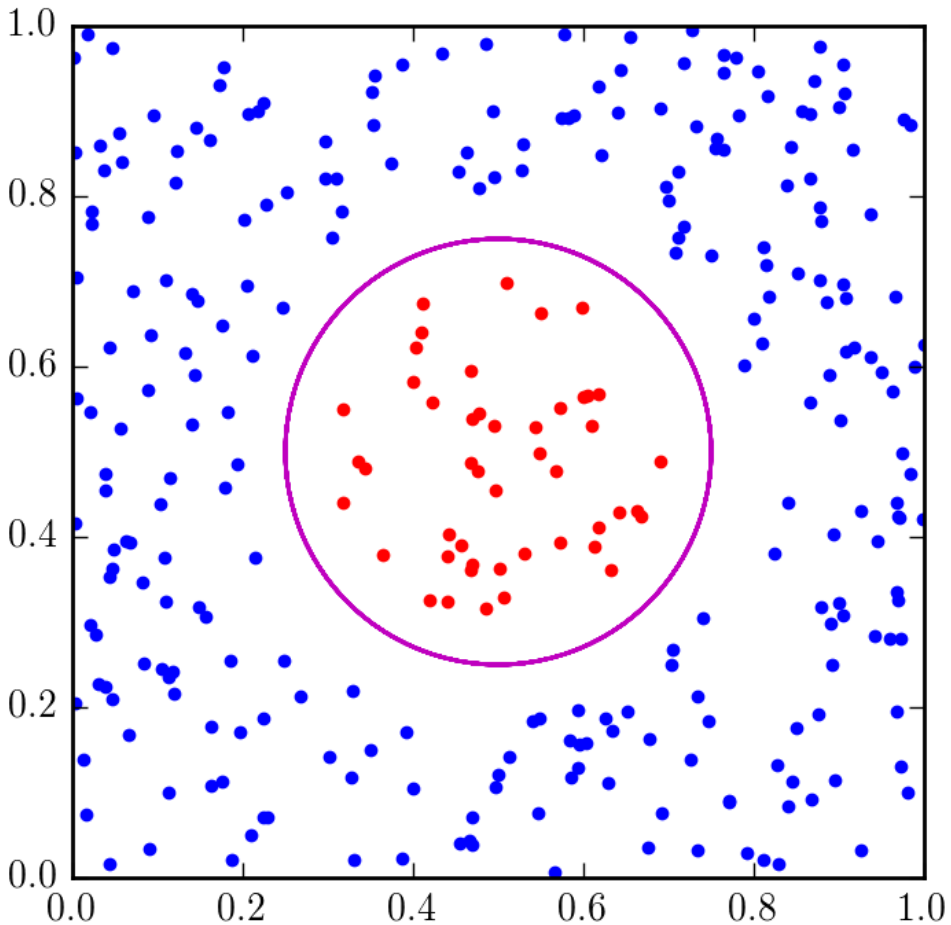


Parameters are indexed such as: $w_{v,u}^{(k)}$, with:

- k layer index from 1 to K.
- v input layers dimension index K. From 1 to $d^{(k)} + 1$ where $d^{(k)}$ is the input vector layer size k.
- u neuronal unit index from 1 to $U^{(k)}$.

Neural network >> logistic regression.

Can we hope to process the following dataset?



Neural network >> logistic regression.

- Seems possible by adding lots of units to the 1^{st} layer.
- Warnings :
  - Slightly more parameters $\Rightarrow$ significantly more data needed,
  - Optimization finding $w_{v,u}^{(k)}$ is more difficult.

Neural network >> logistic regression.

- Seems possible by adding lots of units to the 1^{st} layer.
- Warnings :
  - Slightly more parameters $\Rightarrow$ significantly more data needed,
  - Optimization finding $w_{v,u}^{(k)}$ is more difficult.

How to learn $w_{v,u}^{(k)}$?

Outline

MLP: weight learning

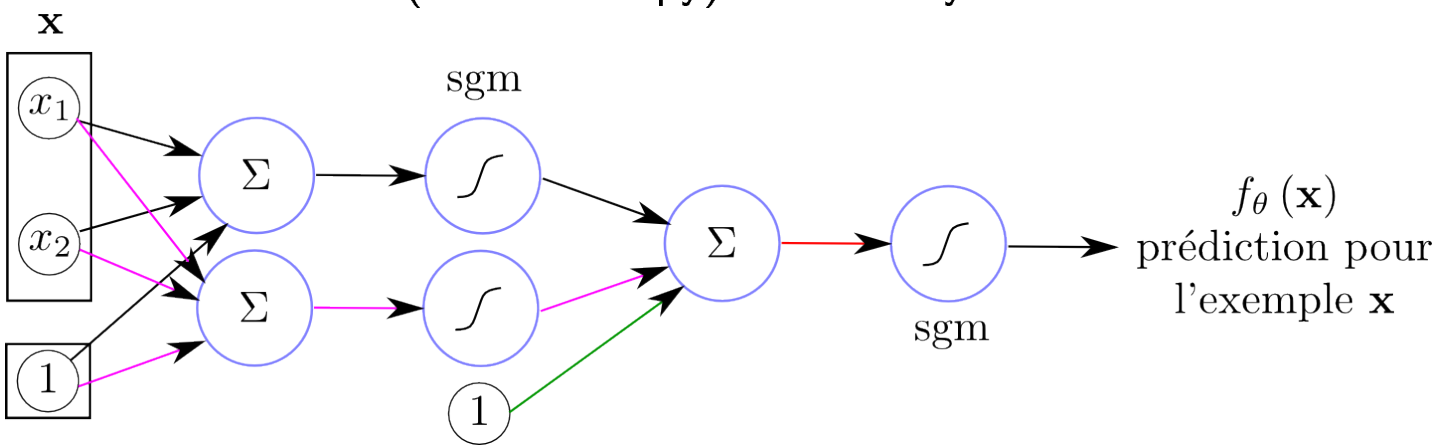
- As for logistic regression, MLP parameters are estimated by gradient descent.

MLP: weight learning

- As for logistic regression, MLP parameters are estimated by gradient descent.
- The last layer is reglog, we know that a good cost function is the corresponding NLL.

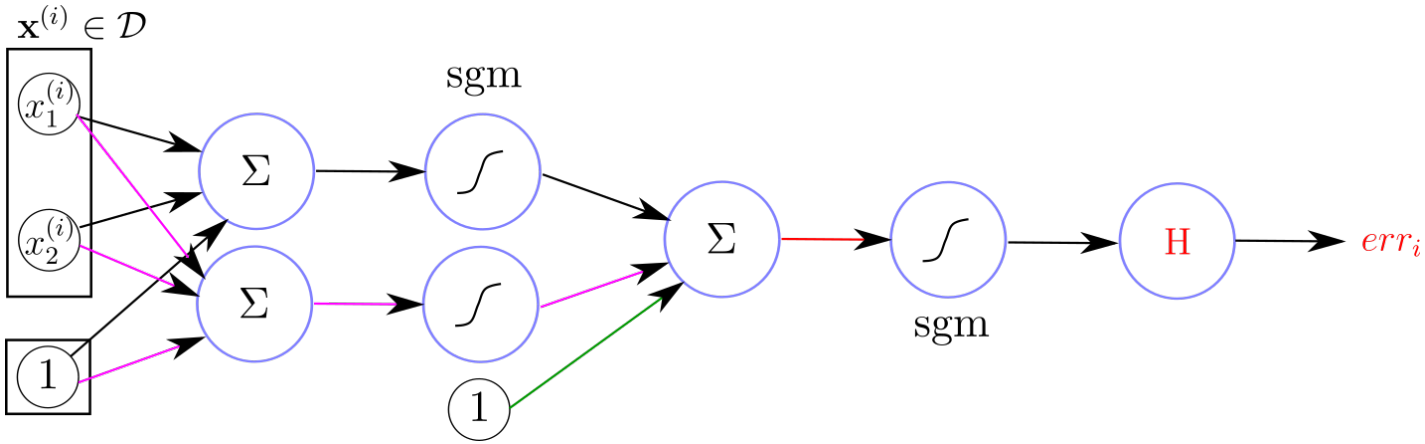
MLP: weight learning

- As for logistic regression, MLP parameters are estimated by gradient descent.
- The last layer is reglog, we know that a good cost function is the corresponding NLL.
- Include the loss H (cross entropy) as final layer:



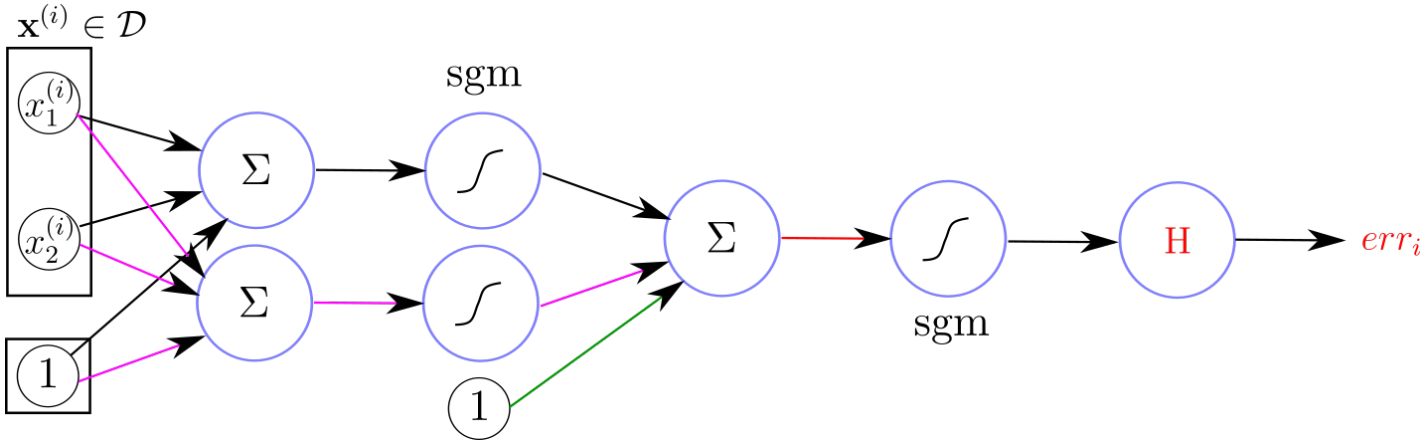
MLP: weight learning

- As for logistic regression, MLP parameters are estimated by gradient descent.
- The last layer is reglog, we know that a good cost function is the corresponding NLL.
- Include the loss H (cross entropy) as final layer:



MLP: weight learning

- As for logistic regression, MLP parameters are estimated by gradient descent.
- The last layer is reglog, we know that a good cost function is the corresponding NLL.
- Include the loss H (cross entropy) as final layer:



- The function to be optimized for the MLP is therefore:

$$J(\boldsymbol{\theta}) = \sum_i \text{cross entropy}\left(y^{(i)}, \text{predicted proba}\right) \tag{1}$$

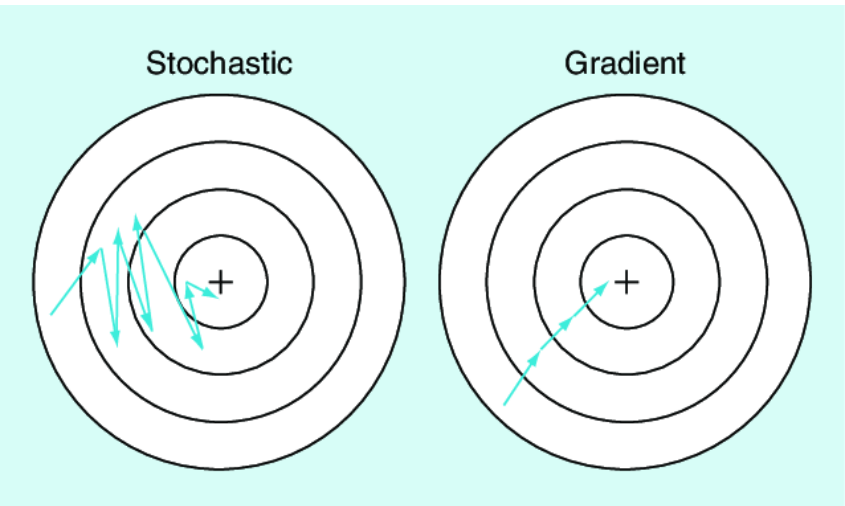
MLP: weight learning

Problem: given some function $f(\mathbf{x})$ where $\mathbf{x}$ is a vector of inputs. We are interested in computing the gradient of f at $\mathbf{x}$: $\nabla f(\mathbf{x})$.

$$\frac{df(\mathbf{x})}{dx} = \lim_{h \to 0} \frac{f(\mathbf{x}+h) - f(\mathbf{x})}{h}$$

Intuition: h is very small, then the function is well-approximated by a straight line, and the derivative is its slope.

$$f(\mathbf{x}+h) = f(\mathbf{x}) + h\frac{df(\mathbf{x})}{dx}$$



MLP: weight learning

- Simplification:

MLP: weight learning

- Simplification:
  - Derivative of sum = sum of derivative.

MLP: weight learning

- Simplification:
  - Derivative of sum = sum of derivative.
  - For the gradient descent → calculate the gradient example by example!

MLP: weight learning

- Simplification:
  - Derivative of sum = sum of derivative.
  - For the gradient descent → calculate the gradient example by example!
  - Let's consider temporarily that:

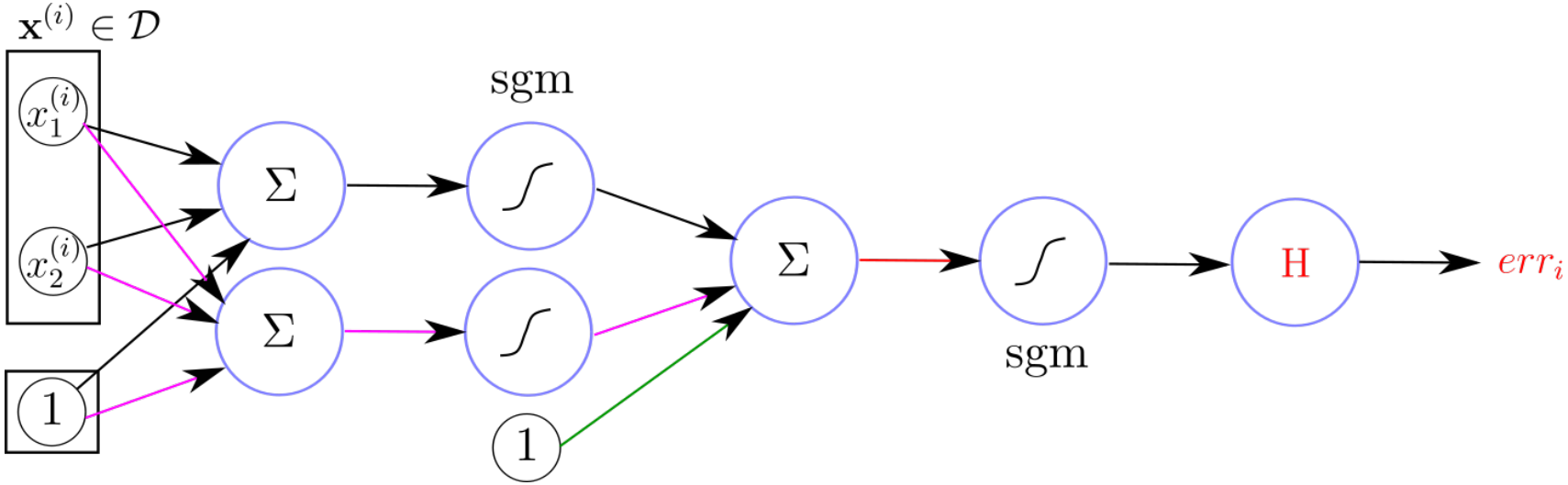
$$J(\theta) = \text{cross entropy}\left(\text{predicted proba}, y^{(i)}\right)$$

MLP: weight learning

- Let's introduce additional variables:

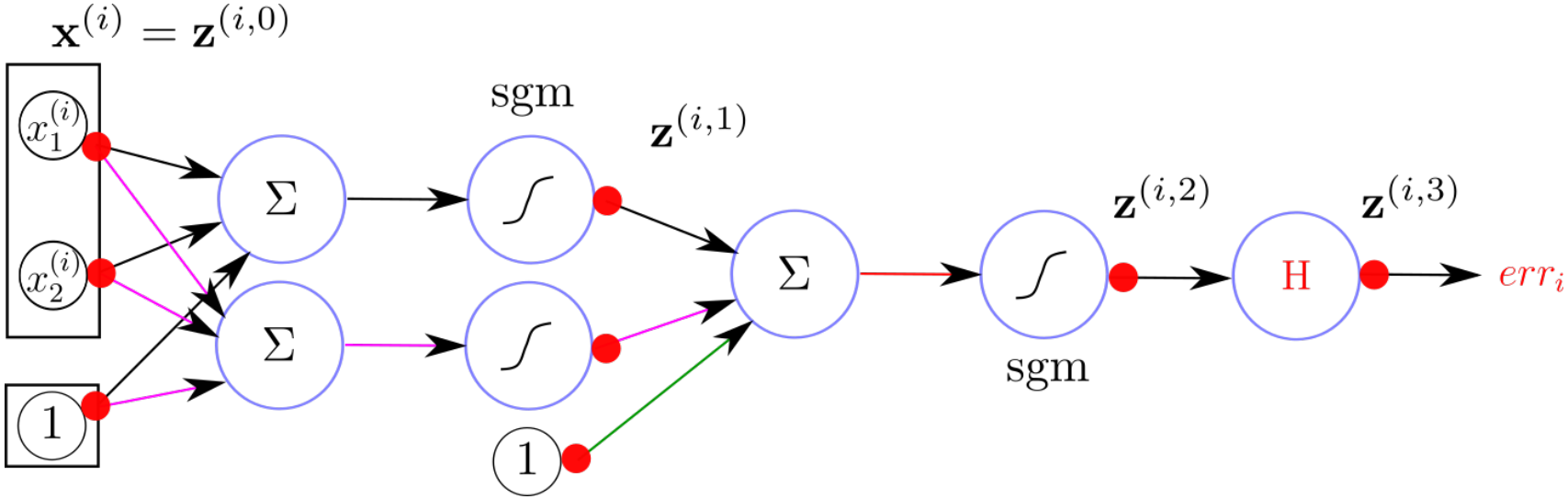
MLP: weight learning

- Let's introduce additional variables:



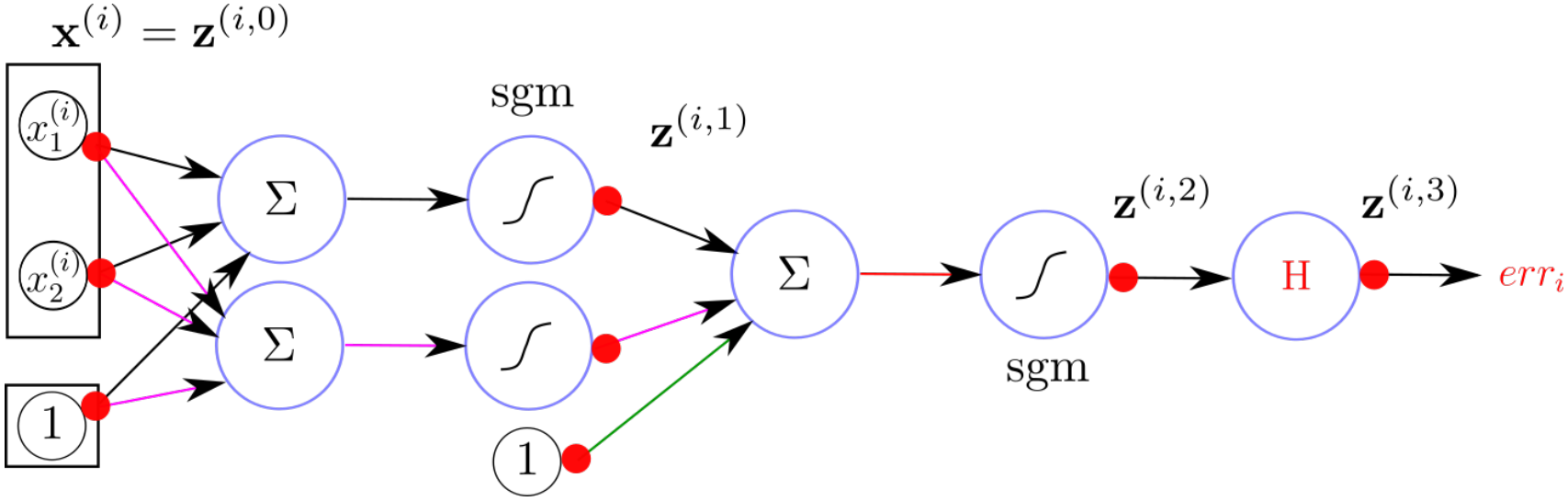
MLP: weight learning

- Let's introduce additional variables:



MLP: weight learning

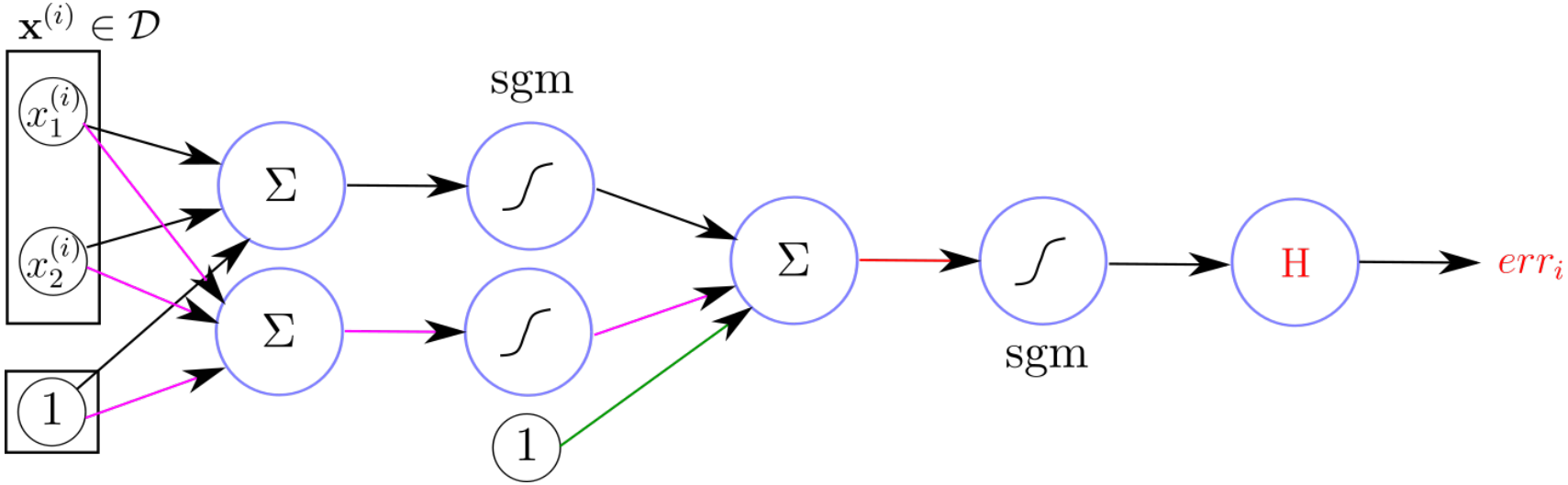
- Let's introduce additional variables:



- $\mathbf{z}^{(\mathbf{i},\mathbf{k})}$: output layer k.

MLP: weight learning

- Finally, learning parameters:



MLP: weight learning: gradient descent

- Let's simplify again:

MLP: weight learning: gradient descent

- Let's simplify again:
  - We calculate the error err_i derivative with respect to each parameter $\theta_j^{(k)}$ using just one forward pass through the network

MLP: weight learning: gradient descent

- Let's simplify again:
  - We calculate the error err_i derivative with respect to each parameter $\theta_j^{(k)}$ using just one forward pass through the network
  - Last layer has no parameter!

MLP: weight learning: gradient descent

- Let's simplify again:
  - We calculate the error err_i derivative with respect to each parameter $\theta_j^{(k)}$ using just one forward pass through the network
  - Last layer has no parameter!
  - We can calculate the output derivative with respect to its input:

$$\frac{d}{d\mathbf{z}^{(\mathbf{i},\mathbf{2})}} err_i\left(\boldsymbol{\theta}\right) = \frac{d}{d\mathbf{z}^{(\mathbf{i},\mathbf{2})}} \mathbf{z}^{(\mathbf{i},\mathbf{3})} =$$

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer.

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer.
  - Contains 3 parameters : $w_{1,1}^{(2)}$, $w_{2,1}^{(2)}$ et $b_1^{(2)}$.

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer.
  - Contains 3 parameters : $w_{1,1}^{(2)}$, $w_{2,1}^{(2)}$ et $b_1^{(2)}$.
  - For exemple compute:

$$\frac{\partial}{\partial w_{1,1}^{(2)}} err_i(\boldsymbol{\theta}) = \frac{\partial}{\partial w_{1,1}^{(2)}} \mathbf{z}^{(\mathbf{i},\mathbf{3})} =$$

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer.
  - Contains 3 parameters : $w_{1,1}^{(2)}$, $w_{2,1}^{(2)}$ et $b_1^{(2)}$.
  - For exemple compute:

$$\frac{\partial}{\partial w_{1,1}^{(2)}} err_i\left(\boldsymbol{\theta}\right) = \frac{\partial}{\partial w_{1,1}^{(2)}} \mathbf{z}^{(\mathbf{i},\mathbf{3})} =$$

  - We can compute the output derivative according to its inputs, for example:

$$\frac{\partial}{\partial z_1^{(i,1)}} \mathbf{z}^{(\mathbf{i},\mathbf{2})} =$$

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer (the 1^{st}).

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer (the 1^{st}).
  - Contains 6 parameters :
    - $w_{1,1}^{(1)}$, $w_{2,1}^{(1)}$ et $b_1^{(1)}$ for the 1^{st} neuron,

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer (the 1^{st}).
  - Contains 6 parameters :
    - $w_{1,1}^{(1)}$, $w_{2,1}^{(1)}$ et $b_1^{(1)}$ for the 1^{st} neuron,
    - $w_{1,2}^{(1)}$, $w_{2,2}^{(1)}$ et $b_2^{(1)}$ for the 2^{sd} neuron.

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer (the 1^{st}).
  - Contains 6 parameters :
    - $w_{1,1}^{(1)}$, $w_{2,1}^{(1)}$ et $b_1^{(1)}$ for the 1^{st} neuron,
    - $w_{1,2}^{(1)}$, $w_{2,2}^{(1)}$ et $b_2^{(1)}$ for the 2^{sd} neuron.
  - For exemple compute:

$$\frac{\partial}{\partial w_{1,1}^{(1)}} err_i(\boldsymbol{\theta}) = \frac{\partial}{\partial w_{1,1}^{(1)}} \mathbf{z}^{(\mathbf{i},\mathbf{3})} =$$

MLP: weight learning: gradient descent

- Let's simplify again:
  - Previous layer (the 1^{st}).
  - Contains 6 parameters :
    - $w_{1,1}^{(1)}$, $w_{2,1}^{(1)}$ et $b_1^{(1)}$ for the 1^{st} neuron,
    - $w_{1,2}^{(1)}$, $w_{2,2}^{(1)}$ et $b_2^{(1)}$ for the 2^{sd} neuron.
  - For exemple compute:

$$\frac{\partial}{\partial w_{1,1}^{(1)}} err_i(\theta) = \frac{\partial}{\partial w_{1,1}^{(1)}} \mathbf{z}^{(\mathbf{i},\mathbf{3})} =$$

  - **And it's done!!**

MLP: weight learning: gradient descent

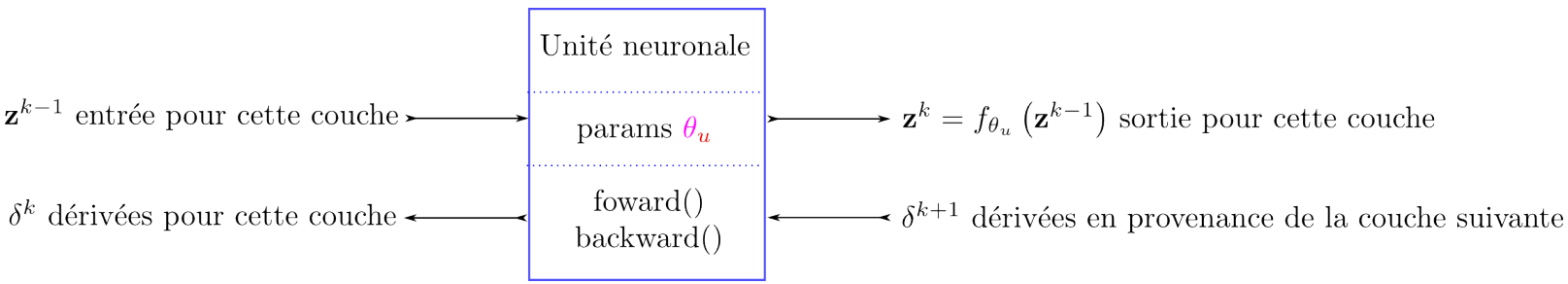
- A way of computing gradients of expressions through recursive application of **chain rule**, from the last to the first layer, hence it is called backpropagation!

MLP: weight learning: gradient descent

- A way of computing gradients of expressions through recursive application of **chain rule**, from the last to the first layer, hence it is called backpropagation!
- MLP is well suited for OOP principles:

MLP: weight learning: gradient descent

- A way of computing gradients of expressions through recursive application of **chain rule**, from the last to the first layer, hence it is called backpropagation!
- MLP is well suited for OOP principles:
  - We can create aclass neuron:

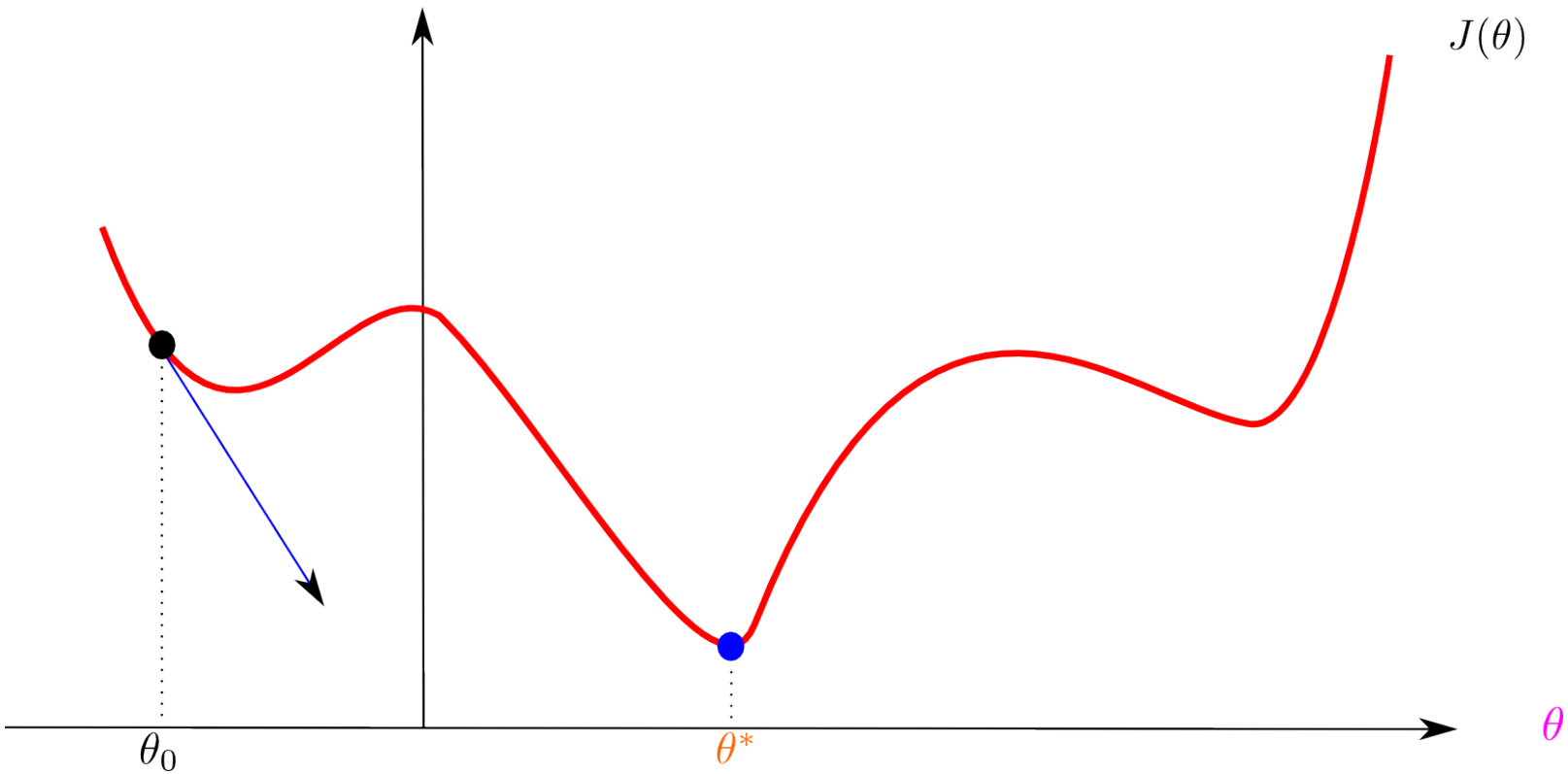


MLP: weight learning: gradient descent

- Will the gradient descent go well?

MLP: weight learning: gradient descent

- Will the gradient descent go well?



MLP: weight learning: gradient descent

- The gradient descent converges toward a local minimum.

MLP: weight learning: gradient descent

- The gradient descent converges toward a local minimum.
- It depends oninitialisation $\boldsymbol{\theta}_0$.

MLP: weight learning: gradient descent

- The gradient descent converges toward a local minimum.
- It depends oninitialisation $\boldsymbol{\theta}_0$.
- The *learning rate* η tuning is not especially more difficult than for reglog.

Outline

Deep network: idea

Deep network: idea

- When classes distribution is obviously complex and performances are disappointing...
  → increase the model capacity by adding layers and neurons.
  Examples: `http://playground.tensorflow.org`.

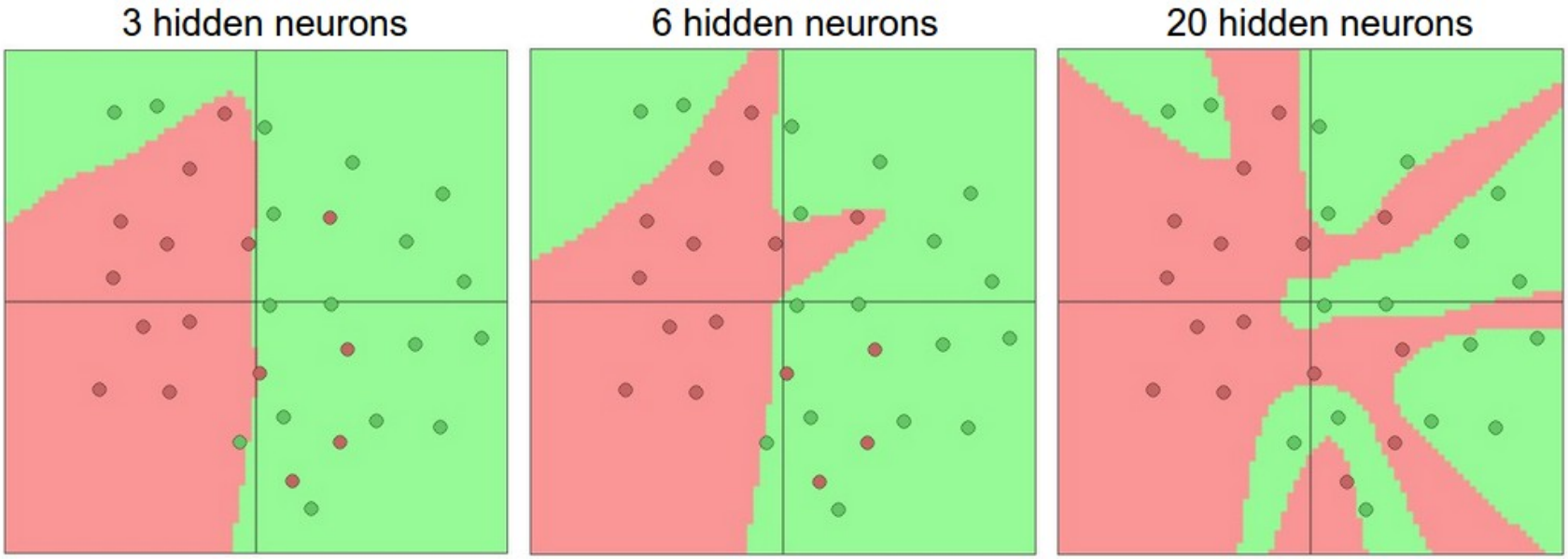
Deep network: idea

- When classes distribution is obviously complex and performances are disappointing...
  → increase the model capacity by adding layers and neurons.
  Examples: `http://playground.tensorflow.org`.
- As we saw, it may cause overfitting !

Deep network: idea

- When classes distribution is obviously complex and performances are disappointing...
  → increase the model capacity by adding layers and neurons.
  Examples: `http://playground.tensorflow.org`.
- As we saw, it may cause overfitting !
- Deep network were proposed in the 90's and were just multi-layered architectures with a reasonable number of parameters.

Deep network: overfitting



(https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

CNN: *convolutional neural network*

- The first successful deep learning algorithm was created by Yann Lecun and it alternates two kinds of layer:

CNN: *convolutional neural network*

- The first successful deep learning algorithm was created by Yann Lecun and it alternates two kinds of layer:
  1. Convolutional layer where parameters are shared between neurons,

CNN: *convolutional neural network*

- The first successful deep learning algorithm was created by Yann Lecun and it alternates two kinds of layer:
  1. Convolutional layer where parameters are shared between neurons,
  2. non-linear transmitting,

CNN: *convolutional neural network*

- The first successful deep learning algorithm was created by Yann Lecun and it alternates two kinds of layer:
  1. Convolutional layer where parameters are shared between neurons,
  2. non-linear transmitting,
  3. *Pooling* layer in order to reduce output dimension.

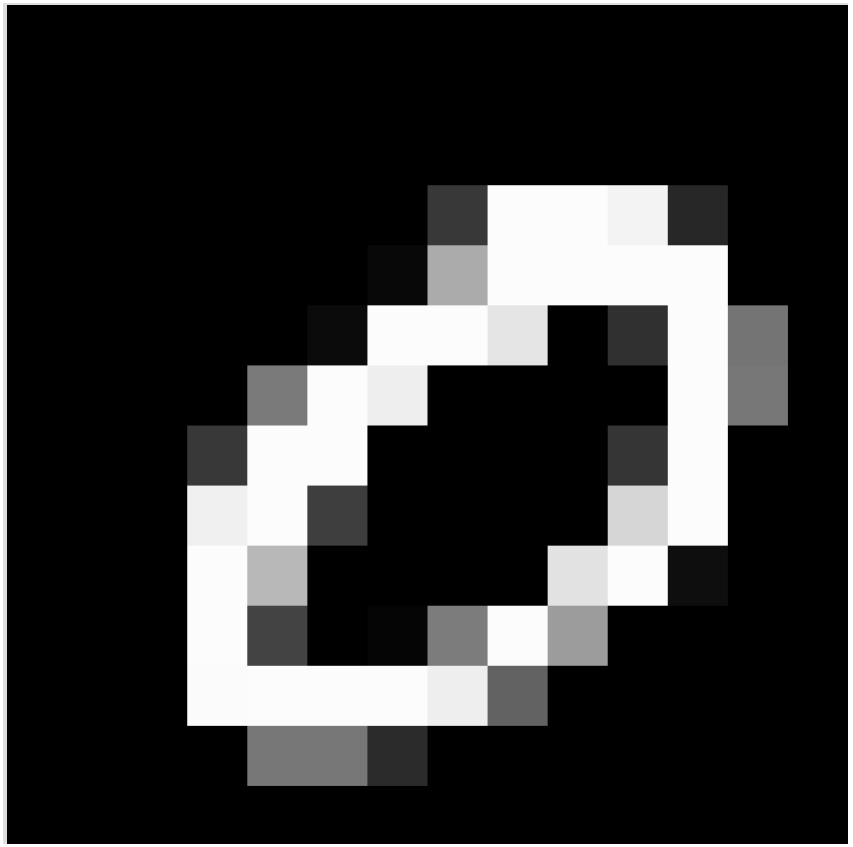
CNN: *convolutional neural network*

- The first successful deep learning algorithm was created by Yann Lecun and it alternates two kinds of layer:
  1. Convolutional layer where parameters are shared between neurons,
  2. non-linear transmitting,
  3. *Pooling* layer in order to reduce output dimension.

Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov 1998

CNN (*convolutional neural network*): convolution

- Suppose that examples $\mathbf{x}^{(\mathbf{i})}$ are images:



CNN (*convolutional neural network*): convolution

- Suppose that examples $\mathbf{x}^{(\mathbf{i})}$ are images:

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	56	252	252	243	39	0	0
0	0	0	0	0	0	8	171	252	252	252	252	0	0
0	0	0	0	0	11	252	252	229	0	48	252	116	0
0	0	0	0	122	252	238	0	0	0	0	252	119	0
0	0	0	56	252	252	0	0	0	0	53	252	0	0
0	0	0	240	252	62	0	0	0	0	214	252	0	0
0	0	0	252	184	0	0	0	0	226	252	14	0	0
0	0	0	252	67	0	5	124	252	156	0	0	0	0
0	0	0	251	252	252	252	238	98	0	0	0	0	0
0	0	0	0	119	119	43	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

CNN (*convolutional neural network*): convolution

- Convolution operator between two vector is $\mathbf{x}$ and $\mathbf{y}$ is denoted $\star$.

CNN (*convolutional neural network*): convolution

- Convolution operator between two vector is $\mathbf{x}$ and $\mathbf{y}$ is denoted $\star$.
- The result is also a vector $\mathbf{z} = \mathbf{x} \star \mathbf{y}$ and:

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \tag{2}$$

CNN (*convolutional neural network*): convolution

- Convolution operator between two vector is $\mathbf{x}$ and $\mathbf{y}$ is denoted $\star$.
- The result is also a vector $\mathbf{z} = \mathbf{x} \star \mathbf{y}$ and:

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \tag{2}$$

Convolution is similar to adot product where one of the vector is "mirrored"!

CNN (*convolutional neural network*): convolution

- Convolution operator between two vector is $\mathbf{x}$ and $\mathbf{y}$ is denoted $\star$.
- The result is also a vector $\mathbf{z} = \mathbf{x} \star \mathbf{y}$ and:

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \tag{2}$$

Convolution is commutative: $\mathbf{x} \star \mathbf{y} = \mathbf{y} \star \mathbf{x}$.

CNN (*convolutional neural network*): convolution

- Convolution operator between two vector is $\mathbf{x}$ and $\mathbf{y}$ is denoted $\star$.
- The result is also a vector $\mathbf{z} = \mathbf{x} \star \mathbf{y}$ and:

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \tag{2}$$

Convolution is associative: $\mathbf{x} \star (\mathbf{y} \star \mathbf{u}) = (\mathbf{x} \star \mathbf{y}) \star \mathbf{u}$.

CNN (*convolutional neural network*): convolution

- Convolution operator between two vector is $\mathbf{x}$ and $\mathbf{y}$ is denoted $\star$.
- The result is also a vector $\mathbf{z} = \mathbf{x} \star \mathbf{y}$ and:

$$z_i = \sum_{k=1}^{\text{len}(y)} x_{i-k} y_k \tag{2}$$

Convolution is distributive over the sum: $\mathbf{x} \star (\mathbf{y} + \mathbf{u}) = \mathbf{x} \star \mathbf{y} + \mathbf{x} \star \mathbf{u}$.

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image $\mathbf{x}$

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image $\mathbf{x}$

-1	0	$+1$
-1	0	$+1$
-1	0	$+1$

filter $\mathbf{y}$

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image $\mathbf{x}$

-1	0	$+1$
-1	0	$+1$
-1	0	$+1$

filter $\mathbf{y}$

-1		

convolved image $\mathbf{x} \star \mathbf{y}$

CNN (*convolutional neural network*): convolution

- Convolution on a binary image::

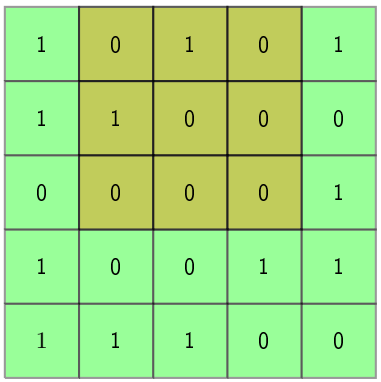
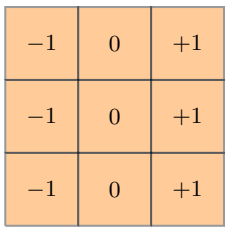
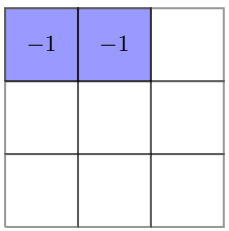


image $\mathbf{x}$



filter $\mathbf{y}$



convolved image $\mathbf{x} \star \mathbf{y}$

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image $\mathbf{x}$

-1	0	$+1$
-1	0	$+1$
-1	0	$+1$

filter $\mathbf{y}$

-1	-1	$+1$

convolved image $\mathbf{x} \star \mathbf{y}$

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image $\mathbf{x}$

-1	0	$+1$
-1	0	$+1$
-1	0	$+1$

filter $\mathbf{y}$

-1	-1	$+1$
-2		

convolved image $\mathbf{x} \star \mathbf{y}$

CNN (*convolutional neural network*): convolution

- Convolution on a binary image:

1	0	1	0	1
1	1	0	0	0
0	0	0	0	1
1	0	0	1	1
1	1	1	0	0

image $\mathbf{x}$

-1	0	$+1$
-1	0	$+1$
-1	0	$+1$

filter $\mathbf{y}$

-1	-1	$+1$
-2	0	$+2$
-1	0	$+1$

... etc. convolved image $\mathbf{x} \star \mathbf{y}$

CNN (*convolutional neural network*): convolution

Edge Handling:

- **Extend**
  The nearest border pixels are conceptually extended as far as necessary to provide values for the convolution.
- **Wrap**
  The image is conceptually wrapped.
- **Mirror**
  The image is conceptually mirrored at the edges.
- **Crop**
  Any pixel in the output image which would require values from beyond the edge is skipped.
- **Kernel Crop**
  Any pixel in the kernel that extends past the input image isn't used and the normalizing is adjusted to compensate.

Convolution: Examples

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Kernel:
Identity

Original

Results

Convolution: Examples

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Kernel:
Edges detection

Original

Results

Convolution: Examples

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Kernel:
Edges detection

Original

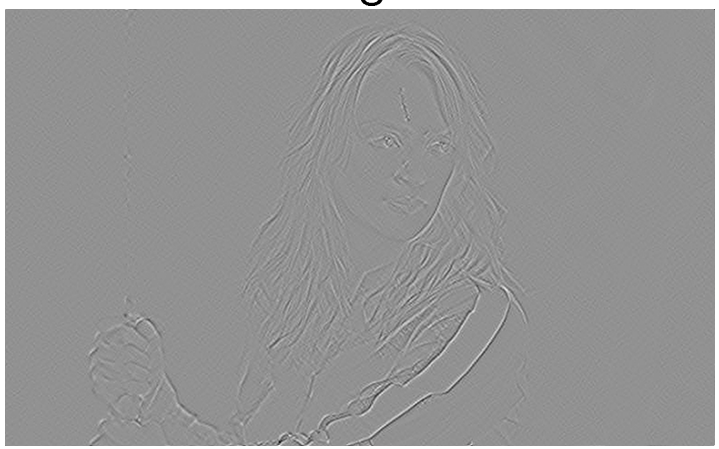
Results

Convolution: Examples

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

Kernel:
Edges detection

Original



Results

Convolution: Examples

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Kernel:
Sharpen

Original

Results

Convolution: Examples

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Kernel:
Gaussien blur

Original

Results

CNN (*convolutional neural network*): convolutional layer parameters

- depth: number of filters used in the layer.
  - In previous example, we have used 1 filter. It is common to use many of them in parallel.
  - The filter at position m possess its own parameters $\mathbf{w}^{(k,m)}$.
  - Each filter generate a convolved image $\mathbf{z}^{(\mathbf{k},m)}$ called *feature map*.
  - These images are agregated by simple addition in following layers:

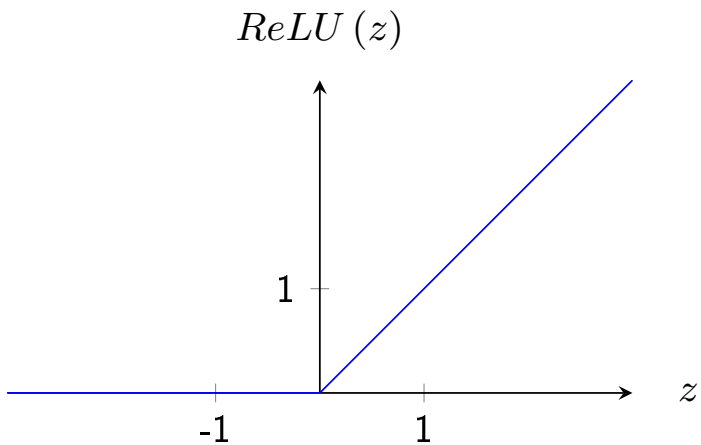
$$\mathbf{z}^{(\mathbf{k},1)} = \mathbf{f}_{\mathbf{act}}\left(\sum_{m=1}^{\mathbf{depth}^{(\mathbf{k-1})}} \mathbf{w}^{(\mathbf{k},1)} \star \mathbf{z}^{(\mathbf{k-1},m)}\right) = \mathbf{f}_{\mathbf{act}}\left(\mathbf{w}^{(\mathbf{k},1)} \star \sum_{m=1}^{\mathbf{depth}^{(\mathbf{k-1})}} \mathbf{z}^{(\mathbf{k-1},m)}\right).$$

- stride: "jump" *stride* pixels between each convolution iteration.

CNN (*convolutional neural network*): convolutional layer parameters

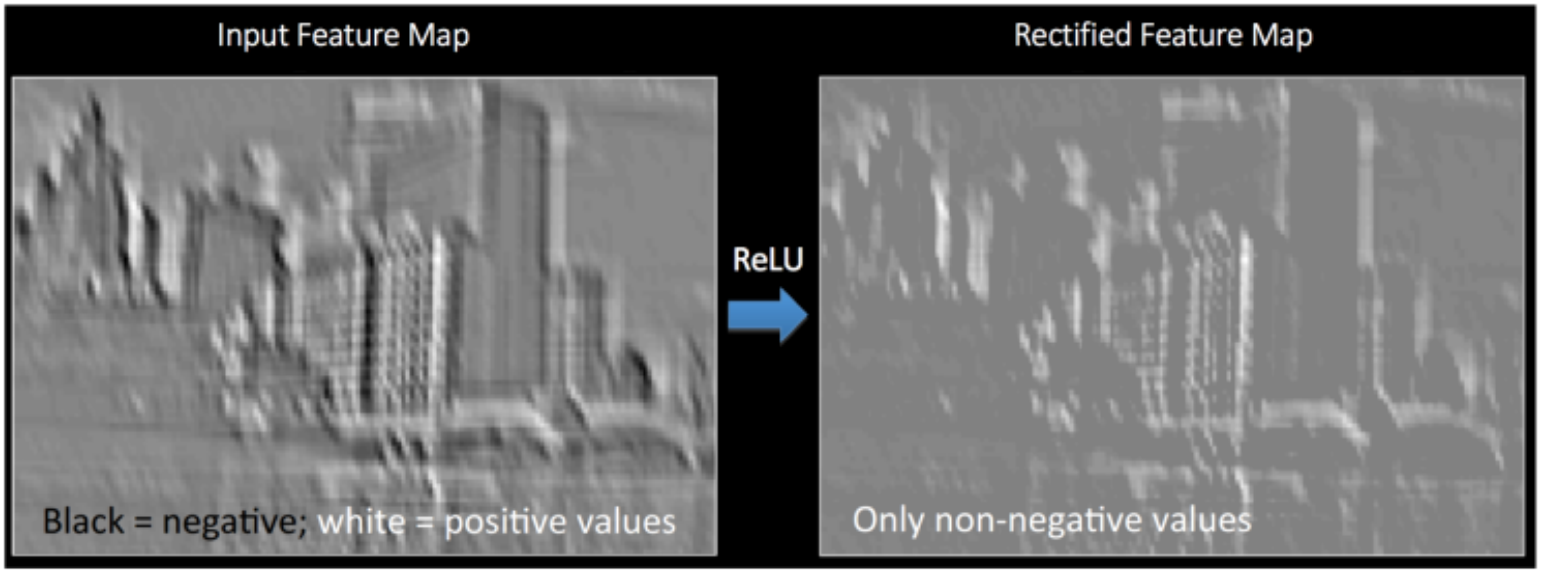
- padding: edge handling
- activation function f_{act}: often ReLU (*rectified linear unit*). Continuous function of $\mathbb{R}$ in $[0; 1]$, such as:

$$\text{ReLU}(z) = \begin{cases} z & \text{if } z \geq 0 \\ 0 & \text{else} \end{cases}. \tag{3}$$



CNN (*convolutional neural network*): activation function

- Consists to keep positive value of the feature map.



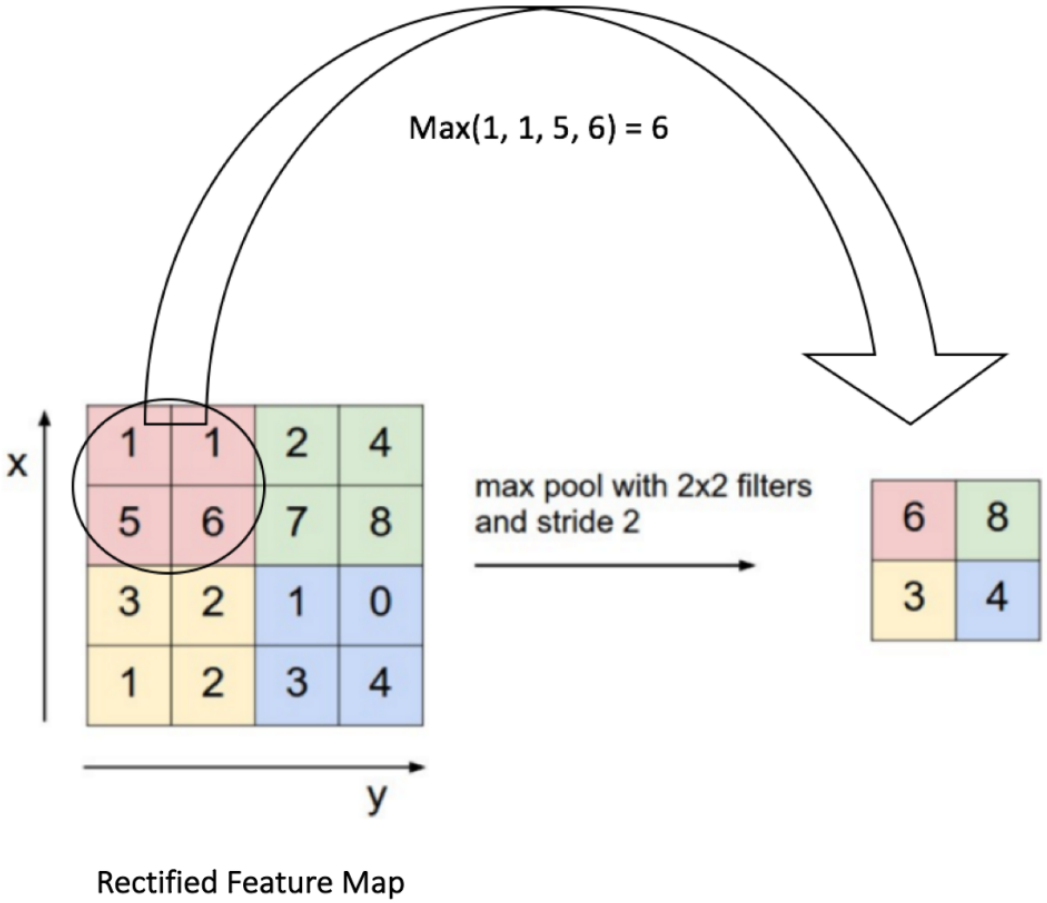
(source : http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf**Rob Fergus - Facebook AI research)**

CNN (*convolutional neural network*): pooling

Consists to reduce the neighborhood in the feature map (after the activation function) by taking:

- max: (*max pooling*),
- average: (*mean pooling*).

CNN (*convolutional neural network*): pooling



Max pooling example.
(source: cs231n.github.io)

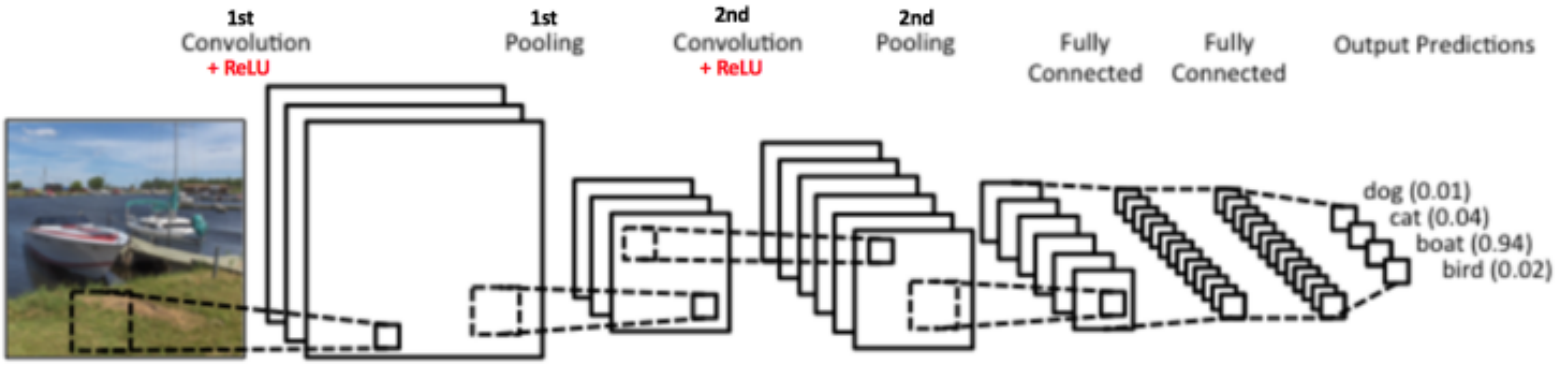
CNN (*convolutional neural network*): pooling

The pooling layer is used for:

- dimension reduction,
- to be invariant to small local distortions/translations.

CNN (*convolutional neural network*): global architecture

Putting everything together:



(source: WILDML - understanding CNNs for nlp)

CNN (*convolutional neural network*) : Backpropagation

Iterative Gradient descent principle (layer by layer) remains valid!

CNN (*convolutional neural network*) : Backpropagation

Iterative Gradient descent principle (layer by layer) remains valid!

- The derivatives of the layer *max pooling* are transfered to pixels selected,

CNN (*convolutional neural network*) : Backpropagation

Iterative Gradient descent principle (layer by layer) remains valid!

- The derivatives of the layer *max pooling* are transfered to pixels selected,
- The ReLU derivatives are very simple,

CNN (*convolutional neural network*) : Backpropagation

Iterative Gradient descent principle (layer by layer) remains valid!

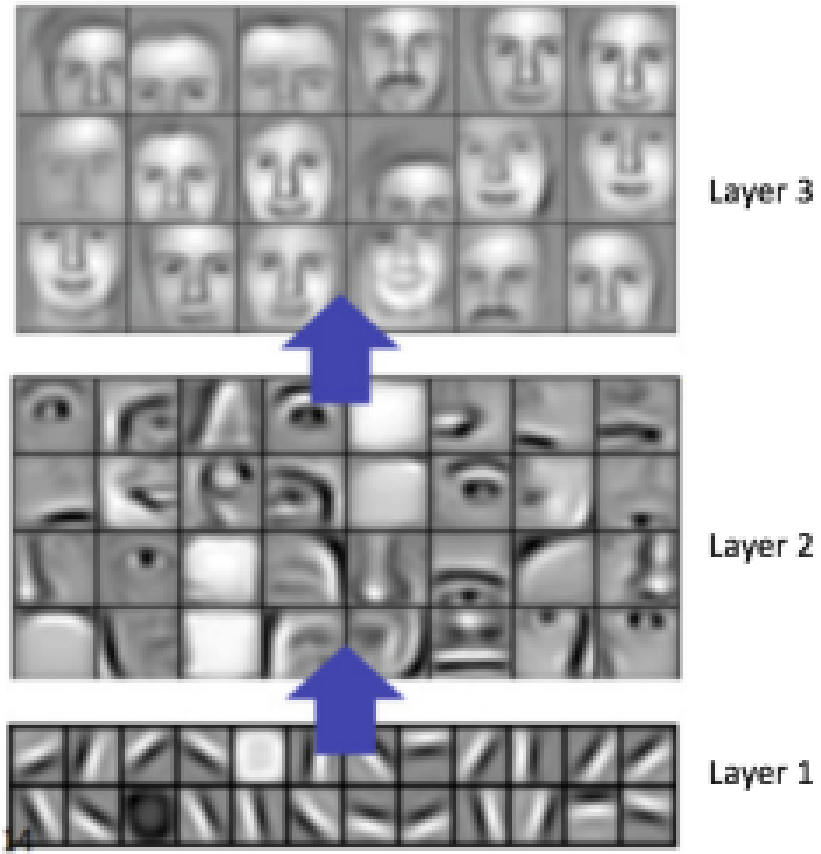
- The derivatives of the layer *max pooling* are transfered to pixels selected,
- The ReLU derivatives are very simple,
- The derivatives according to weights $\mathbf{w}^{(k,m)}$ inside convolution are also very simple to calculate.

CNN (*convolutional neural network*) : Backpropagation

Iterative Gradient descent principle (layer by layer) remains valid!

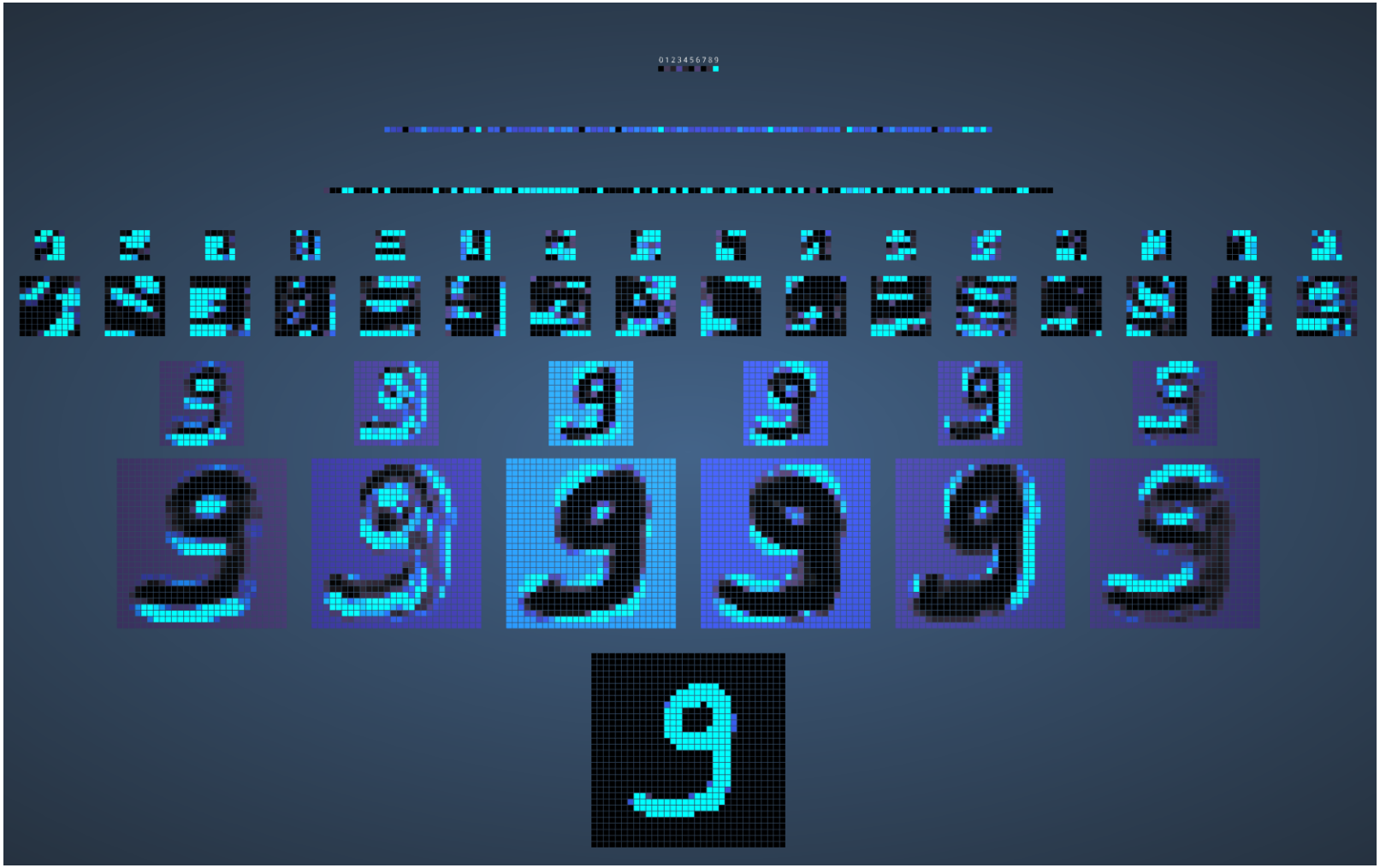
- The derivatives of the layer *max pooling* are transfered to pixels selected,
- The ReLU derivatives are very simple,
- The derivatives according to weights $\mathbf{w}^{(k,m)}$ inside convolution are also very simple to calculate.
- The $\mathbf{w}^{(k,m)}$ of the feature map are updtate many times by iterations.

CNN (*convolutional neural network*): Filter Visualization



(source: "Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations", Lee et al., ICML 2009.)

CNN (*convolutional neural network*): Feature maps Visualization



(image générée par http://scs.ryerson.ca/ aharley/vis/conv/flat.html.)

CNN (*convolutional neural network*) / adds-on: Batch normalization

- A significant modification of the inputs at the first layer strongly impacts the model -> (standardisation).
- What about hidden layers? Can we center and reduce $\mathbf{z}^{(\mathbf{i},\mathbf{k})}$?

CNN (*convolutional neural network*) / adds-on: Batch normalization

- A significant modification of the inputs at the first layer strongly impacts the model -> (standardisation).
- What about hidden layers? Can we center and reduce $\mathbf{z}^{(\mathbf{i},\mathbf{k})}$?
- Transformation BN of the convolution output (before activation function).

CNN (*convolutional neural network*) / adds-on: Batch normalization

- A significant modification of the inputs at the first layer strongly impacts the model -> (standardisation).
- What about hidden layers? Can we center and reduce $\mathbf{z}^{(\mathbf{i},\mathbf{k})}$?
- Transformation BN of the convolution output (before activation function).
- This transformation is only valid for 1 SGD iteration, mini-batch.

S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, ICML 2015.

CNN (*convolutional neural network*) / adds-on: Batch normalization

- Let's $\mathbf{v}^{(\mathbf{i})}$ be the output of the convolution for the sample $\mathbf{x}^{(\mathbf{i})}$ belonging to the current mini-batch:

$$\mathbf{v}^{(\mathbf{i})} = \mathbf{x}^{(\mathbf{i})} \star \text{filtre}.$$

CNN (*convolutional neural network*) / adds-on: Batch normalization

- Let's $\mathbf{v}^{(\mathbf{i})}$ be the output of the convolution for the sample $\mathbf{x}^{(\mathbf{i})}$ belonging to the current mini-batch:

$$\mathbf{v}^{(\mathbf{i})} = \mathbf{x}^{(\mathbf{i})} \star \text{filtre}.$$

- We calculate the mean μ_j and the standard deviation σ_j of the j^{th} dimension of $\mathbf{v}^{(\mathbf{i})}$.

CNN (*convolutional neural network*) / adds-on: Batch normalization

- Let's $\mathbf{v^{(i)}}$ be the output of the convolution for the sample $\mathbf{x^{(i)}}$ belonging to the current mini-batch:

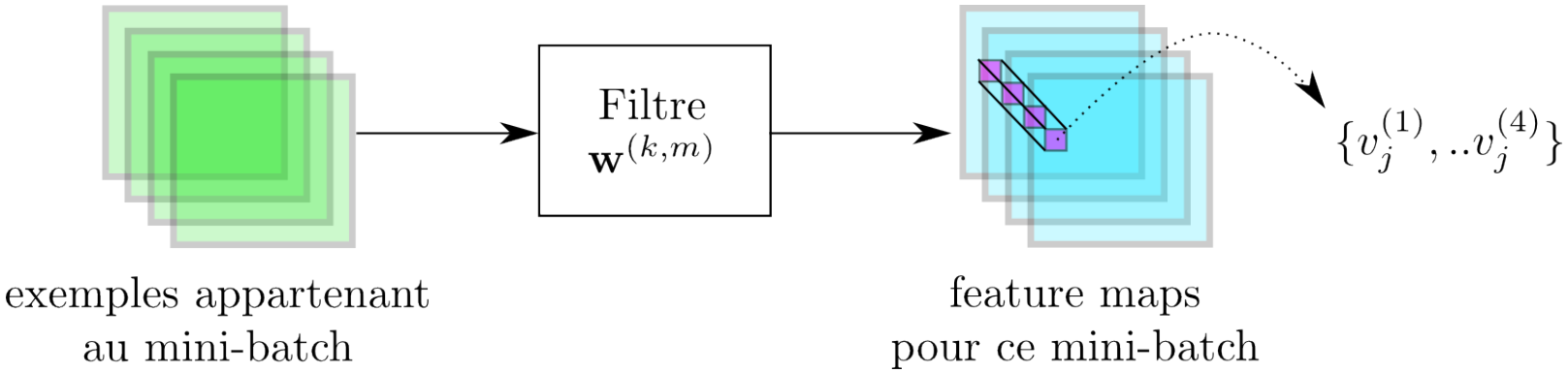
$$\mathbf{v^{(i)}} = \mathbf{x^{(i)}} \star \text{filtre}.$$

- We calculate the mean μ_j and the standard deviation σ_j of the j^{th} dimension of $\mathbf{v^{(i)}}$.
- We replace each $\mathbf{v^{(i)}}$ by:

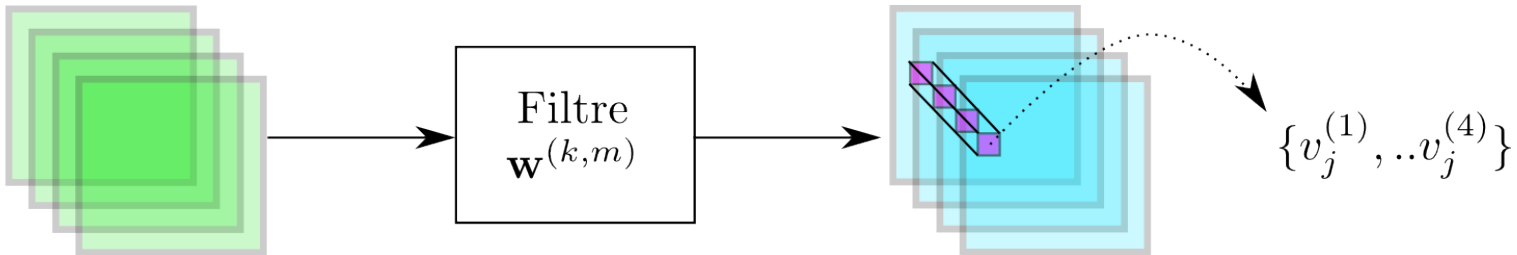
$$\text{BN}\left(\mathbf{v^{(i)}}\right) = \begin{pmatrix} \vdots \\ \gamma_j * \frac{v_j^{(i)} - \mu_j}{\sigma_j^2 + \epsilon} + \beta_j \\ \vdots \end{pmatrix} \tag{4}$$

- γ_j and β_j will be learned, while ϵ is fixed and prevent division by zero.

CNN (*convolutional neural network*) / adds-on: Batch normalization



CNN (*convolutional neural network*) / adds-on: Batch normalization



exemples appartenant au mini-batch

feature maps pour ce mini-batch

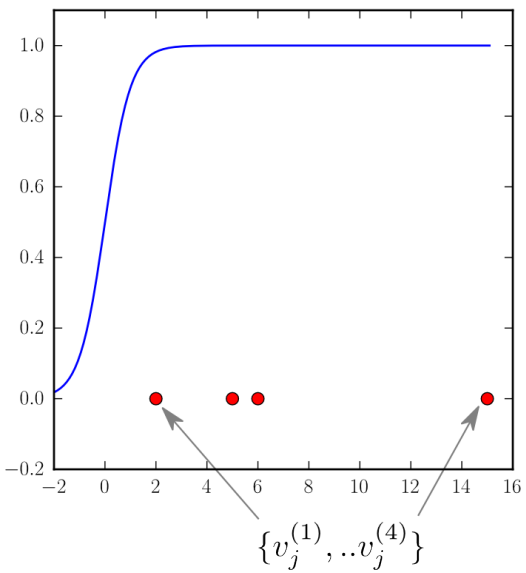
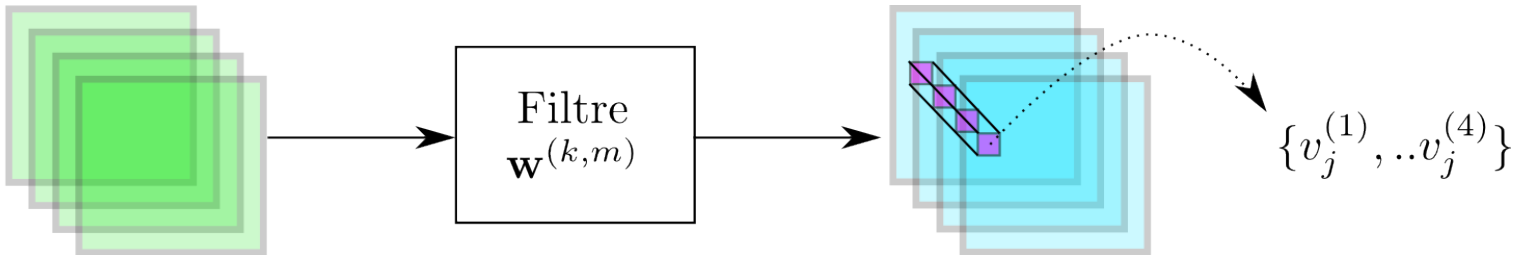


image $\mathbf{x}$

CNN (*convolutional neural network*) / adds-on: Batch normalization



exemples appartenant au mini-batch

feature maps pour ce mini-batch

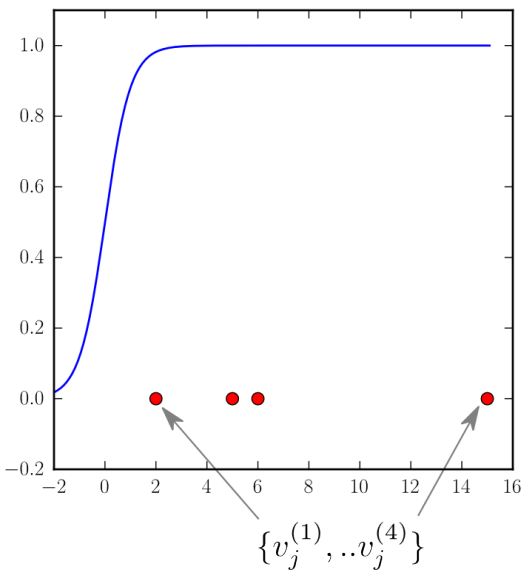
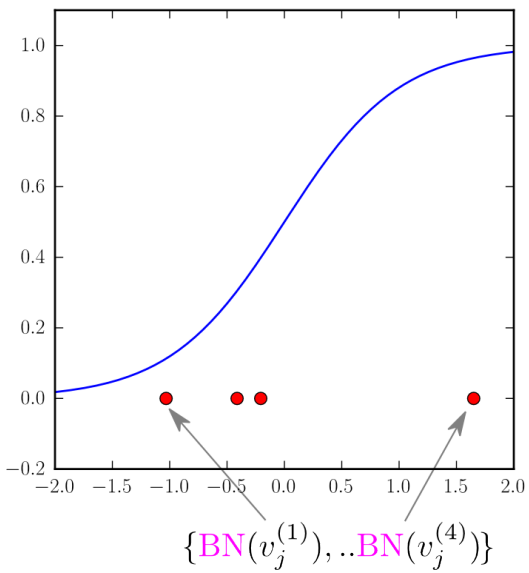


image $\mathbf{x}$



filter $\mathbf{y}$

CNN (*convolutional neural network*) / adds-on: Dropout

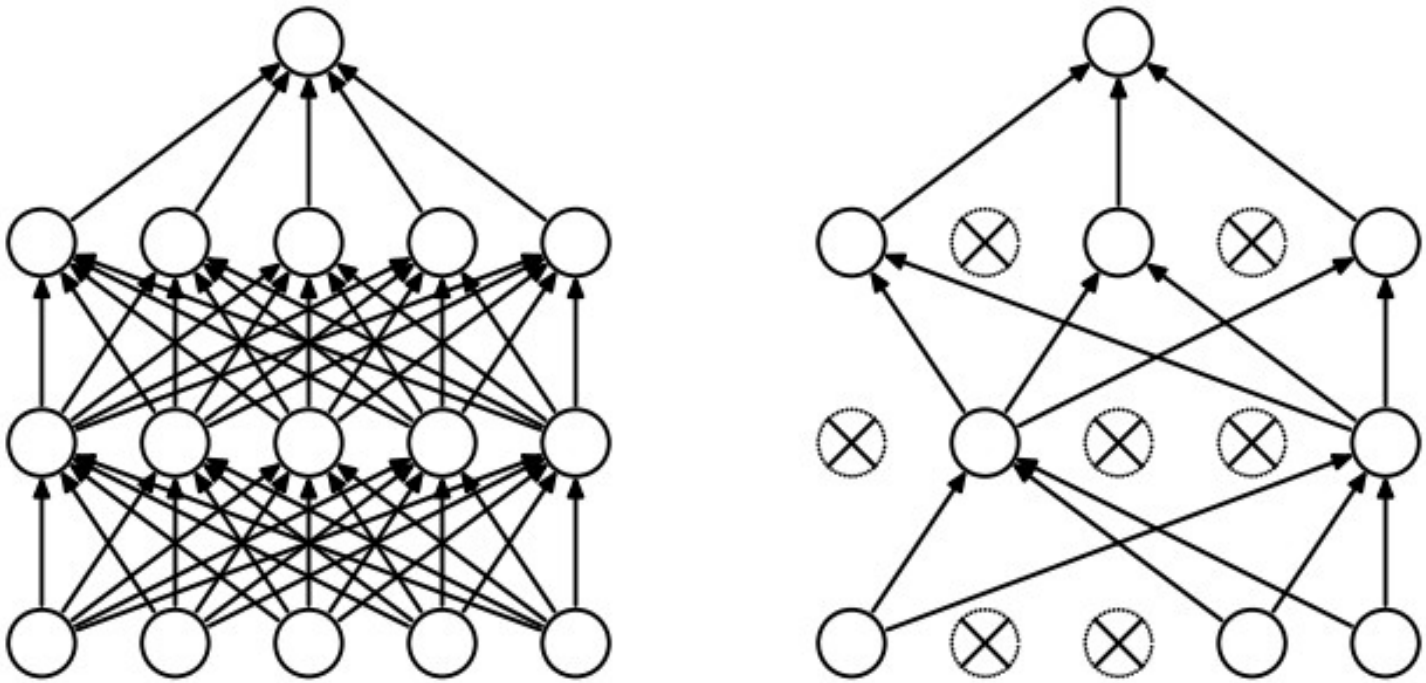
- In order to increase the generalization, we can make things more difficult!

CNN (*convolutional neural network*) / adds-on: Dropout

- In order to increase the generalization, we can make things more difficult!
- The Dropout method "shuts down" randomly few neurons.

CNN (*convolutional neural network*) / adds-on: Dropout

- In order to increase the generalization, we can make things more difficult!
- The Dropout method "shuts down" randomly few neurons.



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

(source: N. Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR 2014.)

CNN (*convolutional neural network*) / adds-on: Dropout

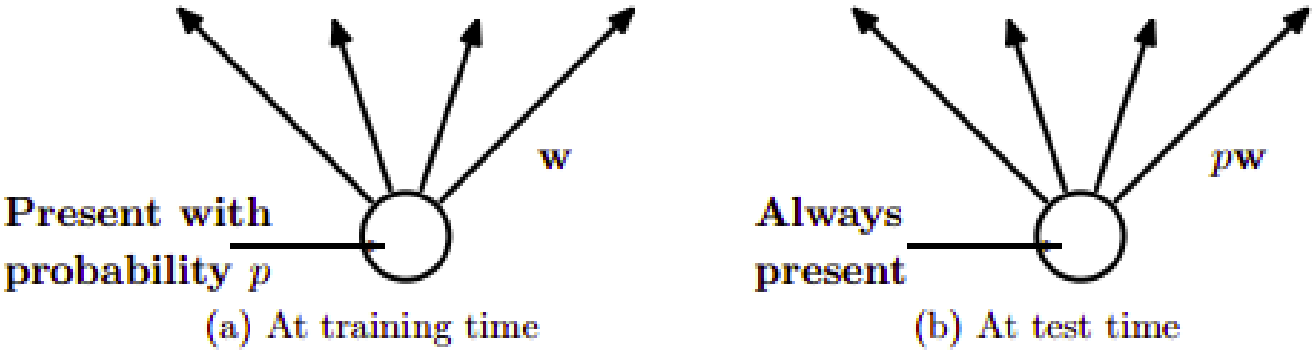
- During the learning phase, each neuron is shut down with a probability p.

CNN (*convolutional neural network*) / adds-on: Dropout

- During the learning phase, each neuron is shut down with a probability p.
- During the test phase, weights connected to the output of neurons are multiply by p.

CNN (*convolutional neural network*) / adds-on: Dropout

- During the learning phase, each neuron is shut down with a probability p.
- During the test phase, weights connected to the output of neurons are multiply by p.



(source: N. Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, JMLR 2014.)

CNN (*convolutional neural network*): Initialization

- CNN filters weights should not be set to zero.
- Heuristic:

$$\mathbf{w}^{(k,m)} \longleftarrow \sqrt{\frac{2}{\dim\left(\mathbf{w}^{(k,m)}\right)}} \times \mathbf{u}, \tag{5}$$

with $\mathbf{u}$ a random vector chosen such as each of its component follow a normal distribution $u_j \sim \mathcal{N}(\prime, \infty)$.

CNN (*convolutional neural network*): Zoo

- LeNet : 90's, 2 series of (conv + max pooling) + MLP,

CNN (*convolutional neural network*): Zoo

- LeNet : 90's, 2 series of (conv + max pooling) + MLP,
- AlexNet : 2012, 5 conv layers (max pooling not systematic) + MLP, 60.000.000 parameters,

CNN (*convolutional neural network*): Zoo

- LeNet : 90's, 2 series of (conv + max pooling) + MLP,
- AlexNet : 2012, 5 conv layers (max pooling not systematic) + MLP, 60.000.000 parameters,
- GoogleLeNet (inception-v4) : 2014, up to 12 conv layers, but some in parallel then regular concatenation,

CNN (*convolutional neural network*): Zoo

- LeNet : 90's, 2 series of (conv + max pooling) + MLP,
- AlexNet : 2012, 5 conv layers (max pooling not systematic) + MLP, 60.000.000 parameters,
- GoogleLeNet (inception-v4) : 2014, up to 12 conv layers, but some in parallel then regular concatenation,
- VGGNet : 2014, up to 19 layers, 140.000.000 parameters,

CNN (*convolutional neural network*): Zoo

- LeNet : 90's, 2 series of (conv + max pooling) + MLP,
- AlexNet : 2012, 5 conv layers (max pooling not systematic) + MLP, 60.000.000 parameters,
- GoogleLeNet (inception-v4) : 2014, up to 12 conv layers, but some in parallel then regular concatenation,
- VGGNet : 2014, up to 19 layers, 140.000.000 parameters,
- ResNet: 2015, some inputs may "jump" layers,

CNN (*convolutional neural network*): Zoo

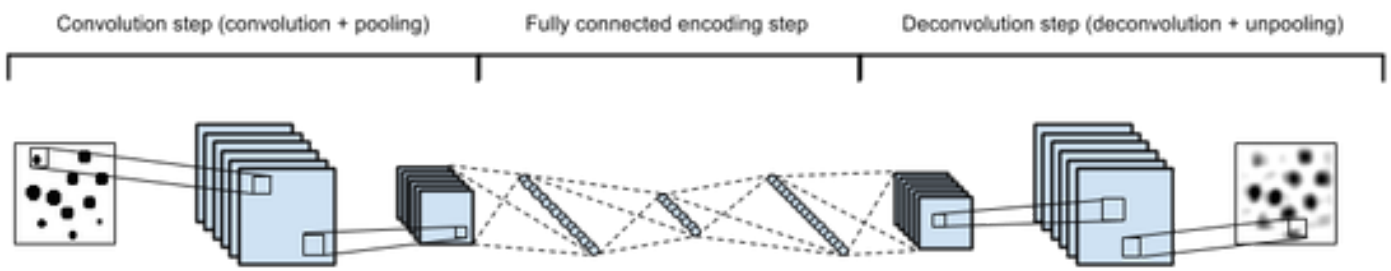
- LeNet : 90's, 2 series of (conv + max pooling) + MLP,
- AlexNet : 2012, 5 conv layers (max pooling not systematic) + MLP, 60.000.000 parameters,
- GoogleLeNet (inception-v4) : 2014, up to 12 conv layers, but some in parallel then regular concatenation,
- VGGNet : 2014, up to 19 layers, 140.000.000 parameters,
- ResNet: 2015, some inputs may "jump" layers,
- DenseNet : 2016, all inputs are fed to all layers.

CNN (*convolutional neural network*): demo

CNN dataset cifar10.

Deep network pre-training:

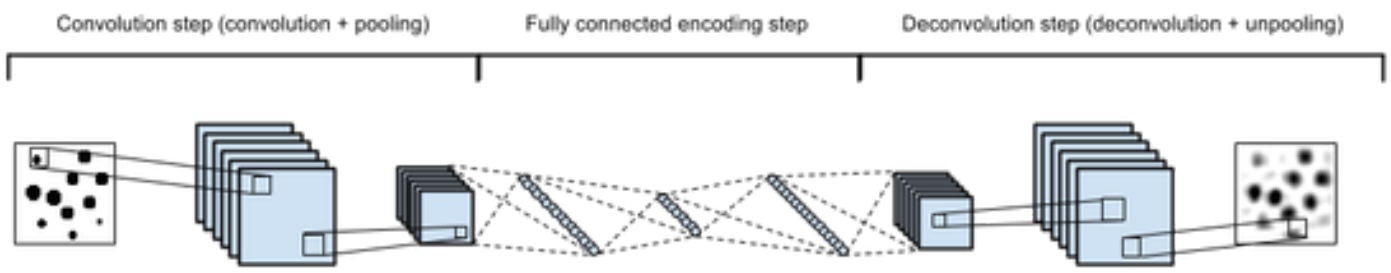
- Auto-encoders :



(source: Blog de Mike Swarbrick Jones.)

Deep network pre-training:

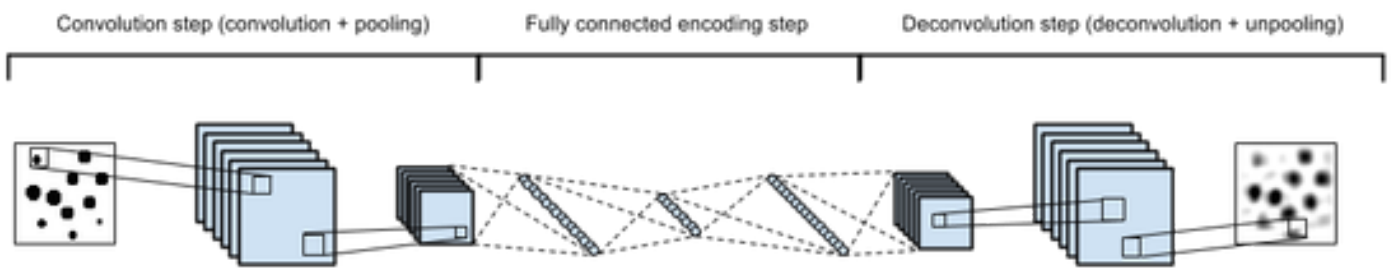
- Auto-encoders :
  - An autoencoder is a neural network that learns to copy its input to its output.



(source: Blog de Mike Swarbrick Jones.)

Deep network pre-training:

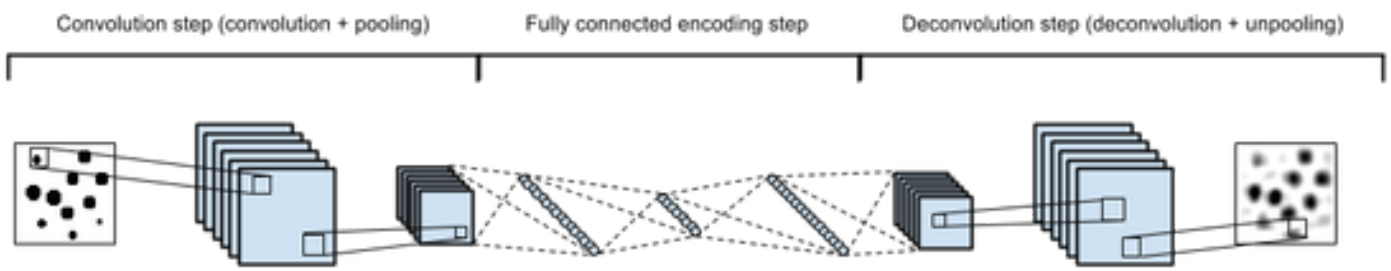
- Auto-encoders :
  - An autoencoder is a neural network that learns to copy its input to its output.
  - A CNN build is build like a "butterfly".



(source: Blog de Mike Swarbrick Jones.)

Deep network pre-training:

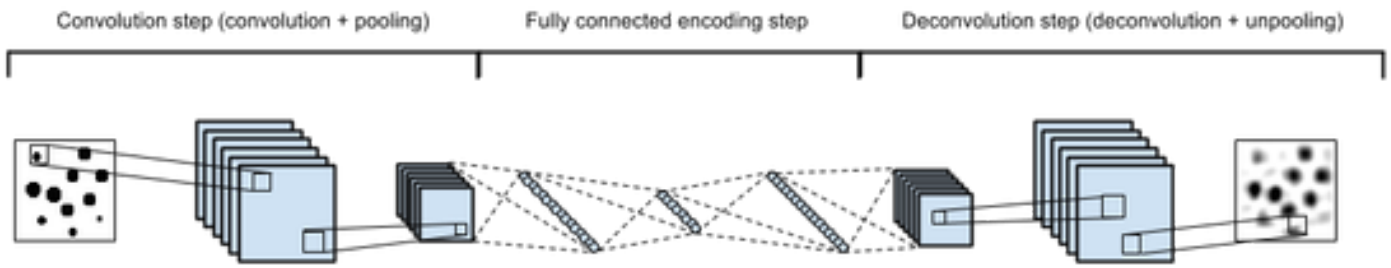
- Auto-encoders :
  - An autoencoder is a neural network that learns to copy its input to its output.
  - A CNN build is build like a "butterfly".
  - The first part transform the input vector into compressed representation.



(source: Blog de Mike Swarbrick Jones.)

Deep network pre-training:

- Auto-encoders :
  - An autoencoder is a neural network that learns to copy its input to its output.
  - A CNN build is build like a "butterfly".
  - The first part transform the input vector into compressed representation.
  - The second part rebuild the input from this representation.



(source: Blog de Mike Swarbrick Jones.)

Deep network pre-training:

- Deep Belief Networks: based on probabilistic model: Restricted Boltzmann machine.

Deep network pre-training:

- Deep Belief Networks: based on probabilistic model: Restricted Boltzmann machine.
  - This model represents the joint distribution of an example $\mathbf{x} = \mathbf{z}^{(0)}$ and output of MLP: $\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(\mathbf{n_c})}$ where n_c is the number of pre-trained layers.

Deep network pre-training:

- Deep Belief Networks: based on probabilistic model: Restricted Boltzmann machine.
  - This model represents the joint distribution of an example $\mathbf{x} = \mathbf{z^{(0)}}$ and output of MLP: $\mathbf{z^{(1)}}, \ldots, \mathbf{z^{(n_c)}}$ where n_c is the number of pre-trained layers.
  - In this model, the conditional law $p\left(z_j^{(i+1)} | \mathbf{z^{(i)}}\right)$ is sgm $\left(\boldsymbol{\theta} \cdot \mathbf{z_+^{(i)}}\right)$.

Deep network pre-training:

- Deep Belief Networks: based on probabilistic model: Restricted Boltzmann machine.
  - This model represents the joint distribution of an example $\mathbf{x} = \mathbf{z^{(0)}}$ and output of MLP: $\mathbf{z^{(1)}}, \ldots, \mathbf{z^{(n_c)}}$ where n_c is the number of pre-trained layers.
  - In this model, the conditional law $p\left(z_j^{(i+1)} | \mathbf{z^{(i)}}\right)$ is sgm $\left(\boldsymbol{\theta} \cdot \mathbf{z_+^{(i)}}\right)$.
  - The *contrastive divergence* algorithm force the model to rebuild the input $\mathbf{x}$ in unsupervised mode.

Deep network pre-training:

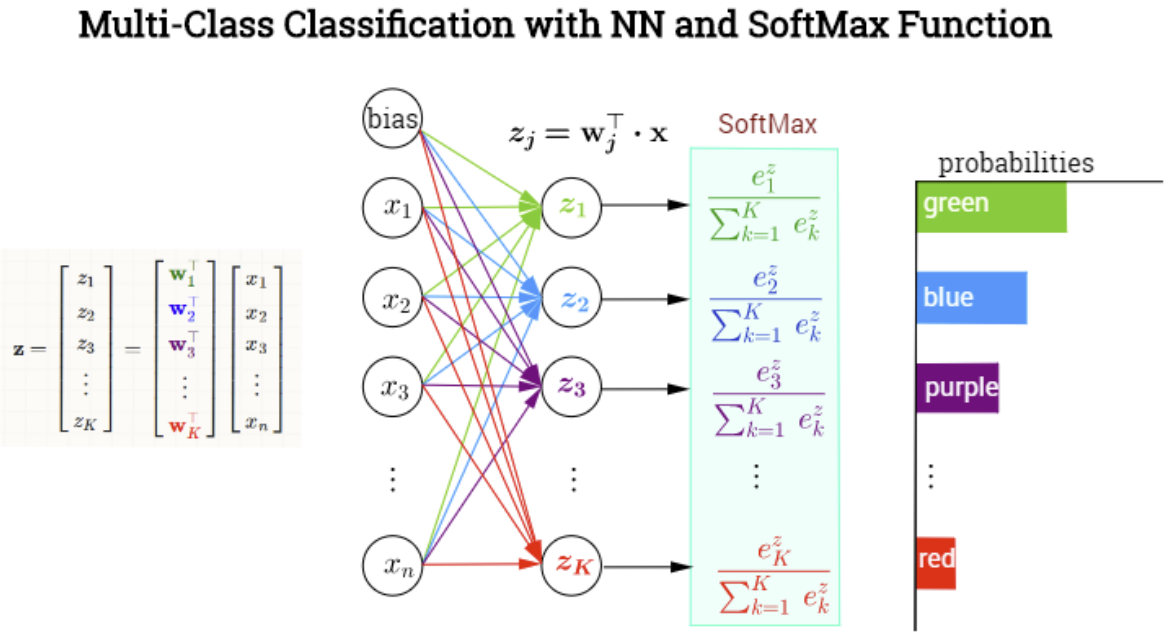
- Deep Belief Networks: based on probabilistic model: Restricted Boltzmann machine.
  - This model represents the joint distribution of an example $\mathbf{x} = \mathbf{z^{(0)}}$ and output of MLP: $\mathbf{z^{(1)}}, \ldots, \mathbf{z^{(n_c)}}$ where n_c is the number of pre-trained layers.
  - In this model, the conditional law $p\left(z_j^{(i+1)} | \mathbf{z^{(i)}}\right)$ is sgm $\left(\boldsymbol{\theta} \cdot \mathbf{z_+^{(i)}}\right)$.
  - The *contrastive divergence* algorithm force the model to rebuild the input $\mathbf{x}$ in unsupervised mode.
  - After convergence, an MLP is trained in supervised way.

Deep network pre-training:

- Huge gain:
  Semi-supervised learning! (pre-training with unlabeled data, fine-tuning with labeled data).

Outline

Neural network: classification



Softmax:

$$\sigma(j) = \frac{e^{\mathbf{w}_j^T \mathbf{x}}}{\sum_{k=1}^{K} e^{\mathbf{w}_k^T \mathbf{x}}} = \frac{\mathbf{z}_j}{\sum_{k=1}^{K} e^{\mathbf{z}_k}} \tag{6}$$

This will result in a normalization of the output adding up to 1, interpretable as a probability mass function.

Neural network: classification Softmax:

- softmax is optimal for maximum-likelihood estimation of the model parameters.
- all output values in $[0, 1]$ and sum up to 1 make it suitable for a probabilistic interpretation.
- Softmax normalization is a way of reducing the influence of extreme values or outliers in the data without removing data points from the set.

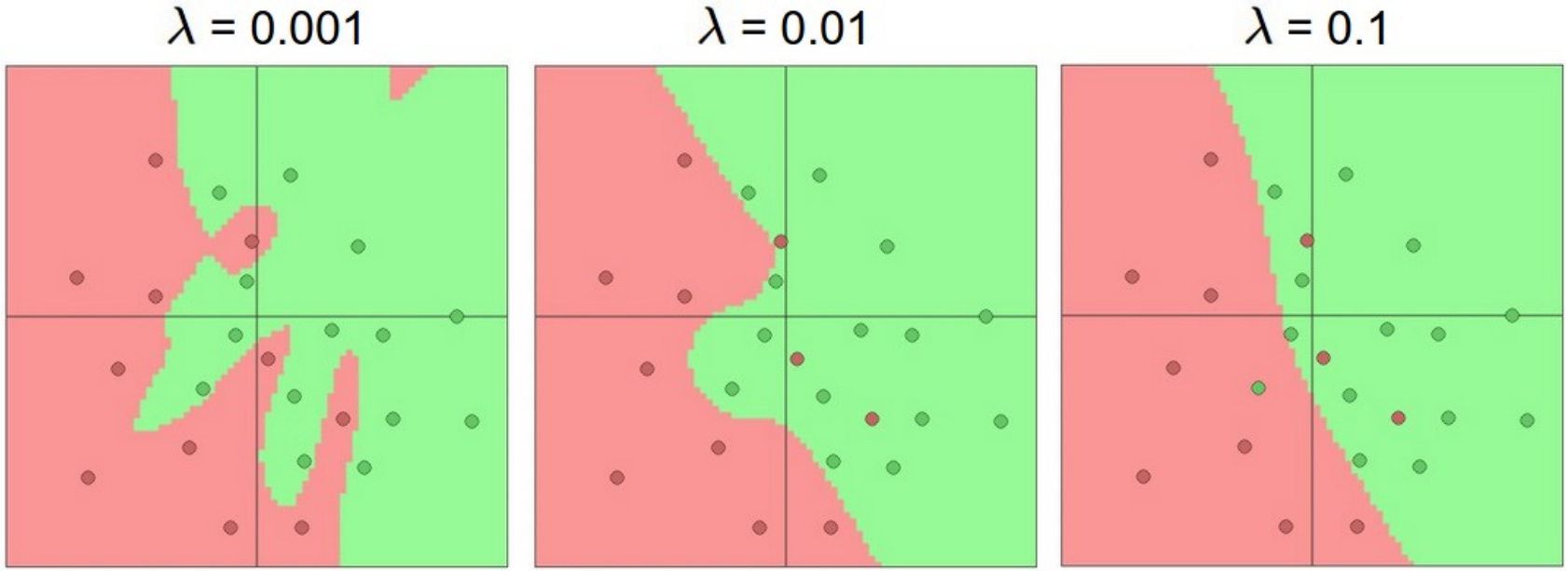
Neural network: classification Loss function: cross entropy

$$E = -\frac{1}{k}\sum_{k=1}^{K}[y_k log(\sigma(z_k)) + (1 - y_k) log(1 - \sigma(z_k))]$$

Cross entropy derivative:

$$\frac{dE}{dz_k} = -\frac{y_k}{z_k} + \frac{1 - y_k}{1 - z_k}$$

Deep network: prevent overfitting, reiterate with the regularization strength



(https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html)

Librairies:

- Caffe
- Tensorflow (Google)
- Microsoft Cognitive Toolkit (CNTK) (Microsoft)
- Pytorch
- Keras: high level backend Theano, Tensorflow, CNTK

Outline

Important points:

- Neural networks allow us to process data whose separating boundary is obviously not linear.
- Thanks to the backpropagation, the learning phase of these model is not exorbitant.
- Modern neural networks produces results of remarkable quality, especially in computer vision.