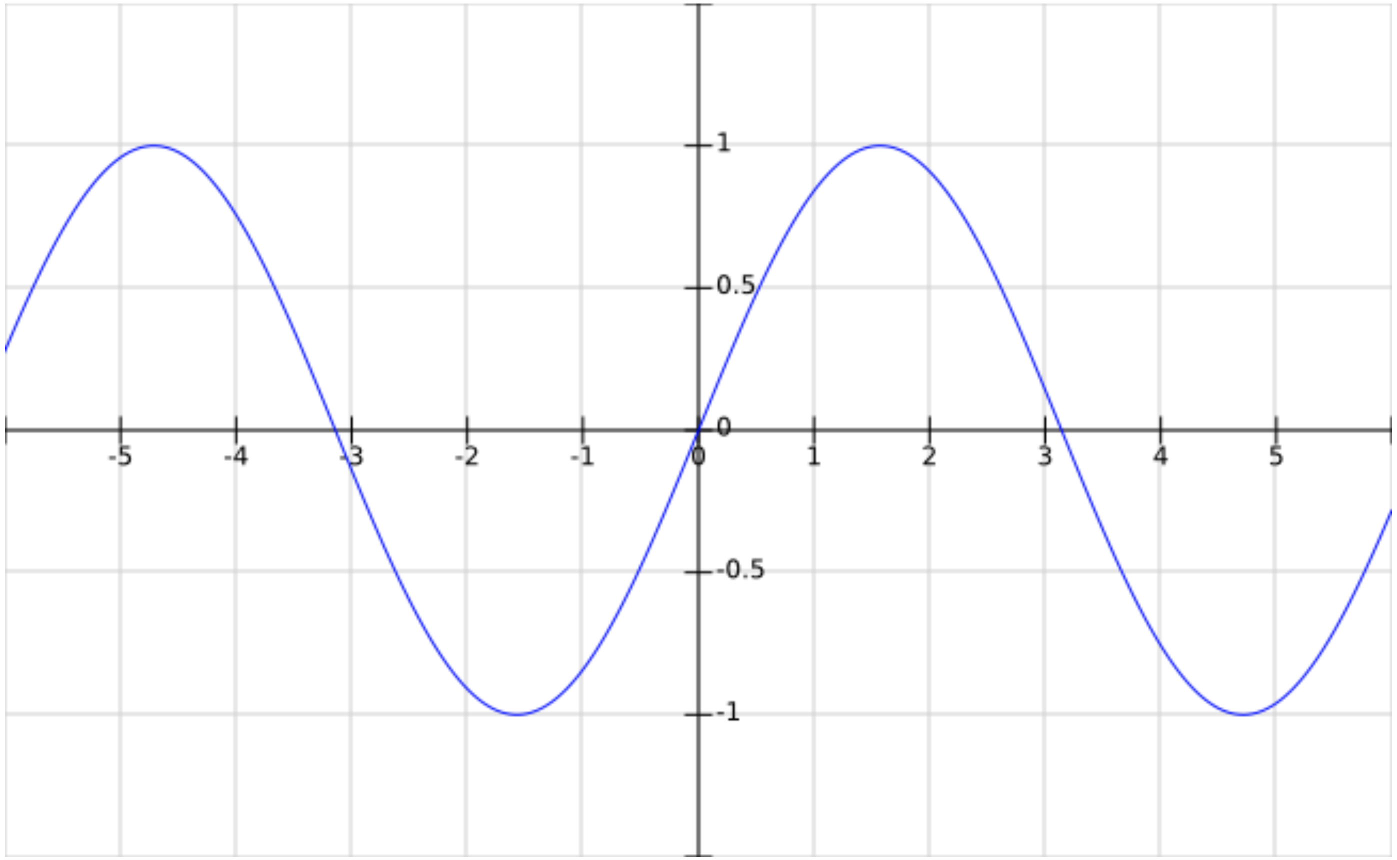
Neural networks & Deep learning

Pierre Chainais
(thanks to Florian Stub)

centralelille

Adaptative Basis - Regression

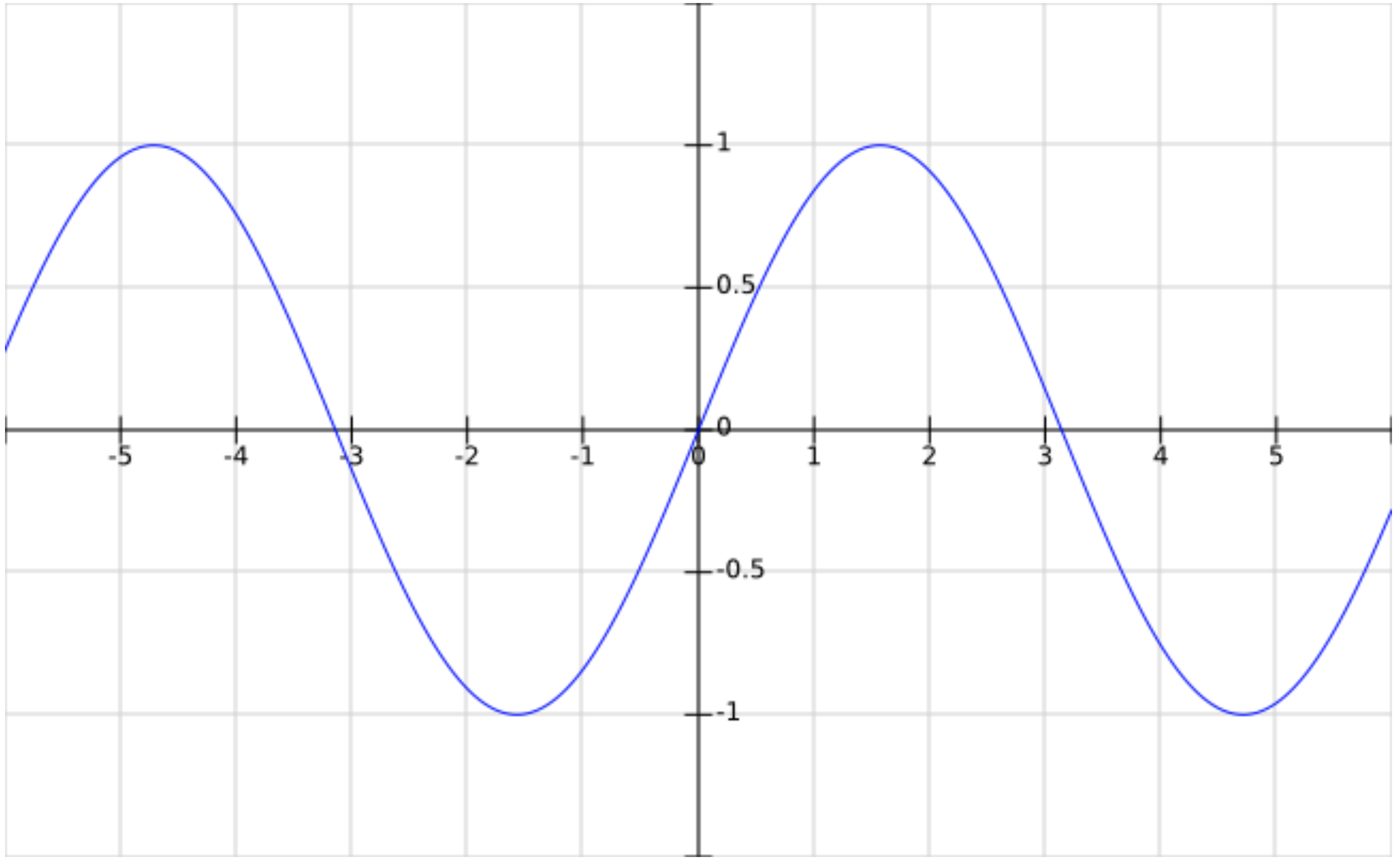


Goal :
Approximate the sinus

$$\mathbf{t} = sin(\mathbf{x})$$

- **t** is the target vector
- **x** *is the input vector*

Adaptative Basis - Regression

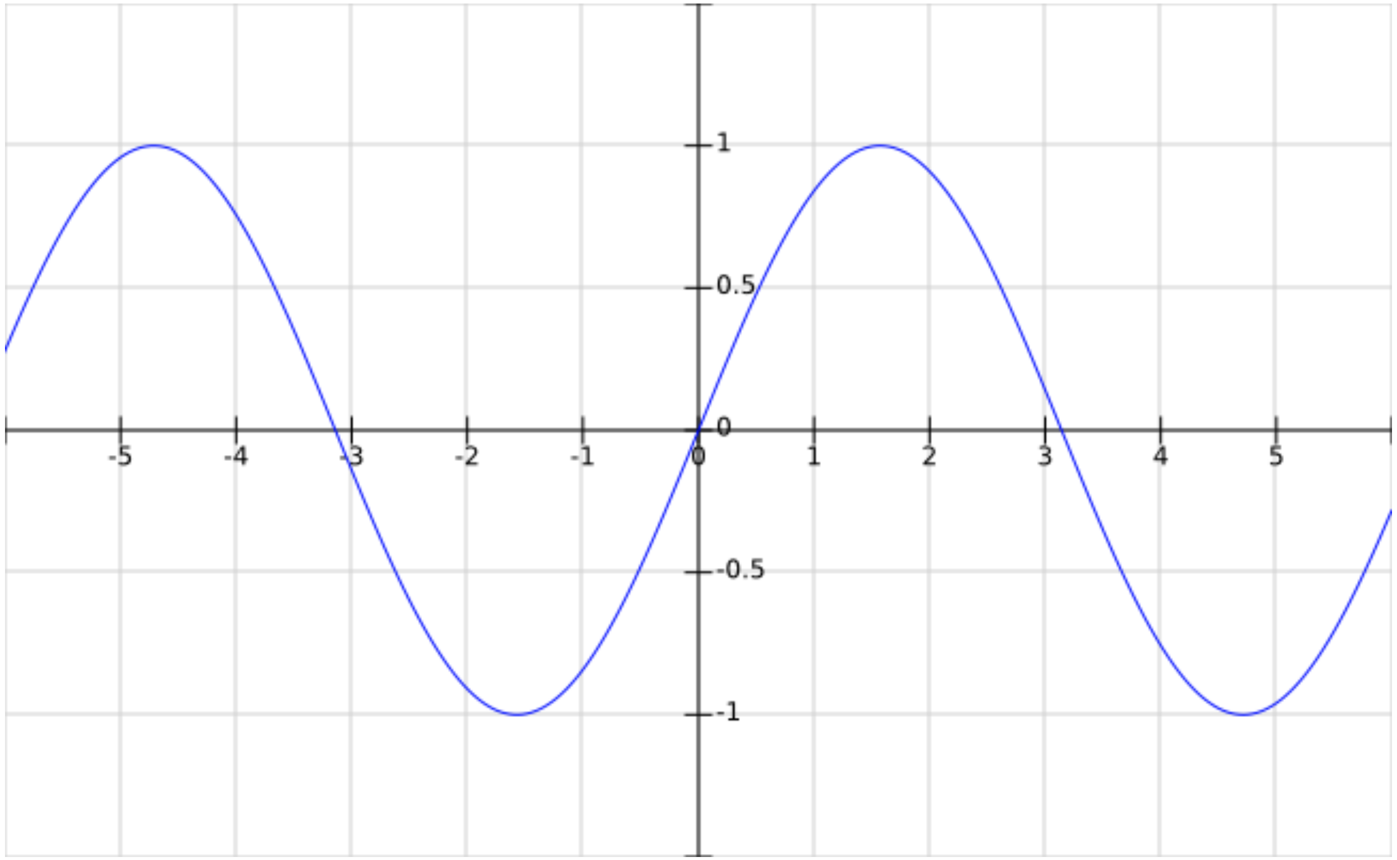


Goal :
Approximate the sinus

$$\mathbf{y} = \mathbf{w}^{\mathbf{t}}\mathbf{x} + b$$

- **y** is the predicted vector
- **w** *is the weight vector*
- *b is the bias*

Adaptative Basis - Regression

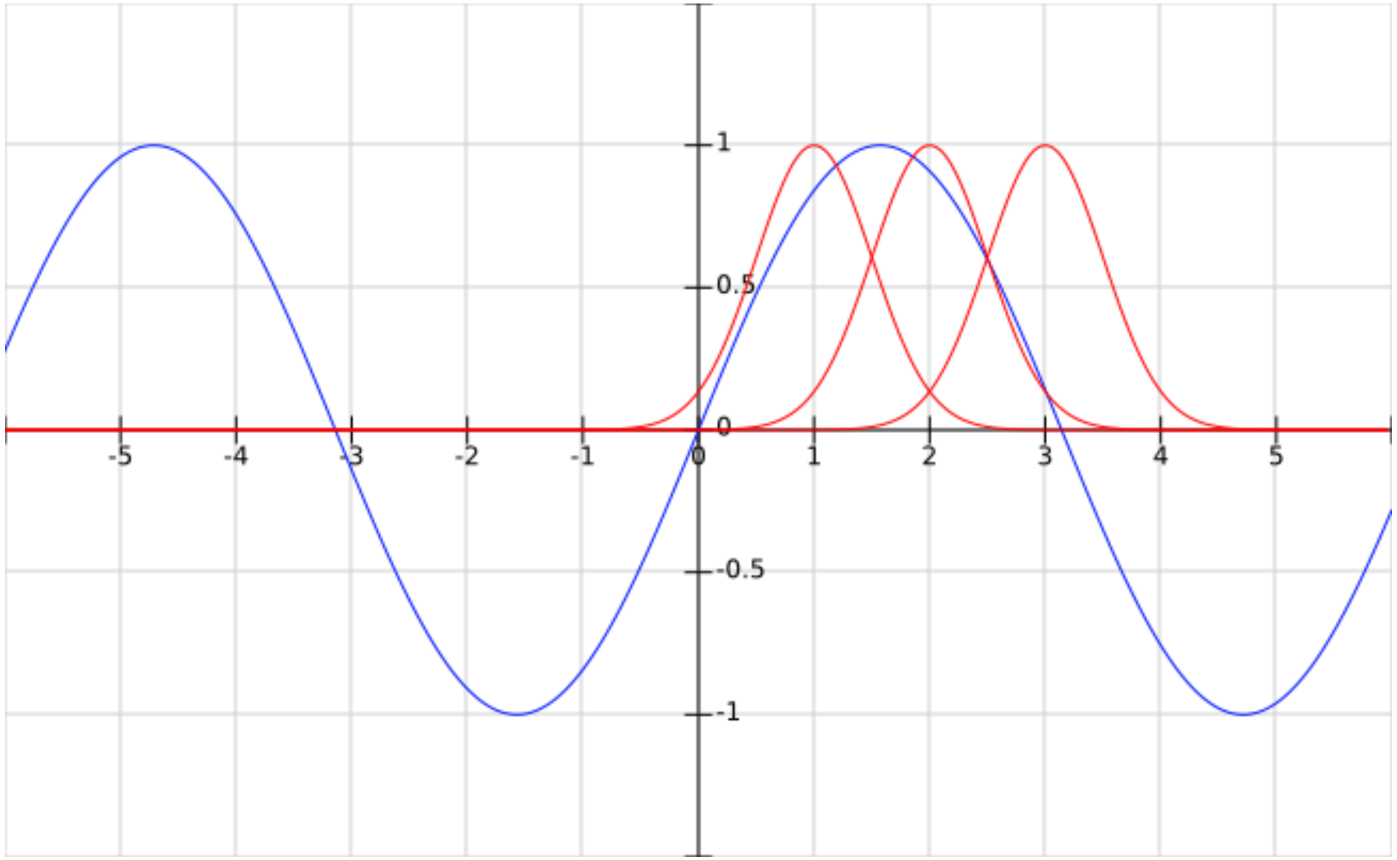


Goal :
Approximate the sinus

$$\mathbf{y} = \mathbf{w}^{\mathbf{t}}\boldsymbol{\Phi}(\mathbf{x}) + b$$

- $\boldsymbol{\Phi}$ are the basis functions

Adaptative Basis - Regression

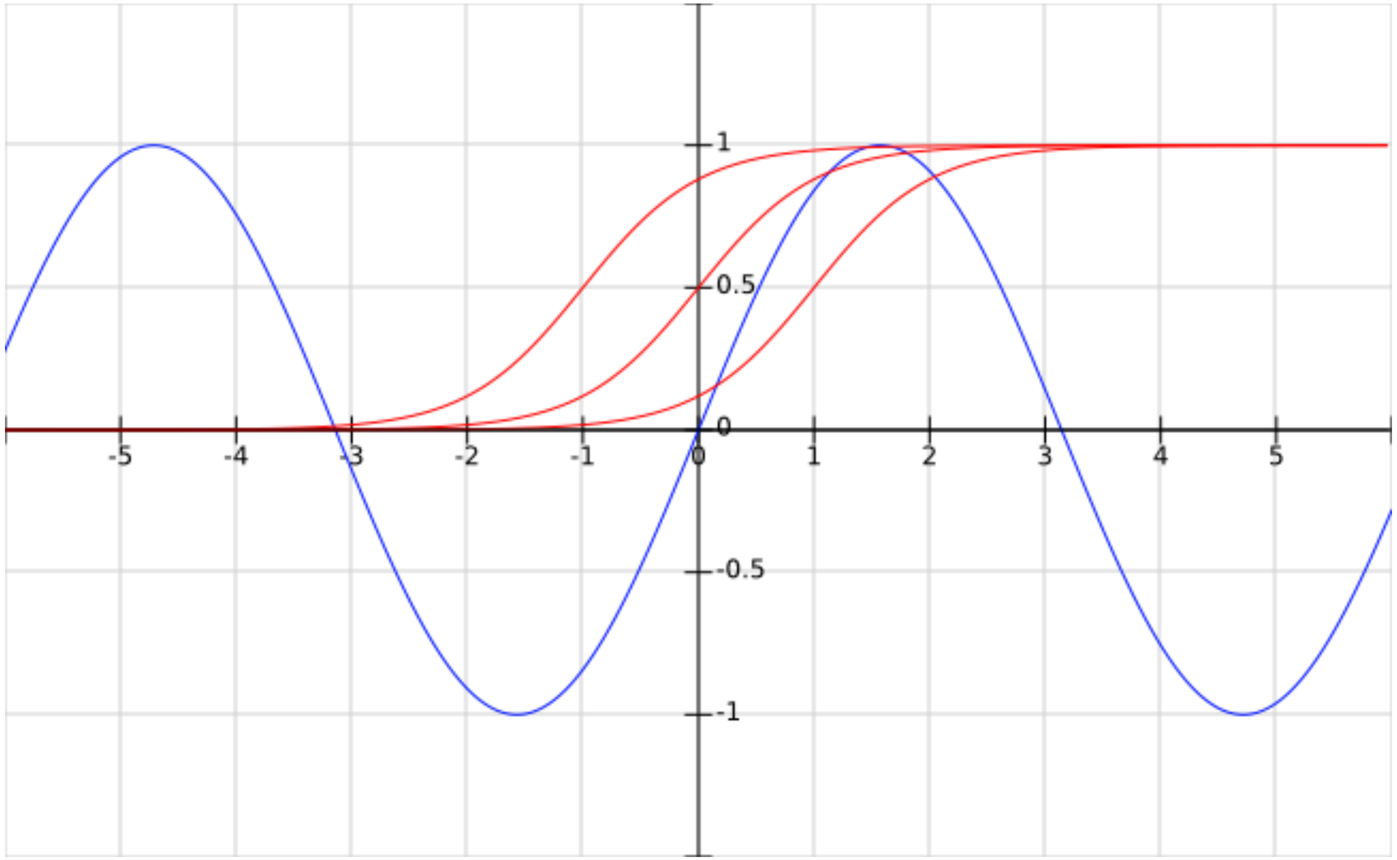


Goal :
Approximate the sinus

$$\mathbf{y} = \mathbf{w}^{\mathbf{t}}\boldsymbol{\Phi}(\mathbf{x}) + b$$

If $\boldsymbol{\Phi}$ are Gaussians

Adaptative Basis - Regression

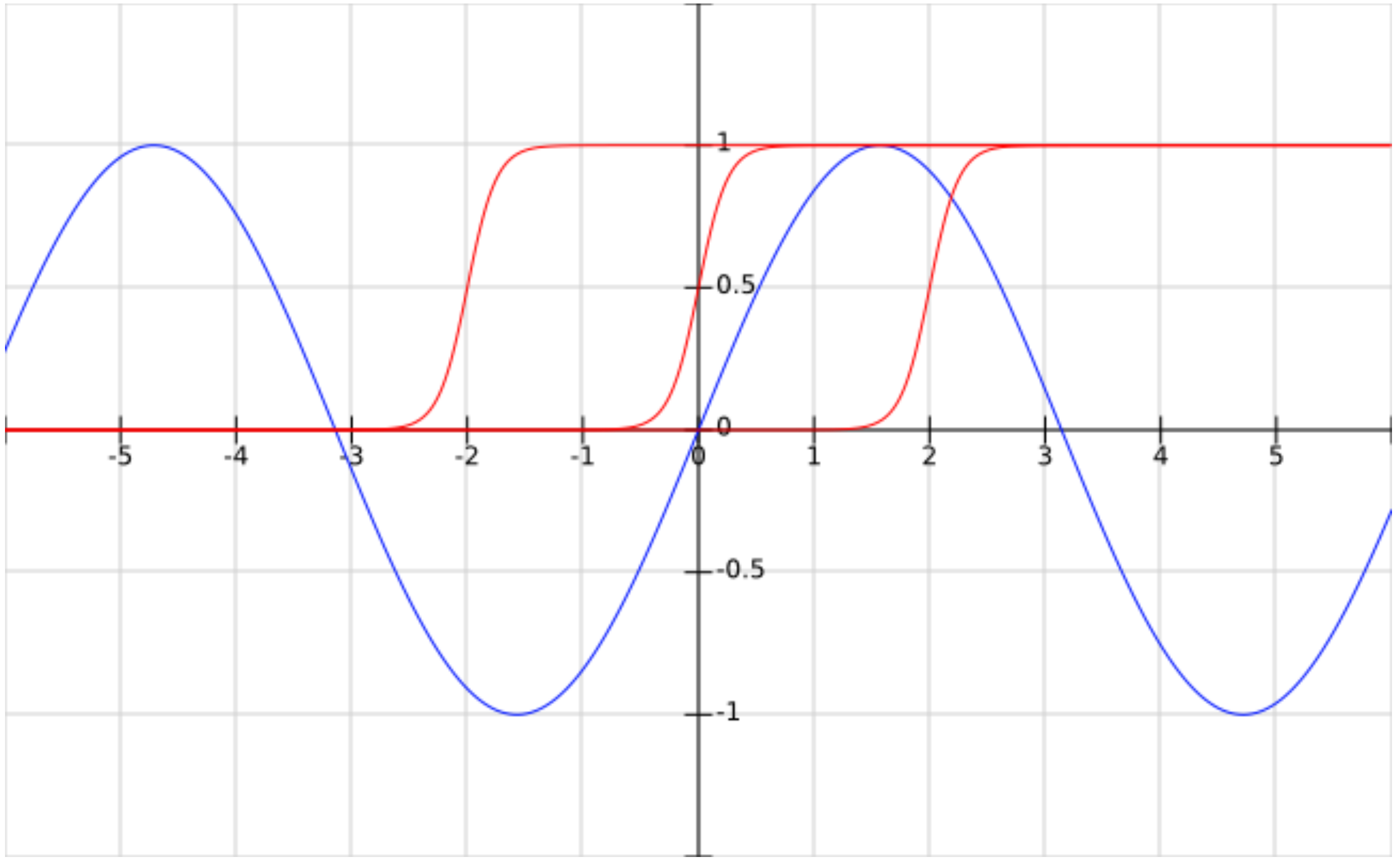


Goal :
Approximate the sinus

$$\mathbf{y} = \mathbf{w}^{\mathbf{t}}\mathbf{\Phi}(\mathbf{x}) + b$$

If $\mathbf{\Phi}$ are Sigmoids

Adaptative Basis - Regression

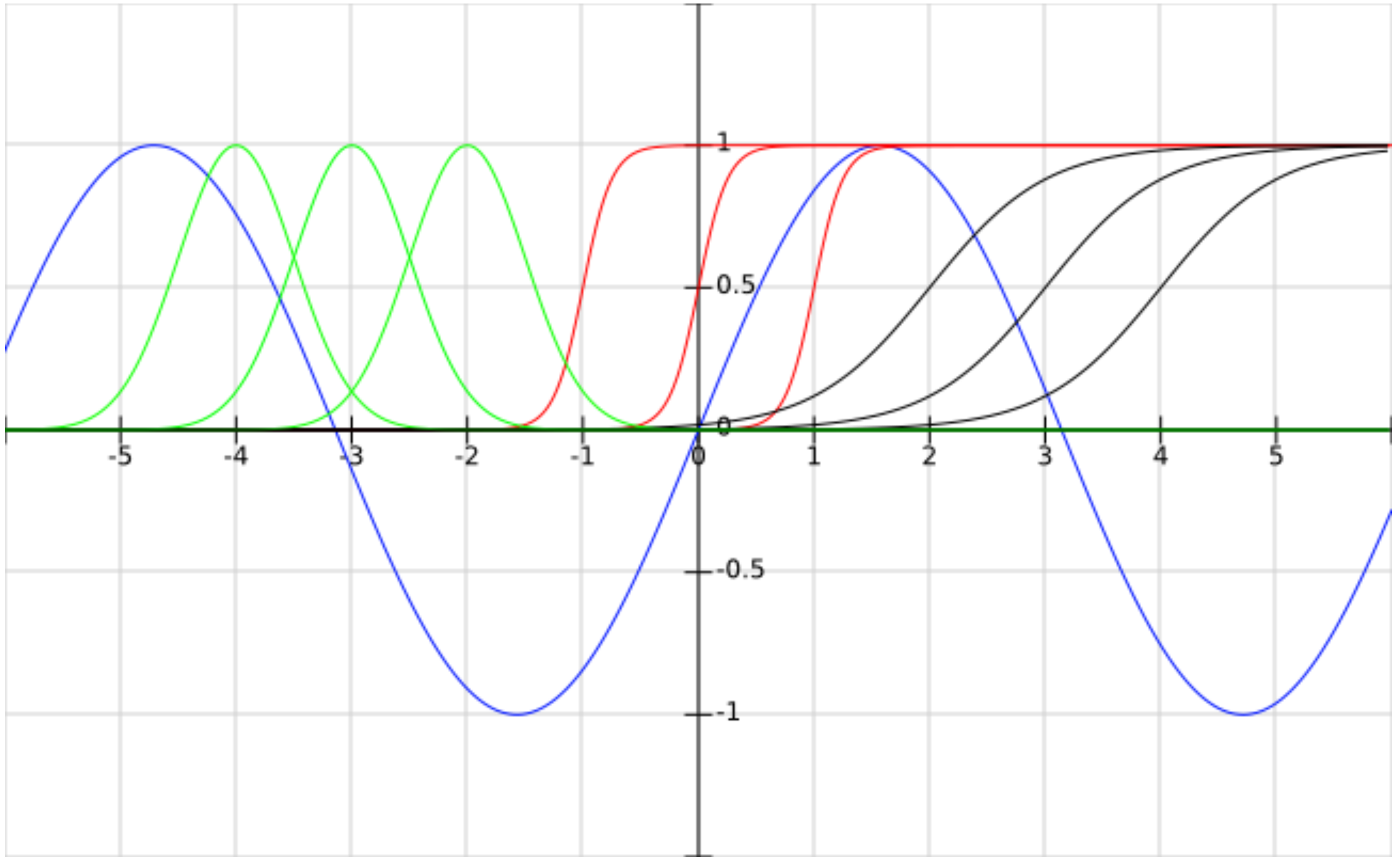


Goal :
Approximate the sinus

$$\mathbf{y} = \mathbf{w}^{\mathbf{t}}\boldsymbol{\Phi}(\mathbf{x}) + b$$

If $\boldsymbol{\Phi}$ are Sigmoids

Adaptative Basis - Regression

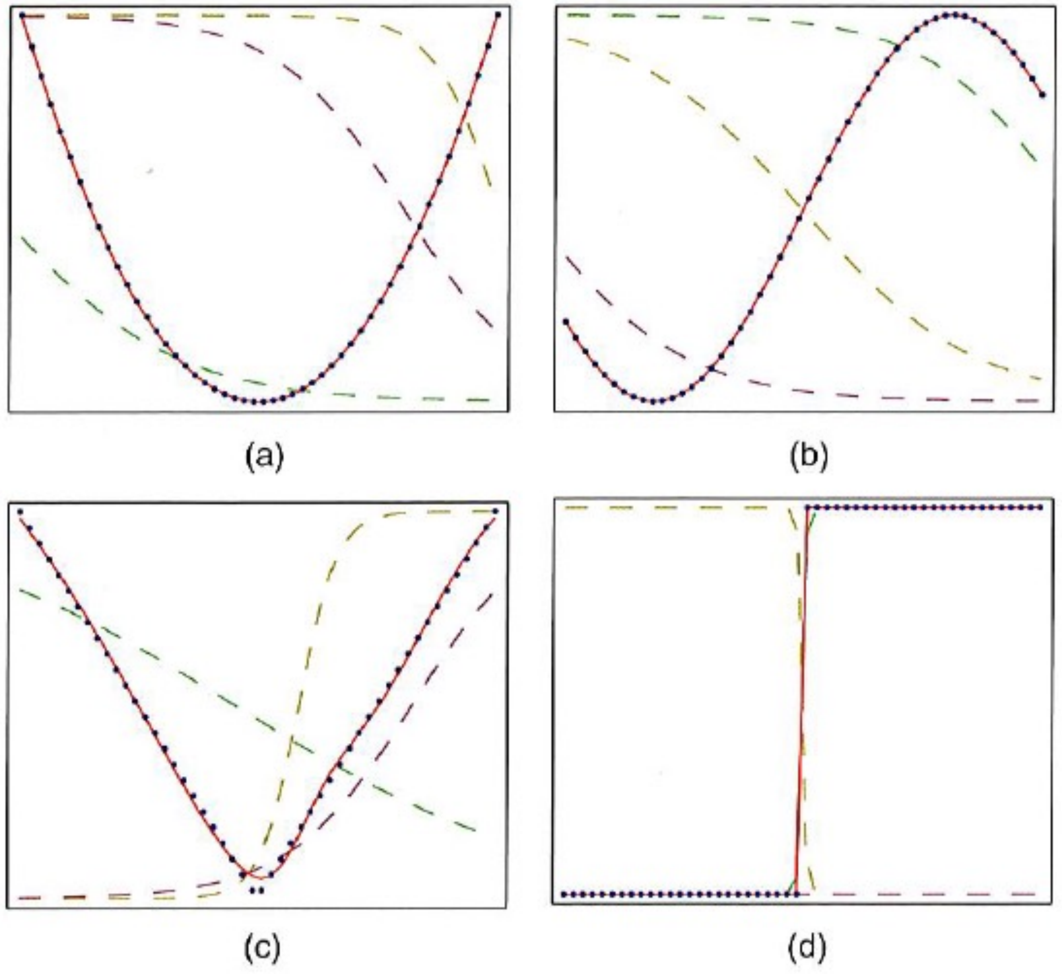


What are the best basis functions ?

- Build $\boldsymbol{\Phi}$ on key samples
  → **SVN**

- Learn $\boldsymbol{\Phi}$
  → **Neural Networks**

Adaptative Basis - Regression



Capacity of multilayer neural networks to learn basis function in order compute basis functions:

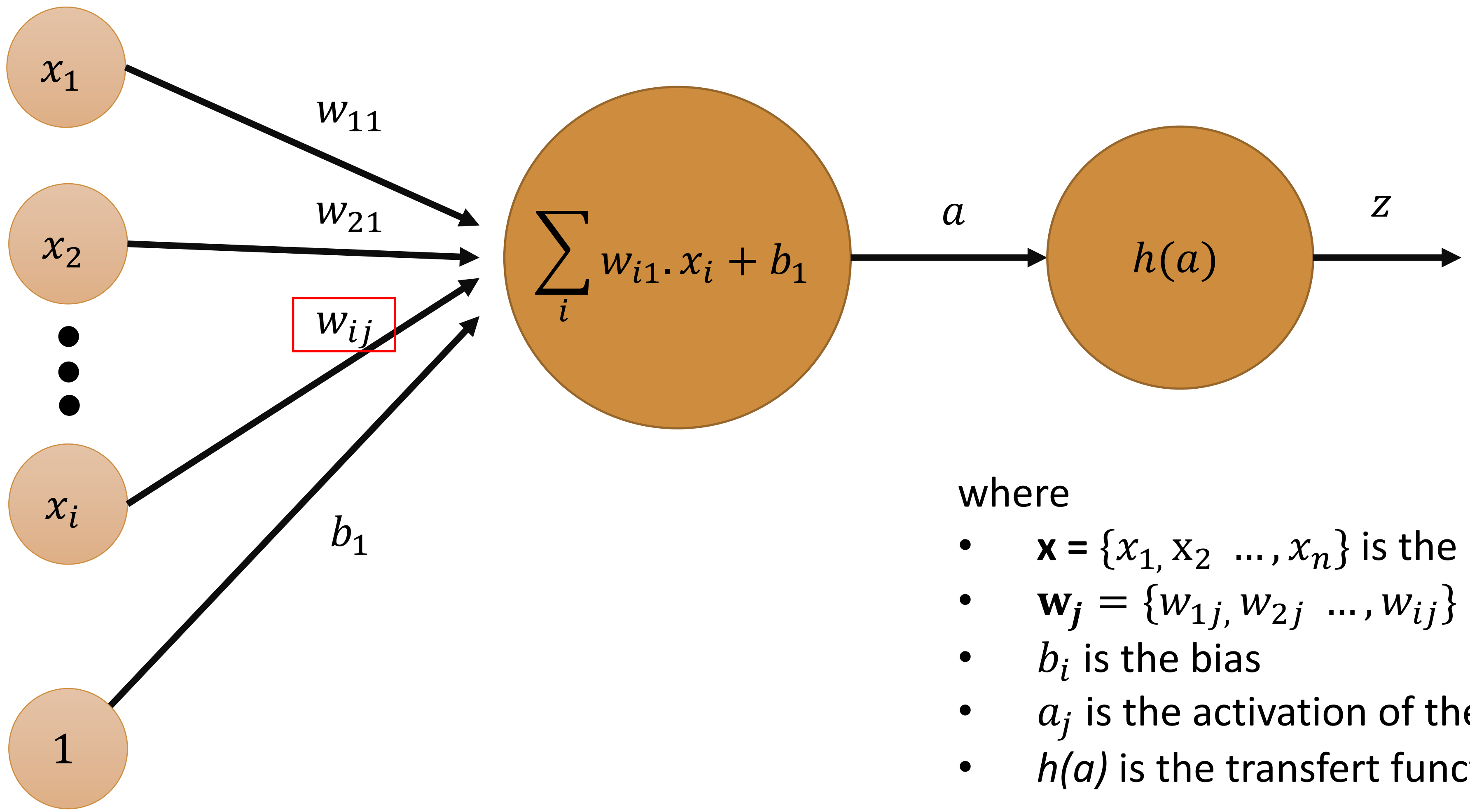
- Red curve is the target function
- Blue points are sampled input (50 points)
- Dashed curves are basis functions (output of hidden units)

Neural network with 3 hidden units with *tanh* activation and linear output units.
Bishop, p. 231.

NB : $if\ h(x) = \tanh(x)\ \ then\ h'(x)$ = 1 + $h(x)^2$

<u>Source</u> : Pattern recognition and Machine Learning, *Bishop* 2006

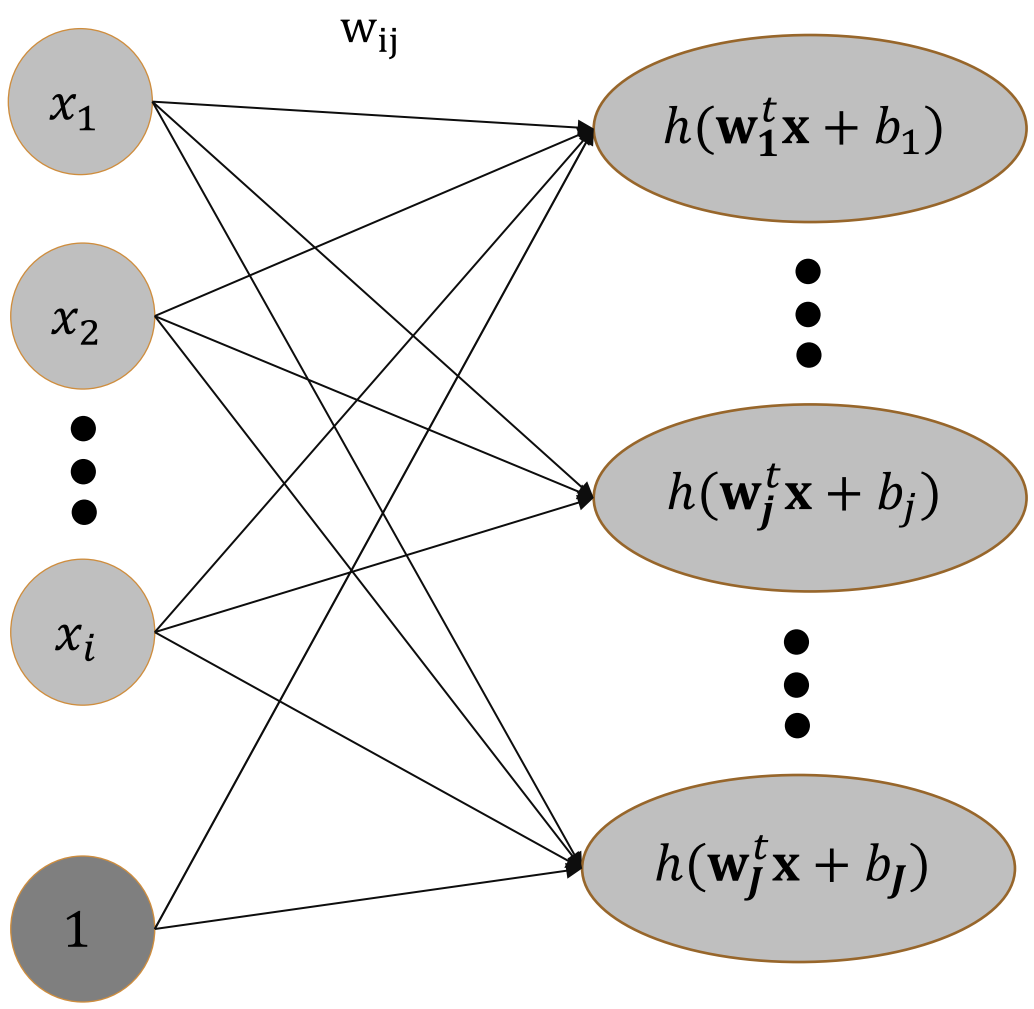
Feed-Forward Neural Networks functions



where

- **x** = $\{x_1, x_2 \ \ldots, x_n\}$ is the input vector
- $\mathbf{w_j} = \{w_{1j}, w_{2j} \ \ldots, w_{ij}\}$ is the weight vector
- b_i is the bias
- a_j is the activation of the neuron
- *h(a)* is the transfert function

Feed-Forward Neural Networks functions

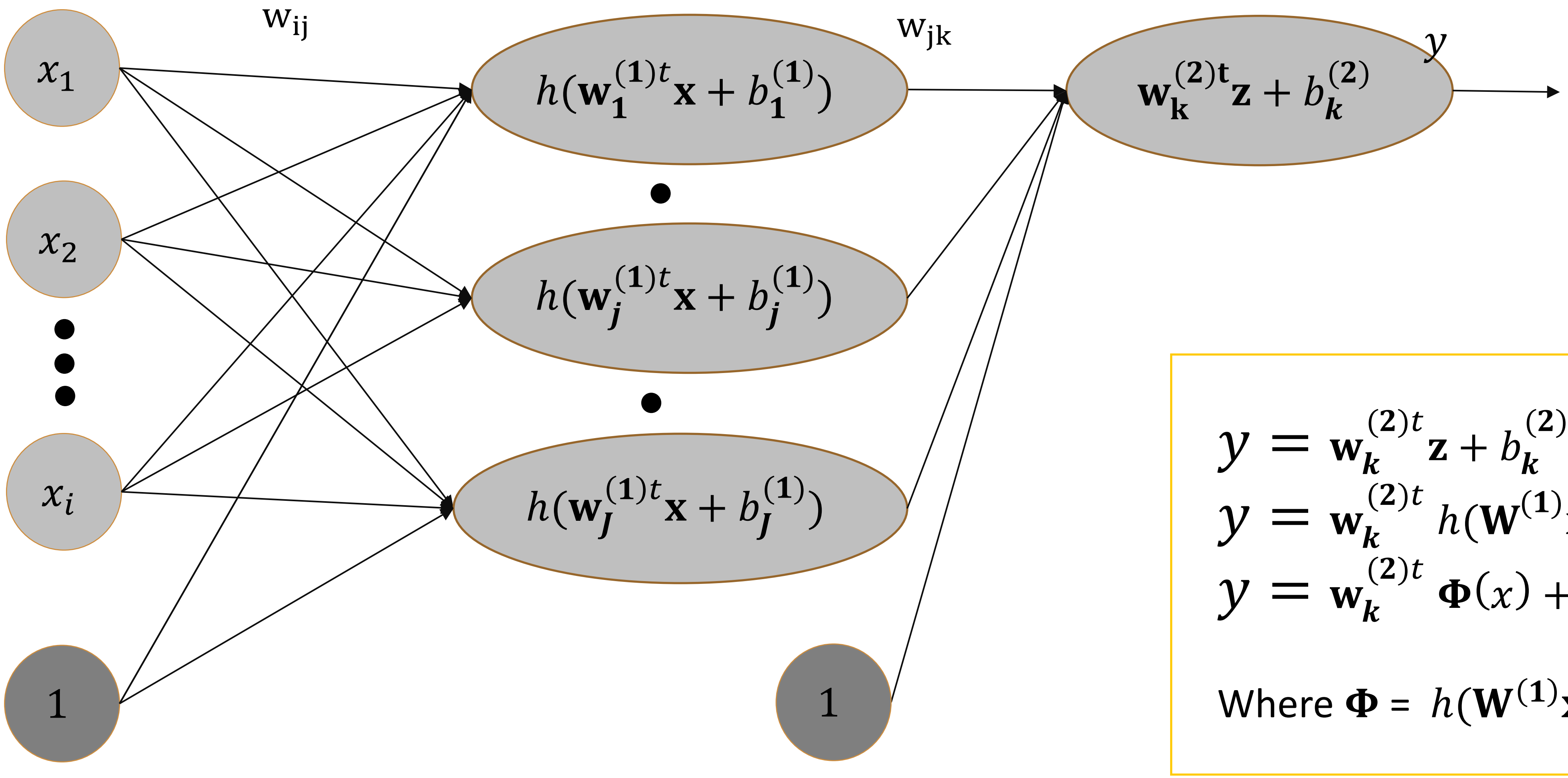


$$\mathbf{z} = h(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where

- $\mathbf{x} = \{x_1, x_2 \ \dots, x_i\}$ is the input vector
- $\mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2 \ \dots, \mathbf{w}_J\}$ is the weight matrix
- $\mathbf{b} = \{b_1, b_2 \ \dots, b_J\}$ is the bias vector
- $\mathbf{z} = \{z_1, z_2 \ \dots, z_J\}$ is the output vector

Feed-Forward Neural Networks functions



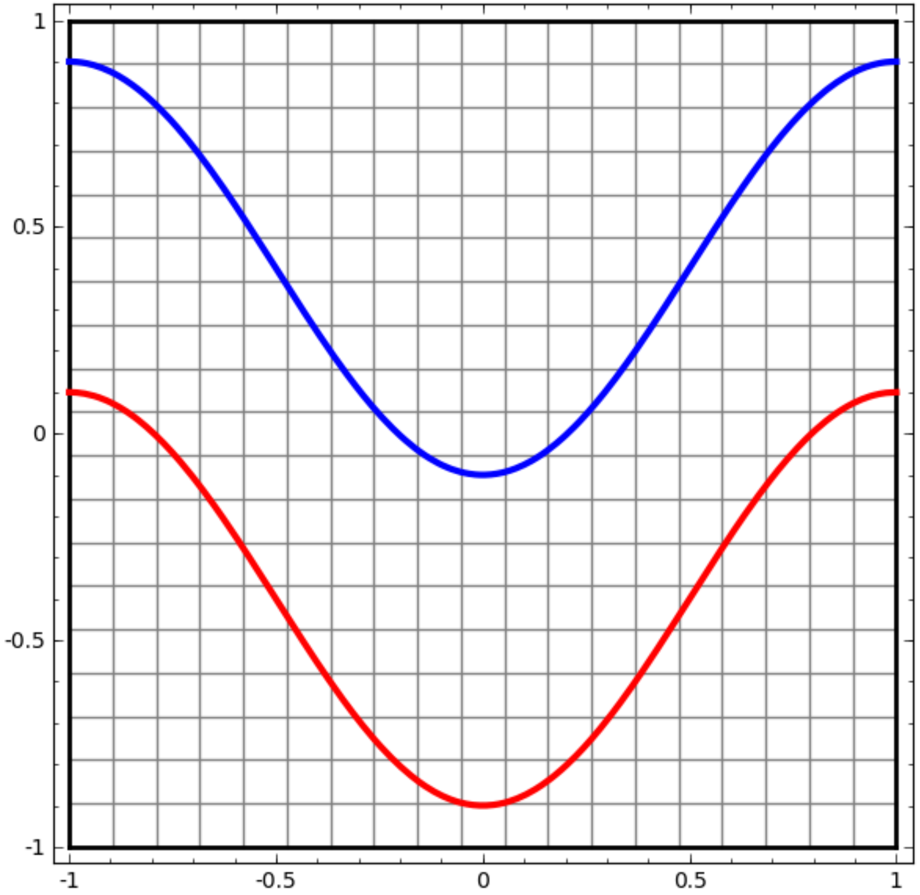
$$y = \mathbf{w}_k^{(2)t}\mathbf{z} + b_k^{(2)}$$
$$y = \mathbf{w}_k^{(2)t}\ h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}) + b_k^{(2)}$$
$$y = \mathbf{w}_k^{(2)t}\ \mathbf{\Phi}(x) + b_k^{(2)}$$

Where $\mathbf{\Phi}$ = $h(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$

Feed-Forward Neural Networks functions

$$y_k(\mathbf{x}) = \sum_{j}^{J} w_{jk}^{(2)}\, h\underbrace{\left(\sum_{i}^{I} w_{ij}^{(1)} x_i + b_j^{(1)}\right)}_{\Phi_k} + b_k^{(2)}$$

Adaptative Basis - Classification

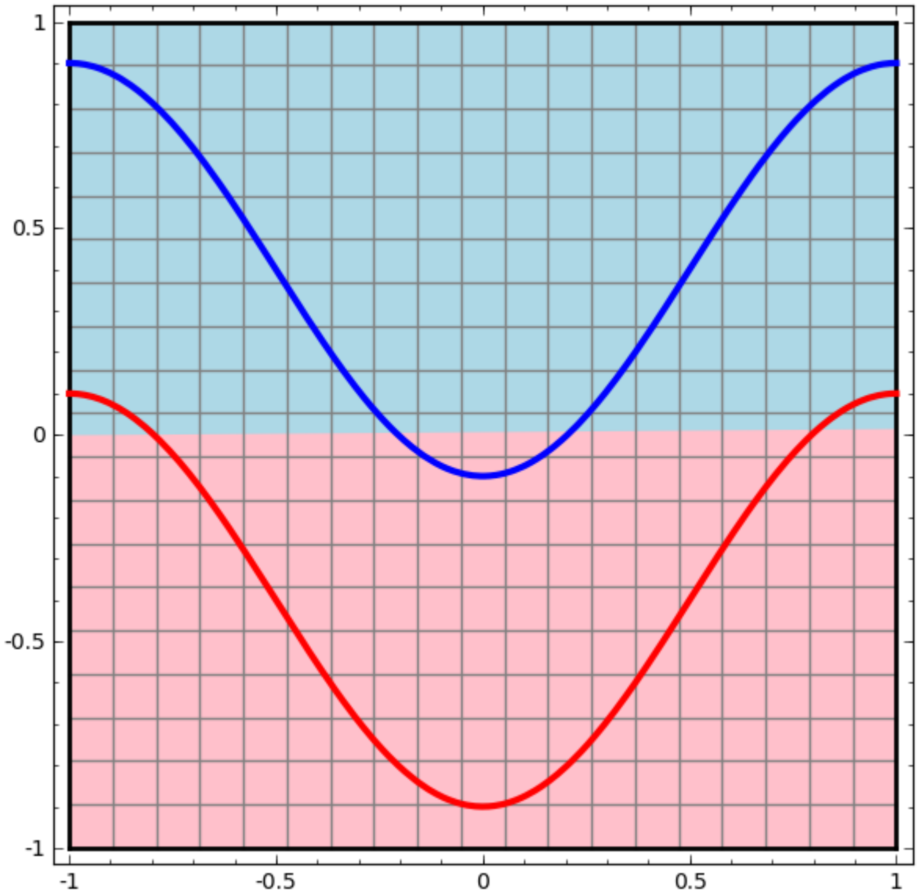


A very simple dataset, two curves on a plane.

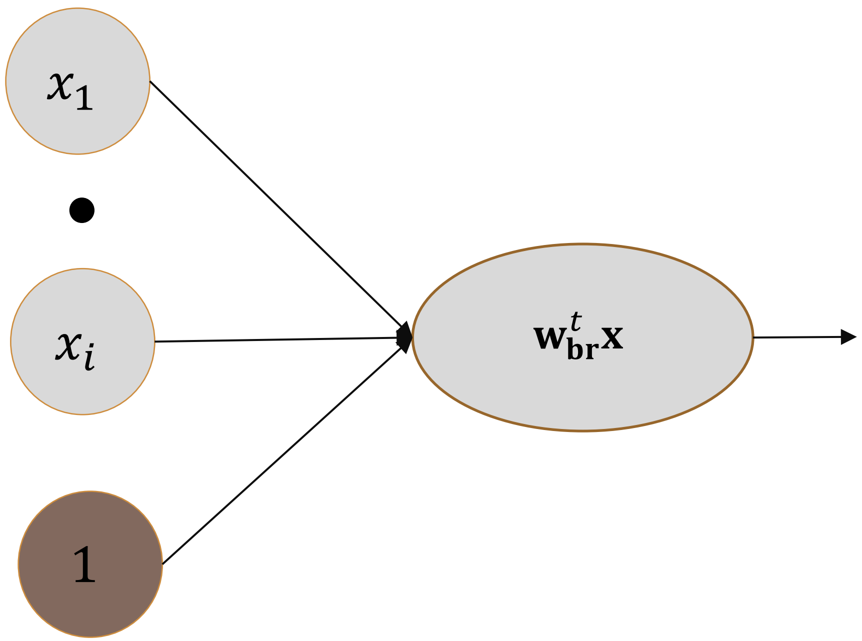
The network will learn to classify points as belonging to one or the other.

Source : http://colah.github.io/
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Adaptative Basis - Classification

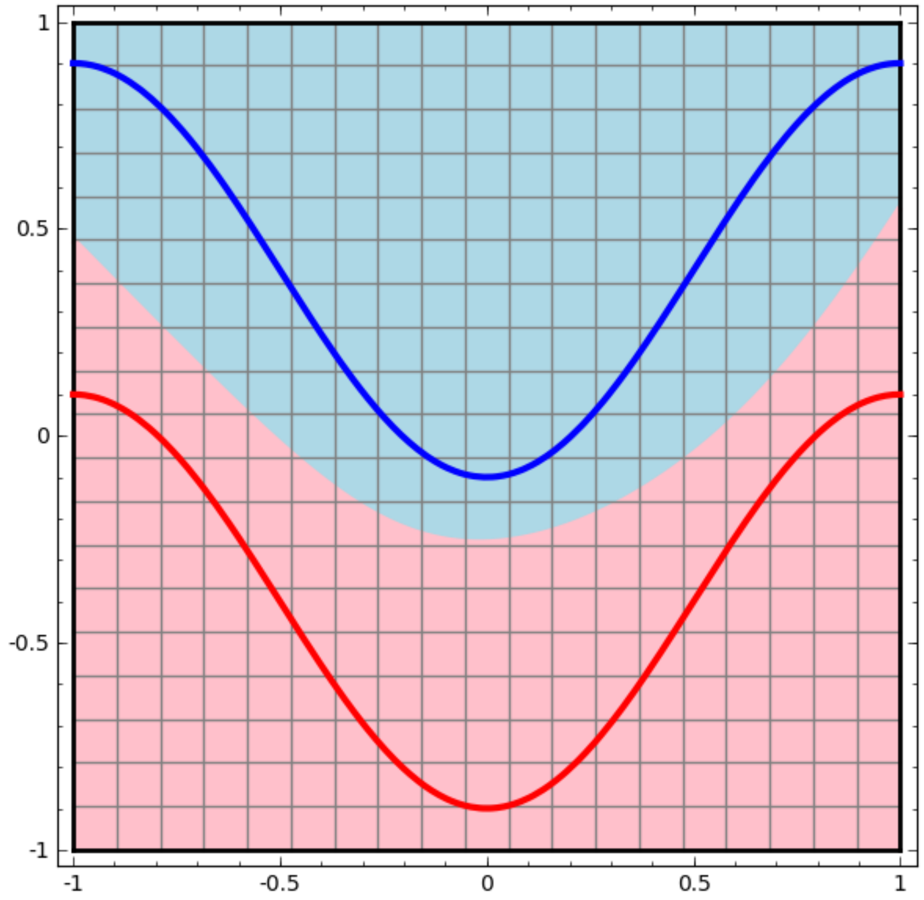


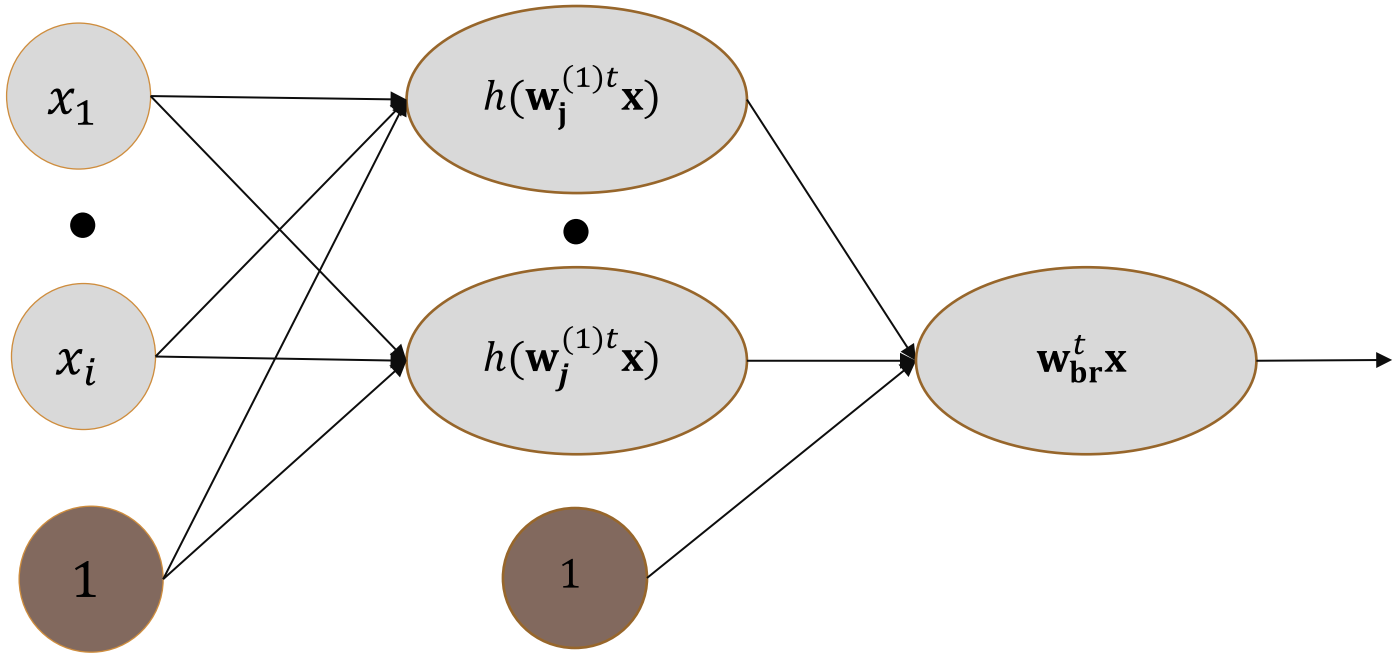
The simplest possible class of neural network,
one with only an input layer and a linear output layer.



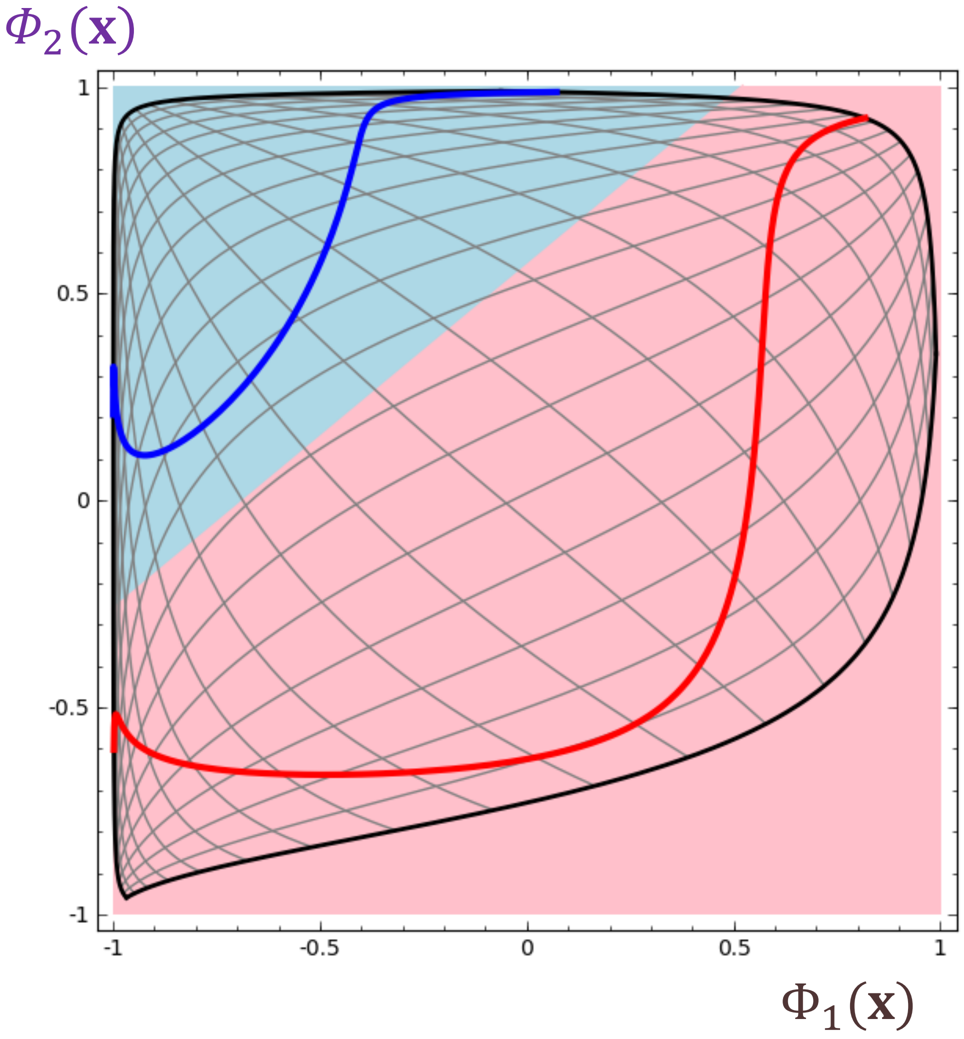
This is a linear classifier.

Adaptative Basis - Classification

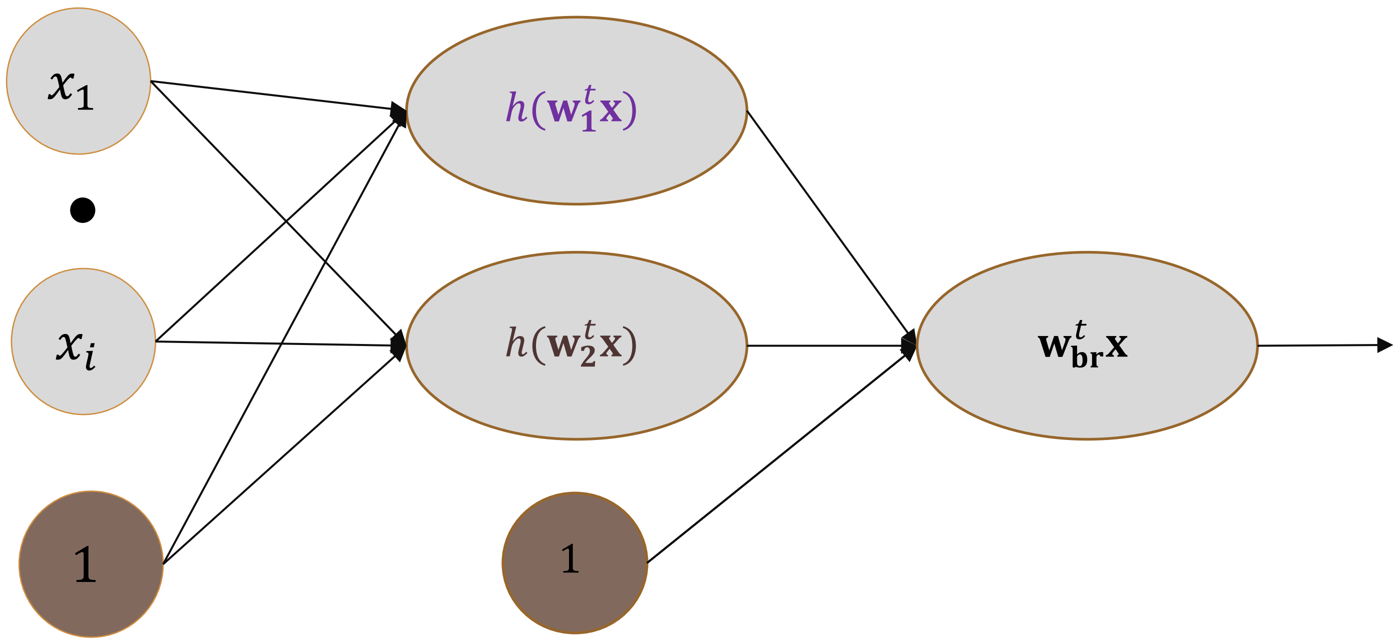




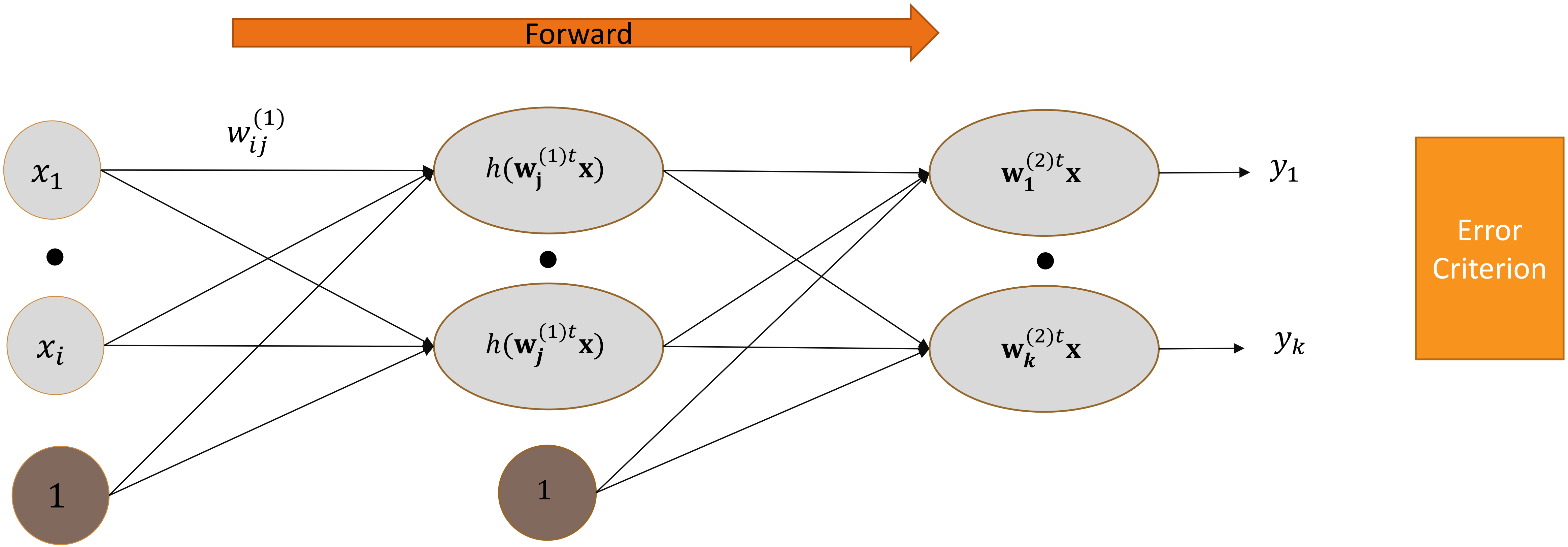
Adaptative Basis - Classification



In the adapted representation using the hidden layer outputs :



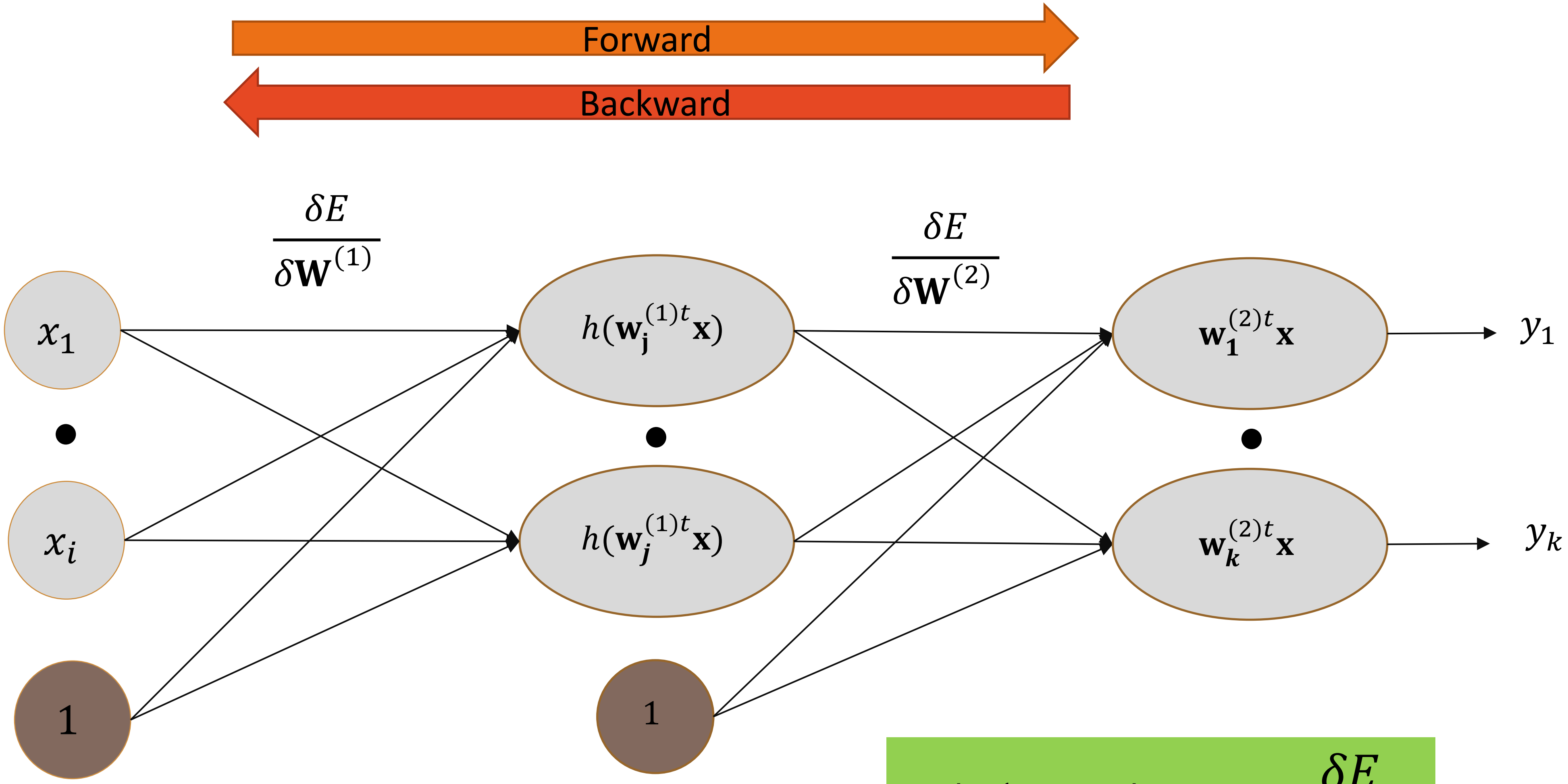
Backpropagation



Error criterions:

- Mean Square Error $E = \frac{1}{2}\|\mathrm{y} - \mathrm{t}\|_{Fro}^2$
- Cross Entropy $E = -\sum_k t_k \ln(y_k)$

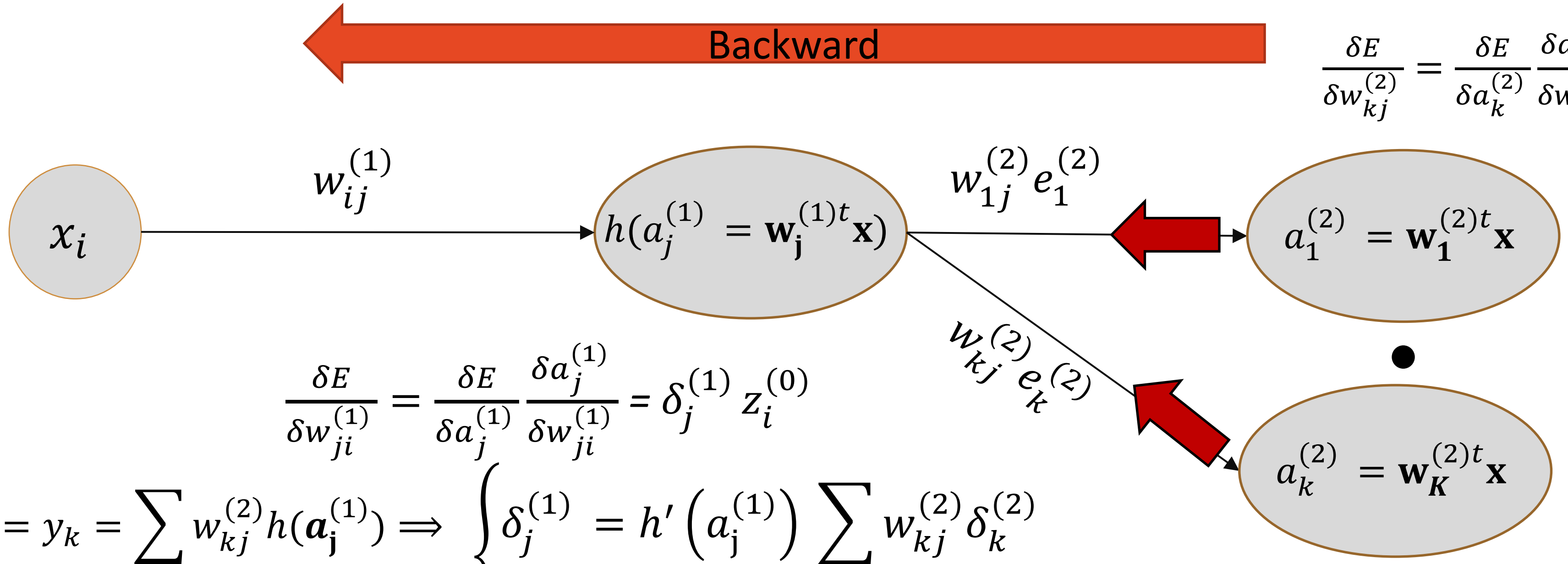
Backpropagation



Next Step : Update the weights →

$$\mathrm{w}_{\alpha\beta}^{\mathrm{t+1}} = \mathrm{w}_{\alpha\beta}^{\mathrm{t}} - \eta \frac{\delta E}{\delta \mathbf{w}_{\alpha\beta}}$$

Backpropagation



$$\frac{\delta E}{\delta w_{kj}^{(2)}} = \frac{\delta E}{\delta a_k^{(2)}} \frac{\delta a_k^{(2)}}{\delta w_{kj}^{(2)}} = \delta_k^{(2)} z_j^{(1)}$$

$$\begin{cases} \delta_k^{(2)} = \dfrac{\delta E}{\delta a_k^{(2)}} = (y_k - t_k) \\ z_j^{(1)} = h(a_\mathbf{j}^{(1)}) \end{cases}$$

Error Criterion

$$E = \sum_k \frac{1}{2}(y_k - t_k)^2$$

NB: $a_k^{(2)} = y_k$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = \frac{\delta E}{\delta a_j^{(1)}} \frac{\delta a_j^{(1)}}{\delta w_{ji}^{(1)}} = \delta_j^{(1)} z_i^{(0)}$$

$$\boldsymbol{a_k}^{(2)} = y_k = \sum_j w_{kj}^{(2)} h(\boldsymbol{a}_\mathbf{j}^{(1)}) \Rightarrow \begin{cases} \delta_j^{(1)} = h'\left(a_\mathrm{j}^{(1)}\right) \sum_k w_{kj}^{(2)} \delta_k^{(2)} \\ z_i^{(0)} = x_i \end{cases}$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = x_i(1 - h^2(a_\mathrm{j}^{(1)})) \sum_k w_{kj}^{(2)} (y_k - t_k)$$

NB : $if\ h(x) = \tanh(x)\ \ then\ h'(x) = 1 - h(x)^2$

Error criterions:

- Mean Sqare Error $E = \frac{1}{2}\|\mathrm{y} - \mathrm{t}\|_{Fro}^2$
- Cross Entropy $E = -\sum_k t_k \ln(y_k)$

Backward

$$E = \sum_k \frac{1}{2}(y_k - t_k)^2$$

$$\frac{\delta E}{\delta w_{kj}^{(2)}} = \frac{\delta E}{\delta a_k^{(2)}} \frac{\delta a_k^{(2)}}{\delta w_{kj}^{(2)}} = \delta_k^{(2)} \, z_j^{(1)} \quad \begin{cases} \delta_k^{(2)} = \dfrac{\delta E}{\delta a_k^{(2)}} = y_k - t_k \\ z_j^{(1)} = h(a_{\mathbf{j}}^{(1)}) \end{cases}$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = \frac{\delta E}{\delta a_j^{(1)}} \frac{\delta a_j^{(1)}}{\delta w_{ji}^{(1)}} = \delta_j^{(1)} \, z_i^{(0)} \quad \begin{cases} \delta_j^{(1)} = h'\left(a_j^{(1)}\right) \sum_k w_{kj}^{(2)} \delta_k^{(2)} \\ z_i^{(0)} = x_i \end{cases}$$

$NB\text{: } a_k^{(2)} = y_k$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = x_i(1 - h^2(a_{\mathrm{j}}^{(1)})) \sum_k w_{kj}^{(2)}(y_k - t_k)$$

$NB\text{: } a_k^{(2)} = y_k = \sum_j w_{kj}^{(2)} h(\boldsymbol{a}_{\mathbf{j}}^{(1)})$

$$\delta_k^{(2)} = \frac{\delta E}{\delta a_k^{(2)}} = y_k - t_k$$

NB : $if \ h(x) = \tanh(x) \ \ then \ \ h'(a) = 1 - h(a)^2$

Backprogagation of the gradient

$$\frac{\delta E}{\delta w_{kj}^{(2)}} = \frac{\delta E}{\delta a_k^{(2)}} \frac{\delta a_k^{(2)}}{\delta w_{kj}^{(2)}} = \delta_k^{(2)}\, z_j^{(1)}$$

$$a_{\mathbf{k}}^{(2)} = y_k = \sum_j w_{kj}^{(2)} h(a_{\mathbf{j}}^{(1)})$$

$$\delta_k^{(2)} = \frac{\delta E}{\delta a_k^{(2)}} = (y_k - t_k)$$

$$z_j^{(1)} = h(a_{\mathbf{j}}^{(1)})$$

$$E = \sum_k \frac{1}{2} (y_k - t_k)^2$$

$$a_{\mathbf{k}}^{(2)} = \sum_j w_{kj}^{(2)} h(a_{\mathbf{j}}^{(1)})$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = \frac{\delta E}{\delta a_j^{(1)}} \frac{\delta a_j^{(1)}}{\delta w_{ji}^{(1)}} = \delta_j^{(1)}\, z_i^{(0)}$$

$$\delta_j^{(1)} = \frac{\delta E}{\delta a_j^{(1)}} = \sum\nolimits_k \frac{\delta E}{\delta a_k^{(2)}} \frac{\delta a_k^{(2)}}{\delta a_j^{(1)}}$$

$$\frac{\delta a_k^{(2)}}{\delta a_j^{(1)}} = w_{kj}^{(2)} h'\left(a_{\mathbf{j}}^{(1)}\right)$$

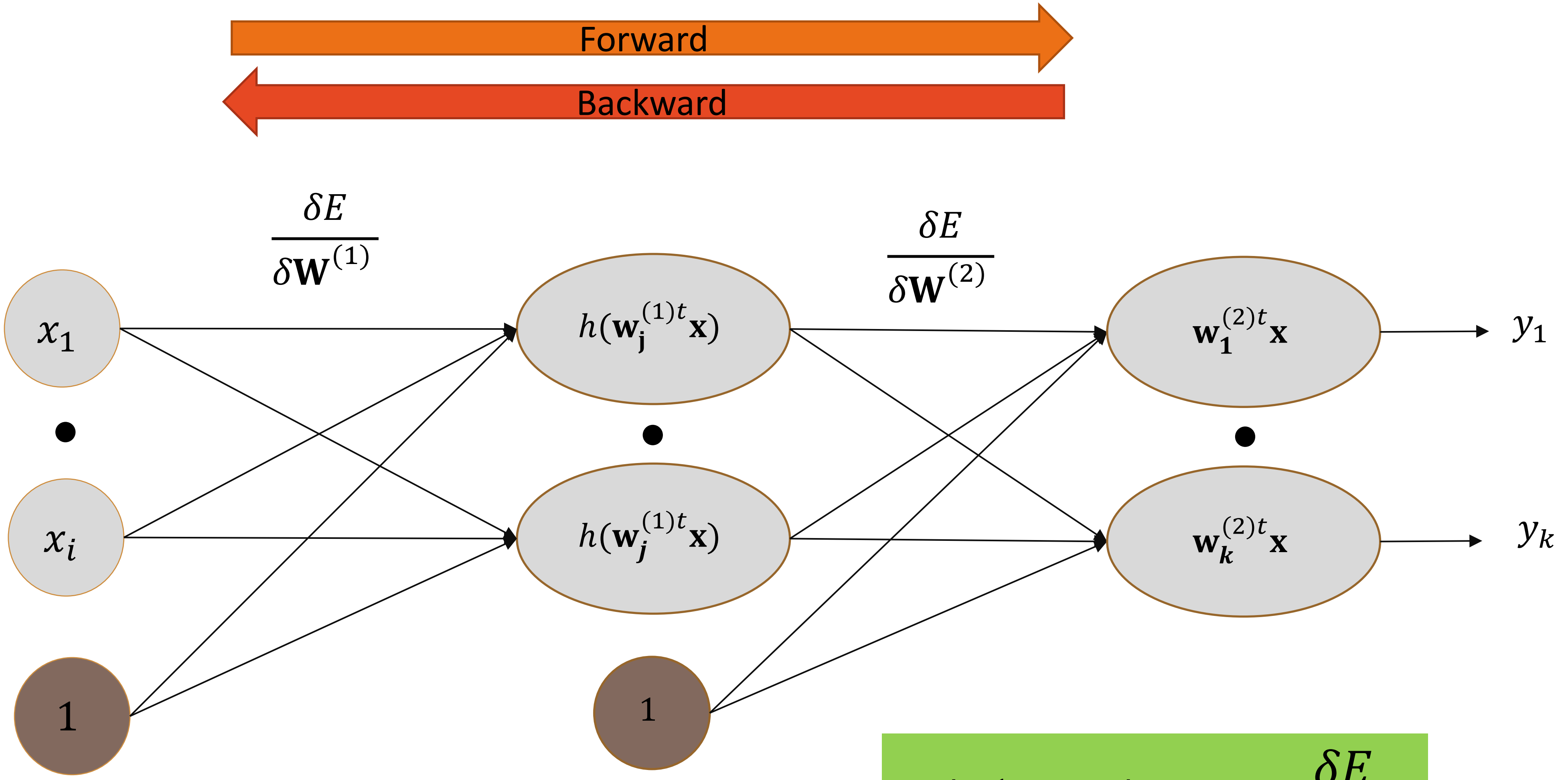
$$\delta_j^{(1)} = h'\left(a_{\mathrm{j}}^{(1)}\right) \sum_k w_{kj}^{(2)} \delta_k^{(2)}$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = (1 - h^2(a_{\mathbf{j}}^{(1)})) \sum_k w_{kj}^{(2)} (y_k - t_k) \quad x_i$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = \delta_j^{(1)}\, z_i^{(0)}$$

NB : $if\ h(x) = \tanh(x)\ \ then\ h'(x) = 1 - h(x)^2$

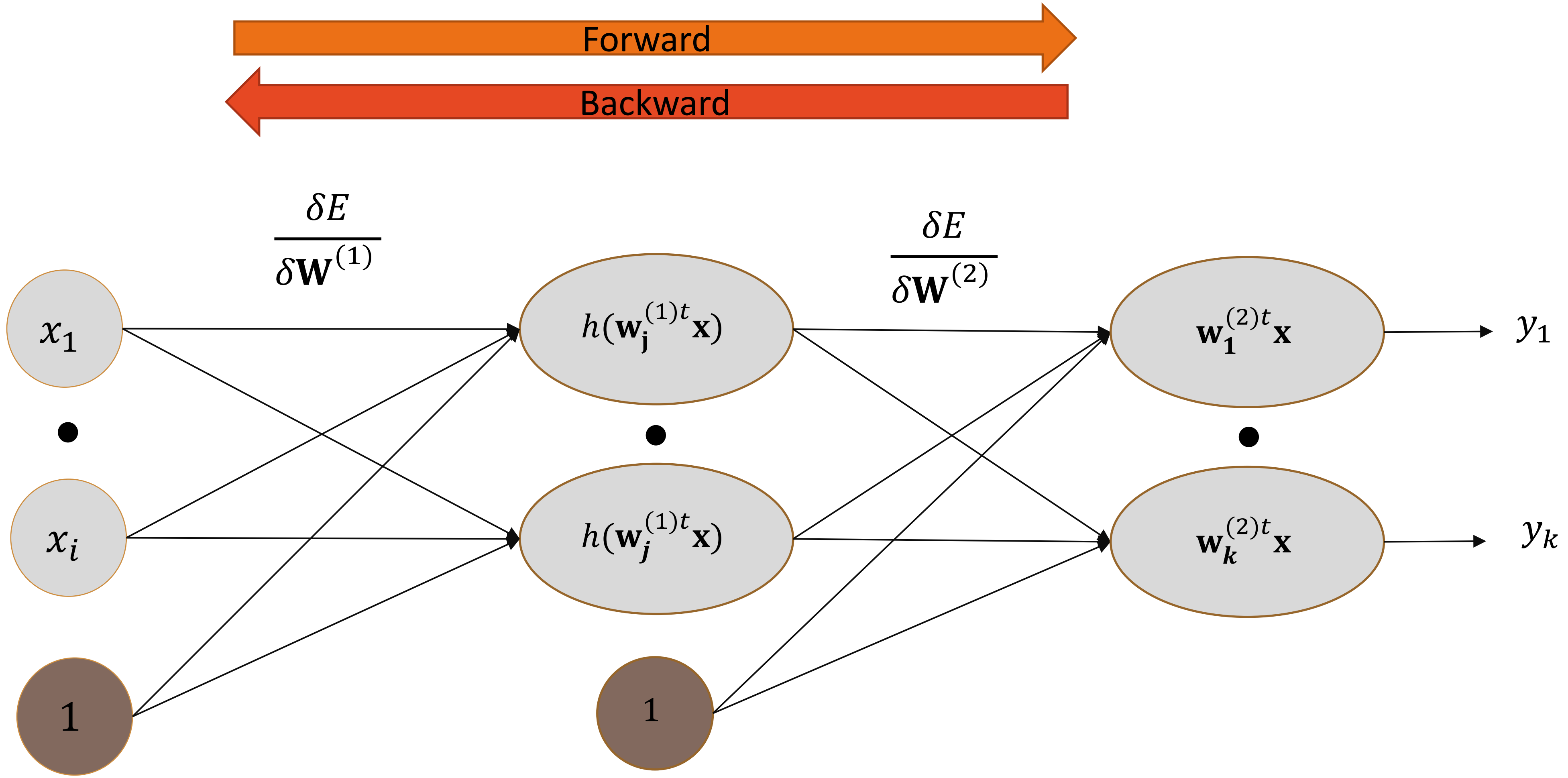
Backpropagation



Next Step : Update the weights →

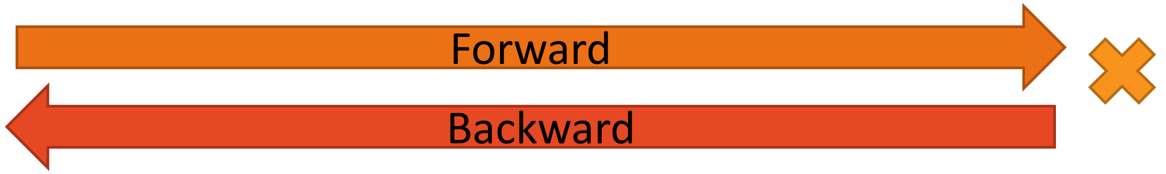
$$\mathrm{w}_{\alpha\beta}^{t+1} = \mathrm{w}_{\alpha\beta}^{t} - \eta \frac{\delta E}{\delta \mathbf{w}_{\alpha\beta}}$$

Backpropagation



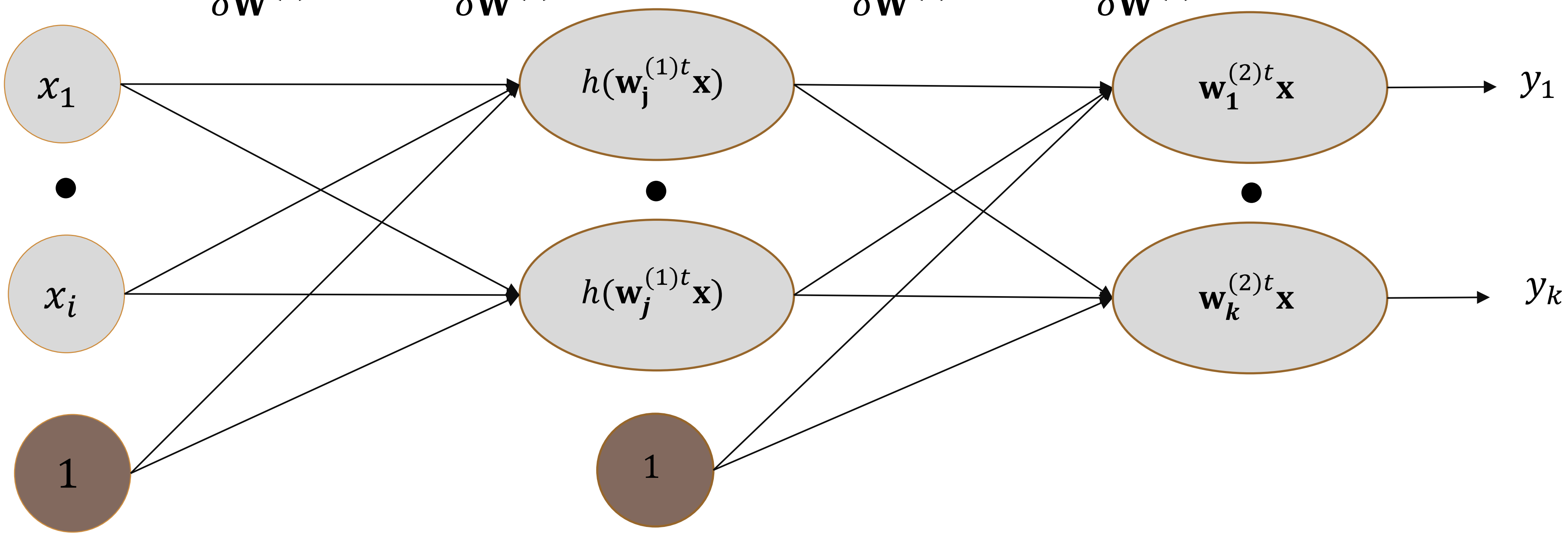
Stochastic : The error is accumulated over one input

Backpropagation



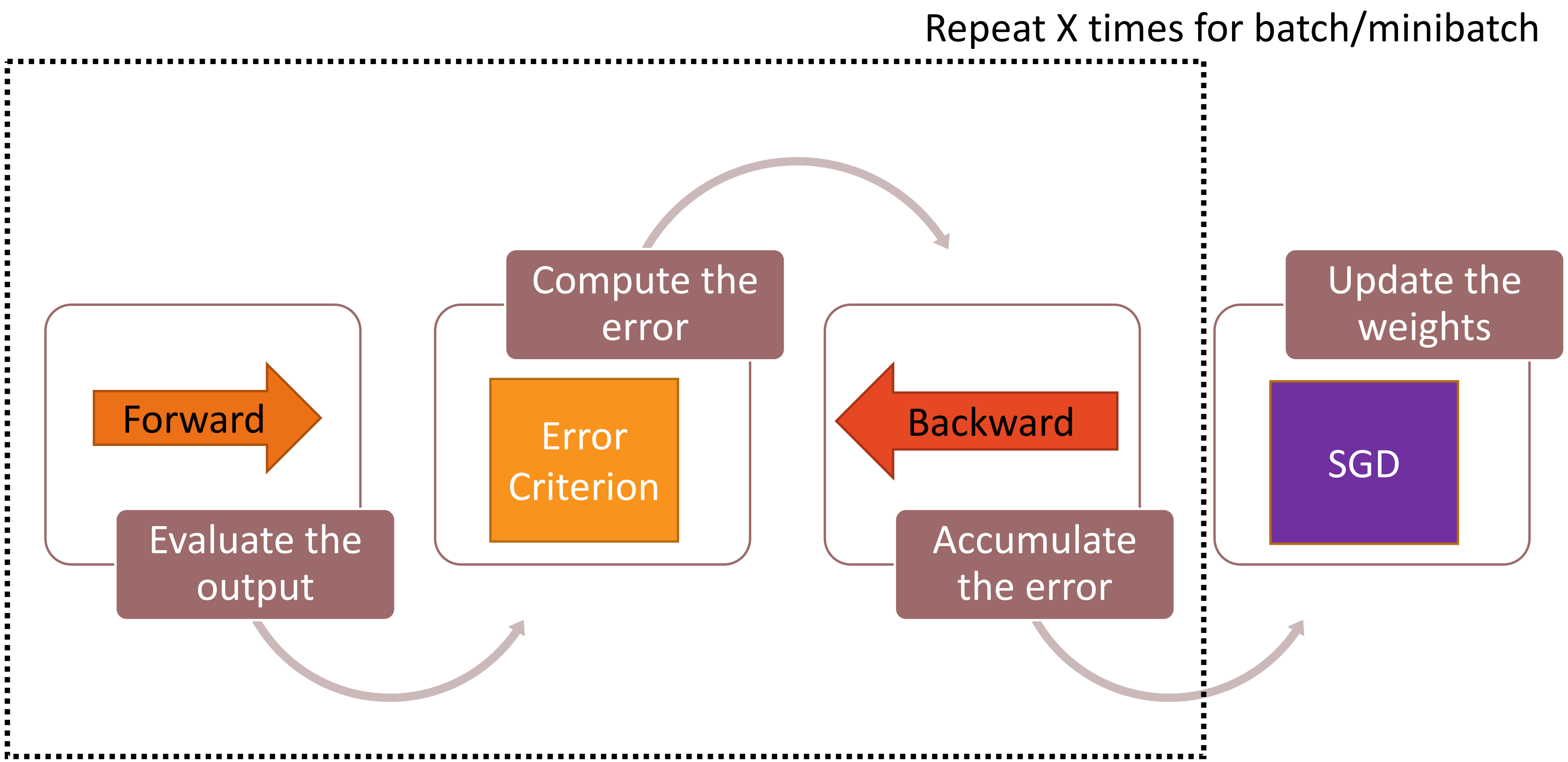
M inputs picked randomly

$$\mathbf{e}^{(1)} = \frac{\delta E_1}{\delta \mathbf{W}^{(1)}} + \ldots + \frac{\delta E_M}{\delta \mathbf{W}^{(1)}} \qquad \mathbf{e}^{(2)} = \frac{\delta E_1}{\delta \mathbf{W}^{(2)}} + \ldots + \frac{\delta E_M}{\delta \mathbf{W}^{(2)}}$$



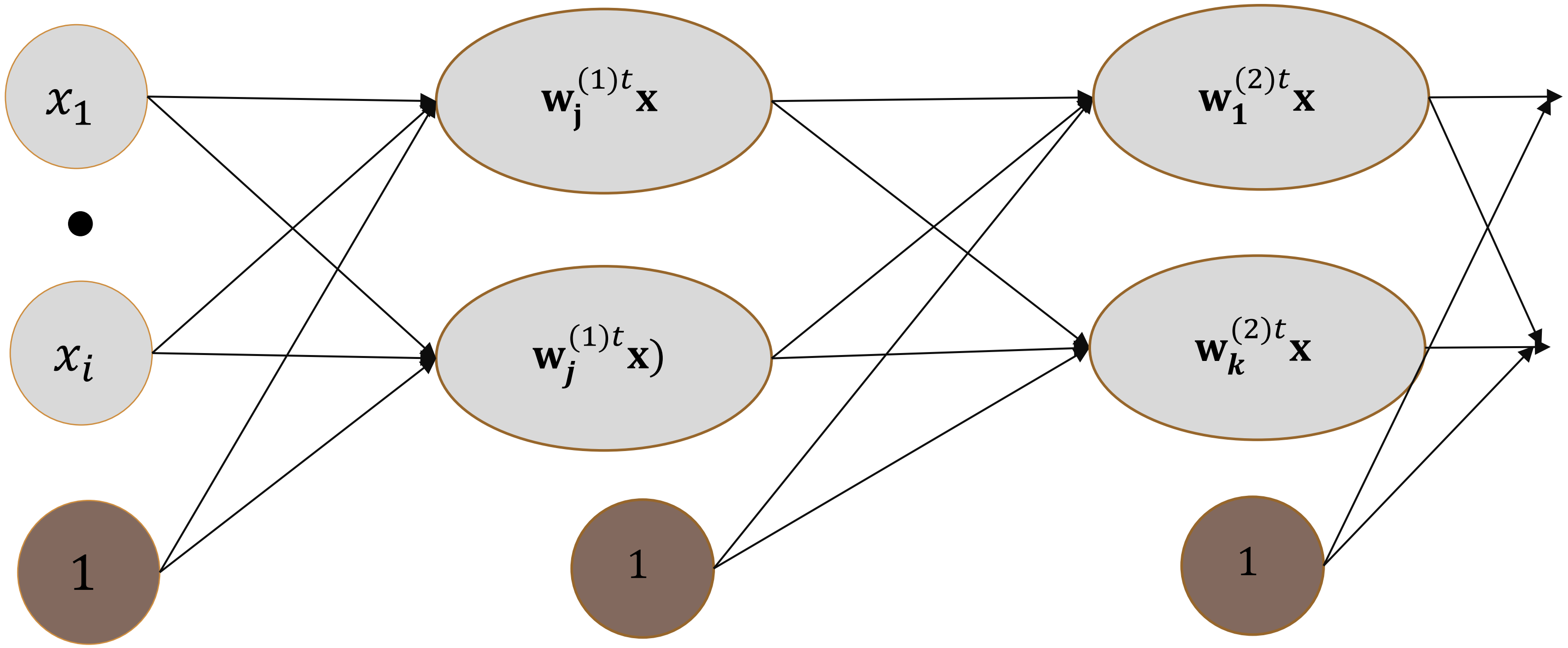
Mini-batch: The error is accumulated over M inputs

Backpropagation



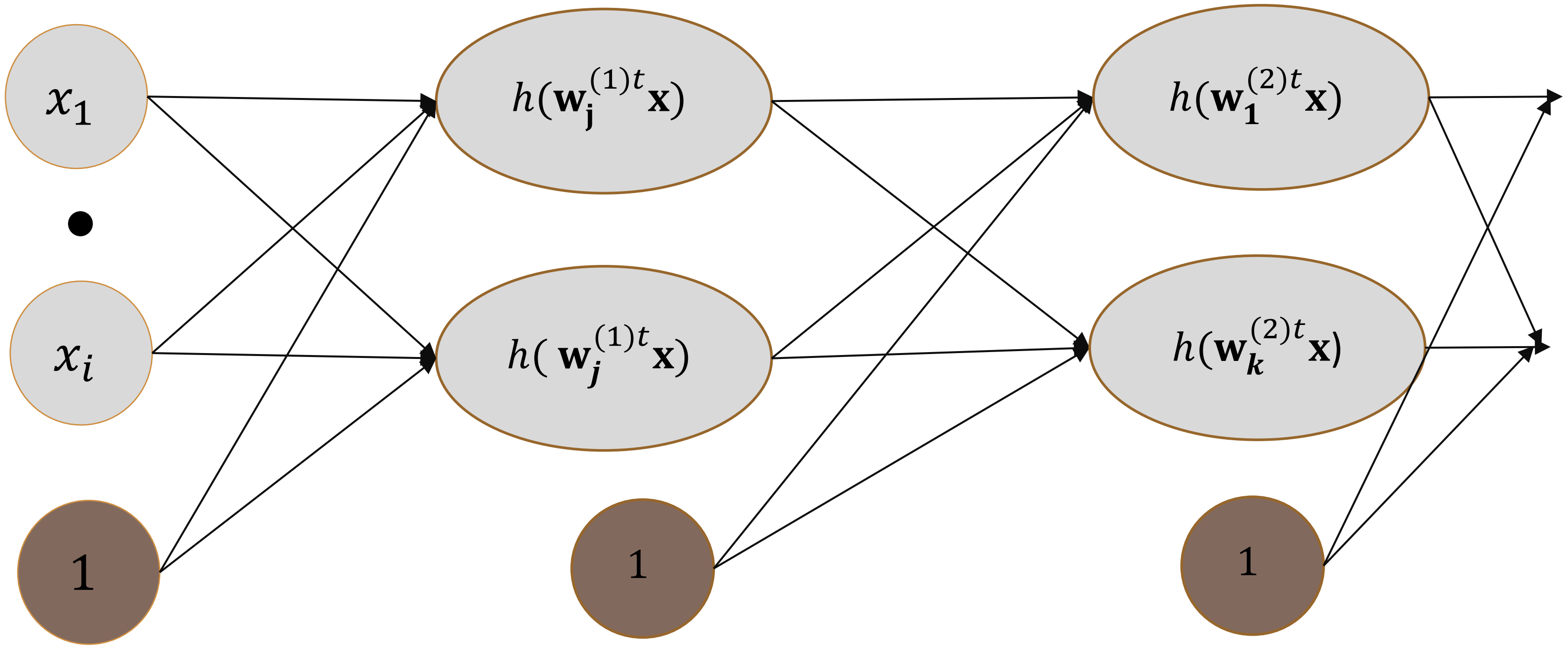
Some important properties

- Linear networks:
  - Proof of convergence
  - N-layers linear networks can be turned into a 2-layer linear networks



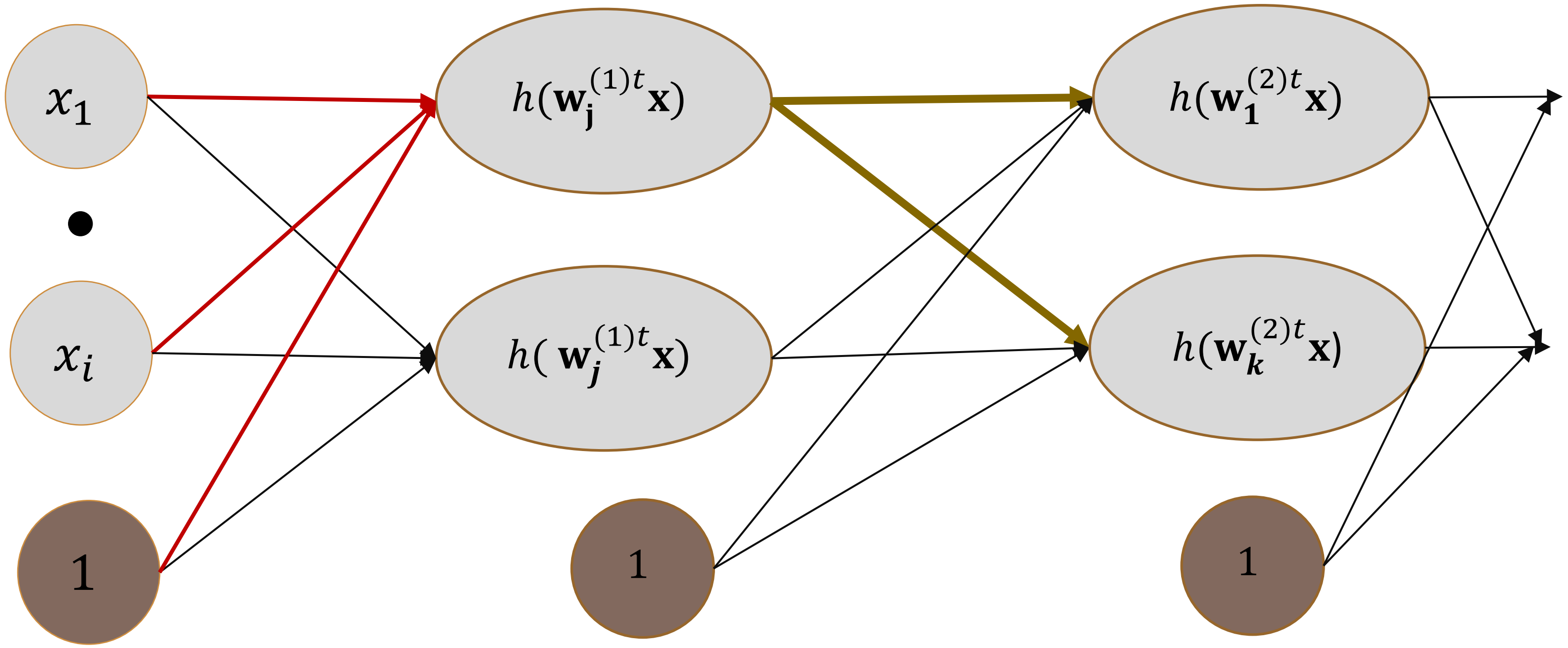
Some important properties

- Non-Linear networks:
  - h(z) = tanh / Sigmoid / relu …
  - $\mathbf{W}^{t+1} = \mathbf{W}^{t} \ - \eta \frac{\delta E}{\delta W}$ is non-convex



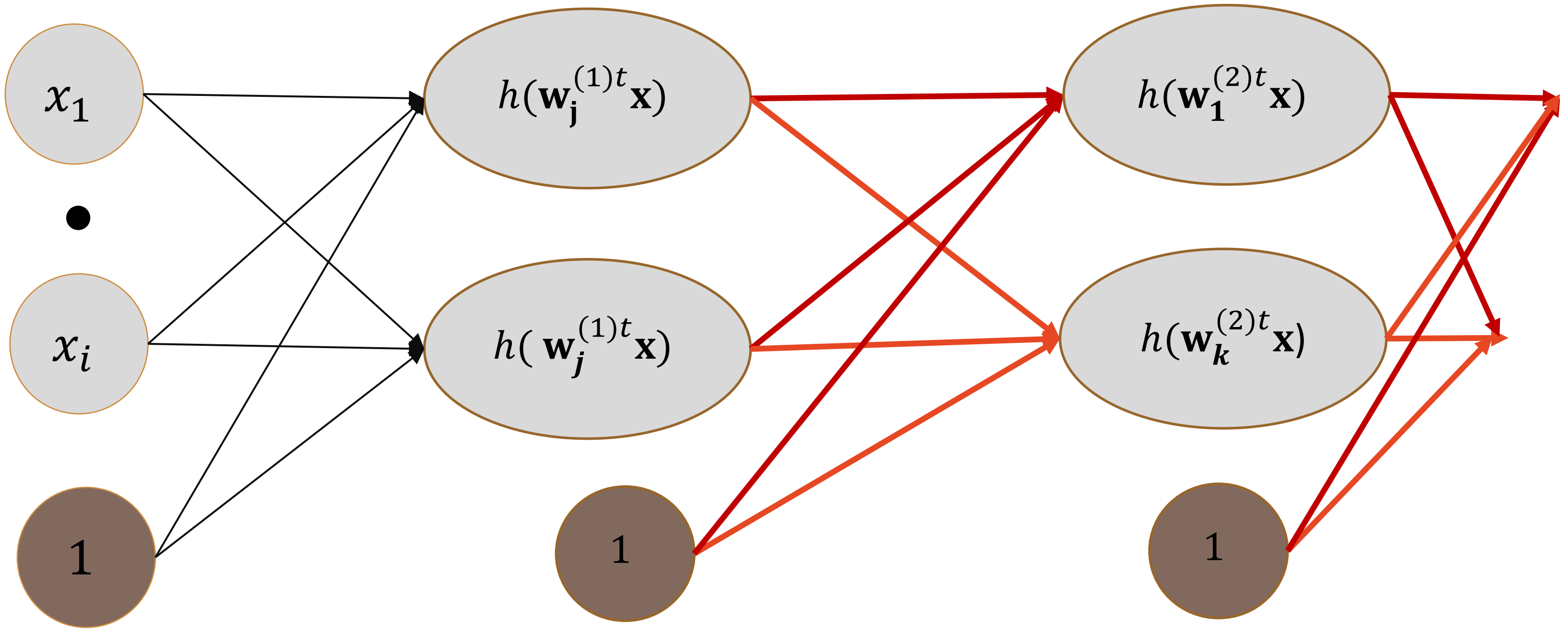
Some important properties

- Weight spaces symmetries:
  - $\tanh(x) = -\tanh(x)$ → 2^M sign flip



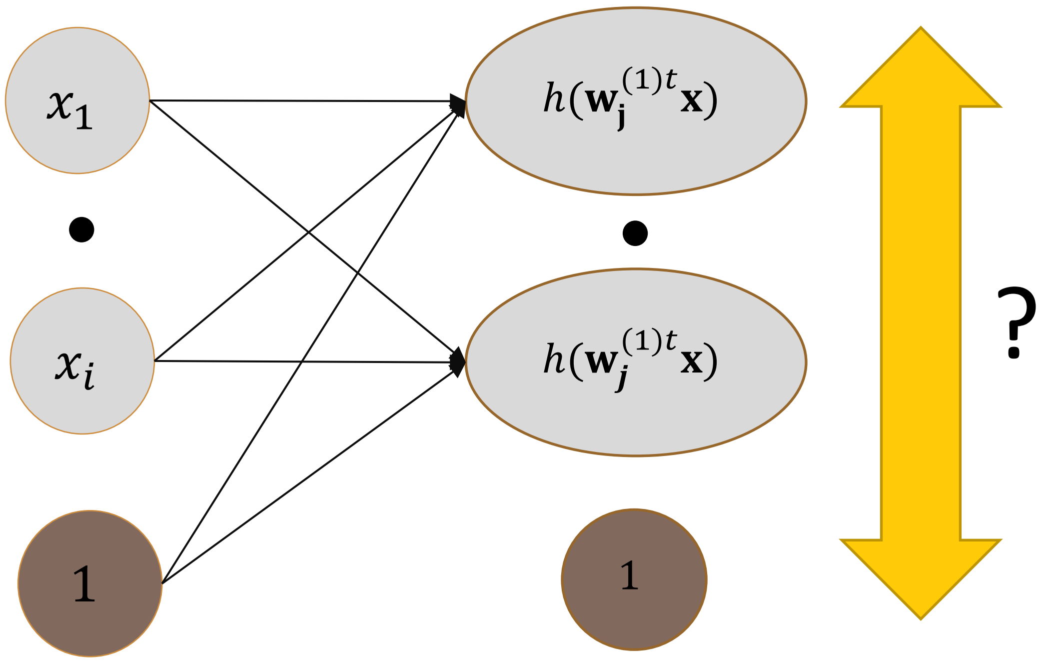
Some important properties

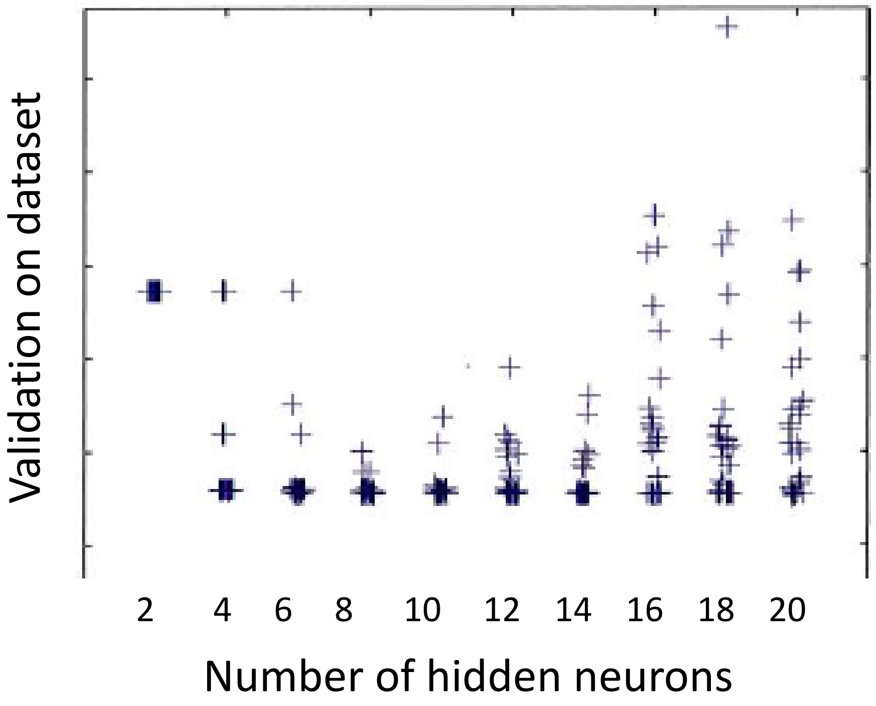
- Weight spaces symmetries:
  - $\tanh(x) = -\tanh(x)$ → 2^M sign flip
  - Interchanging the weight values → M! flip



What is a good neural networks?

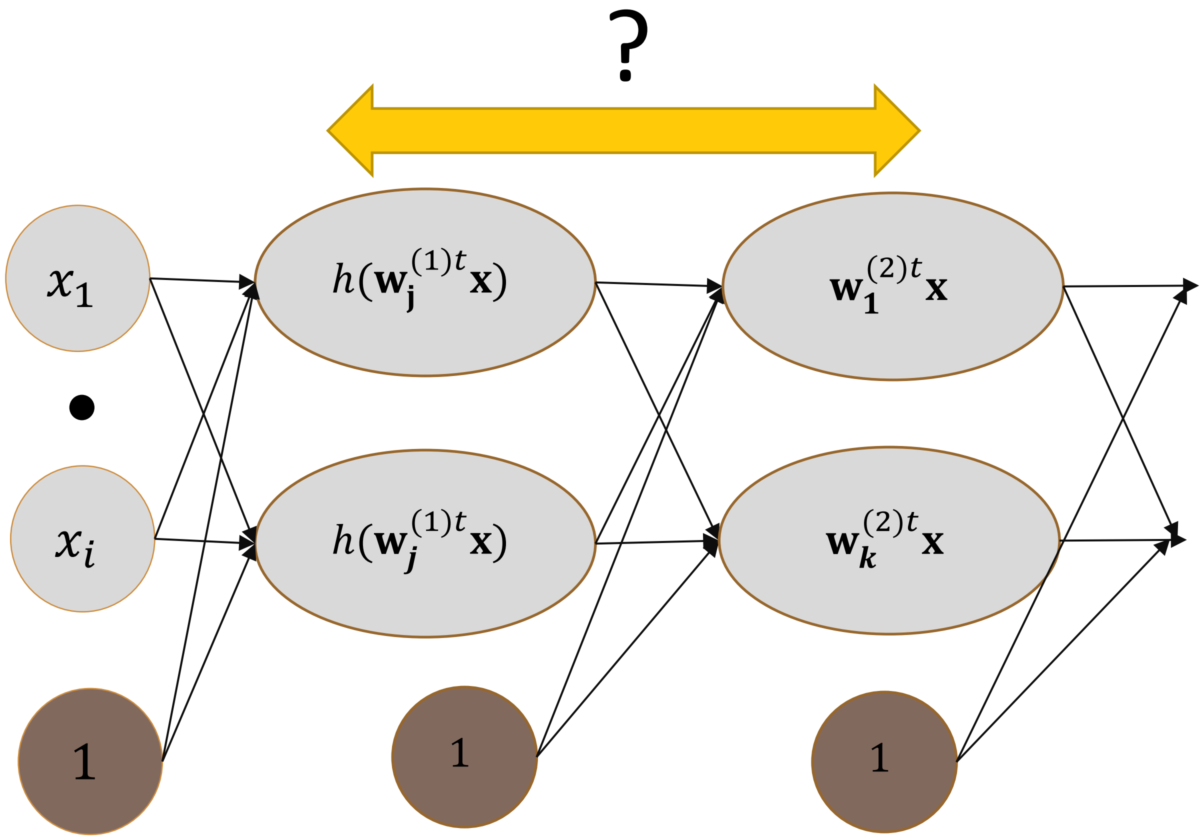
The larger, the better?





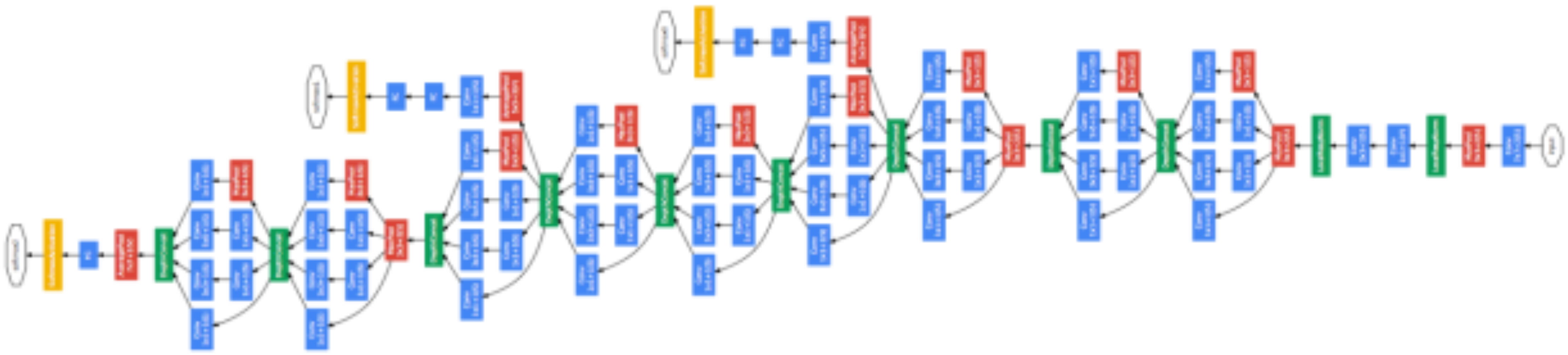
What is a good neural networks?

The deeper, the better?



What is a good neural networks?

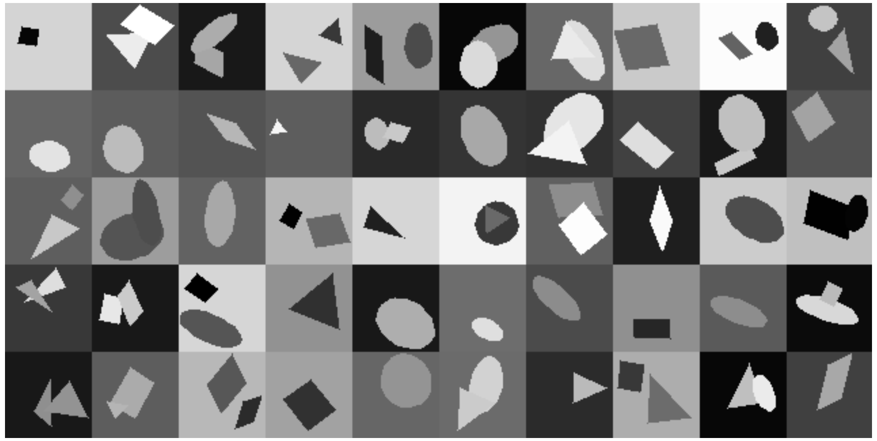
The deeper, the better?



Inception Network by Google

What is a good neural networks?

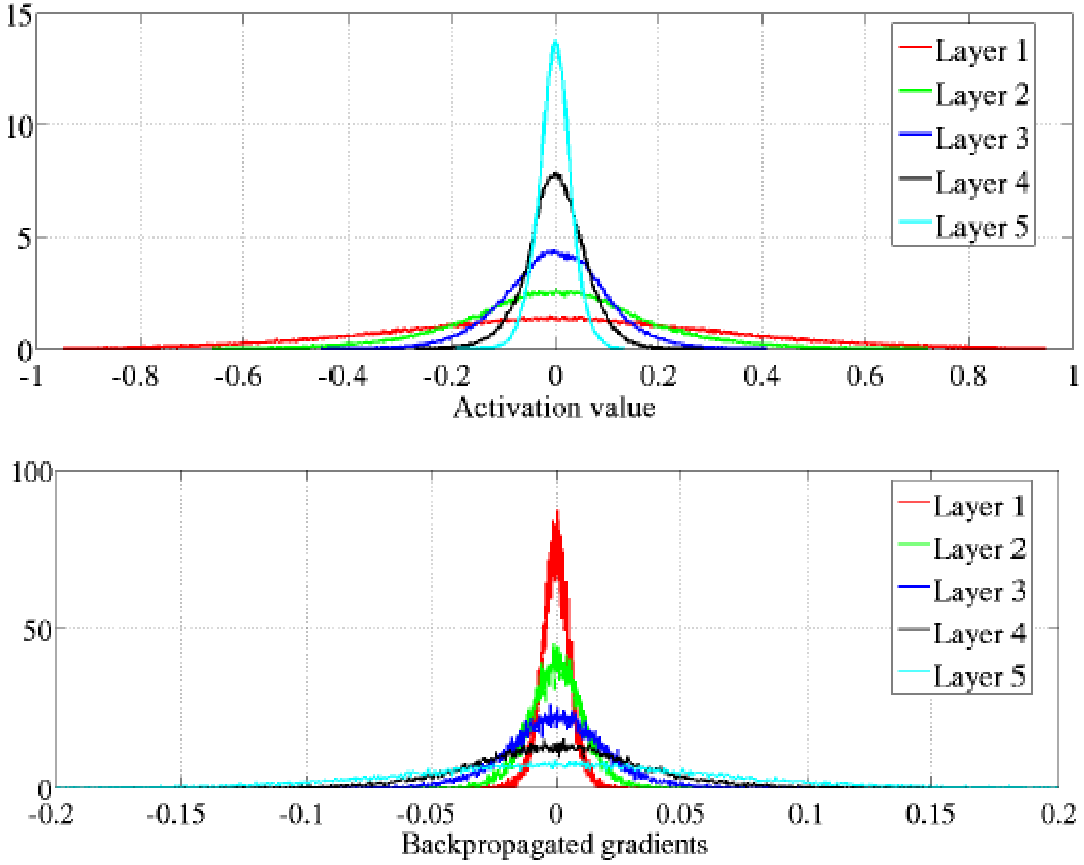
The deeper, the better?



Shapeset-3×2 images at 64×64 resolution

3 objects : parallelogram, triangle, or ellipse

1 or 2 objects can be present → 9 possible classifications.



Source : Xavier Glorot, Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, 2010

What is a good neural networks?

To summarize:

- A network must be large enough.
  - Too small : underfitting
  - Too big : overfitting
- The deeper the better
  - Beware of vanishing gradient

What can we do?

- Increase the number of samples ☺
- Regularization
- Better initialization

What about gradient Descent?

- LBFGS
- Natural Gradient
- rProp

Momentum = 0.8

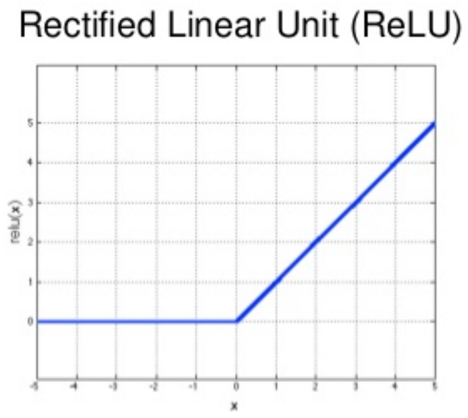
When network stop learning, divide the learning rate by 10

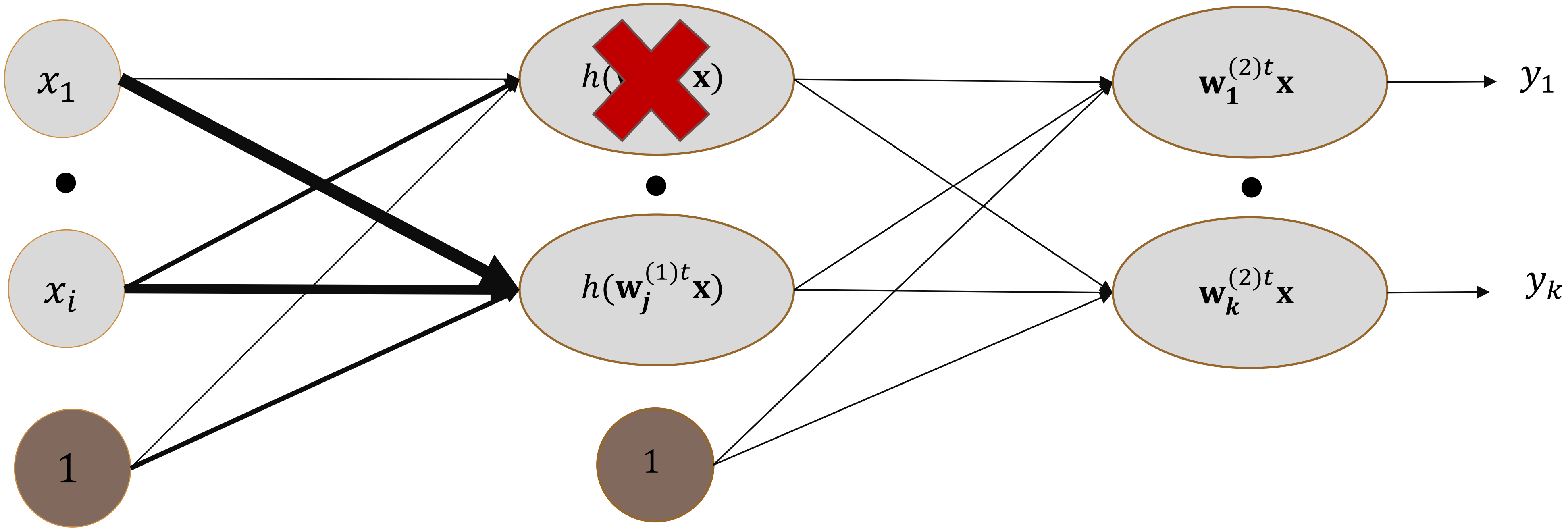
Regularization

Weight Decays	• L2 regularization : $w_{\alpha\beta}^{t+1} = w_{\alpha\beta}^{t} - \eta \left(\frac{\delta E}{\delta w_{\alpha\beta}} + \boldsymbol{\lambda w_{\alpha\beta}} \right)$
Corrupted input	• Add noise, modifying data. → Increase data / redundancy
Sparsity	• Dropout • Rectified linear units
Weight Matrix reduction	• Approximating the weight matrix by a low rank matrix → $\mathbf{W}_{[\mathbf{I} * \mathbf{J}]} = \mathbf{U}_{[\mathbf{I} * \mathbf{K}]} * \mathbf{V}_{[\mathbf{K} * \mathbf{J}]}$

Regularization

Relu : Inhibit hidden neurons when the input is too low

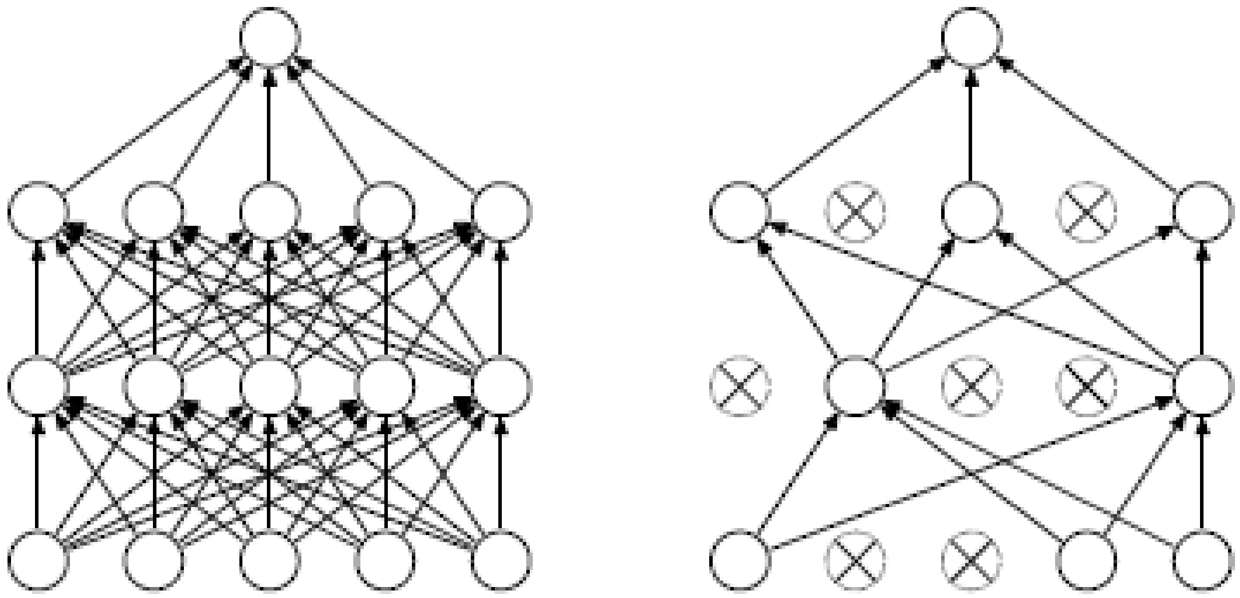




Regularization

Dropout

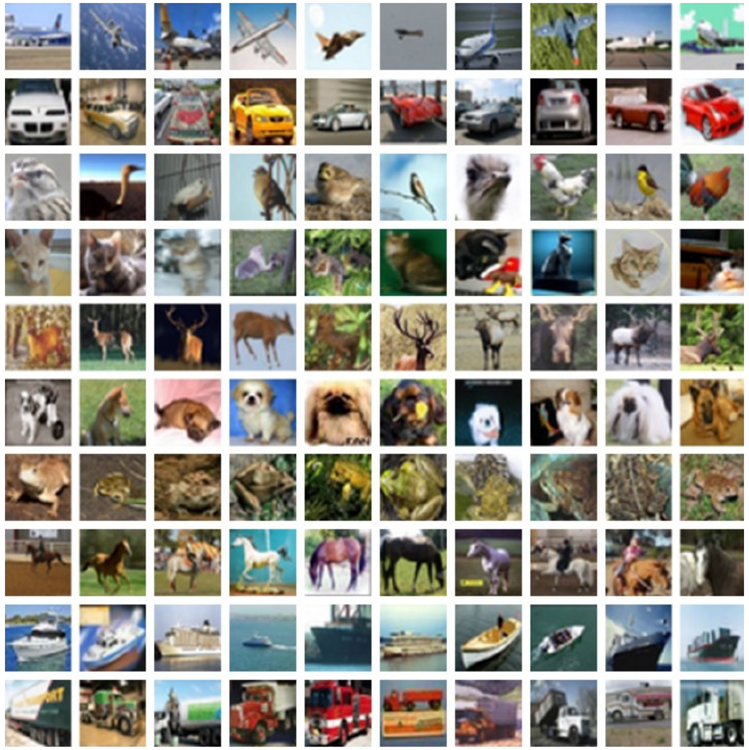
- Training : Randomly (with probability *p*) remove some nodes in the forward step
- Evaluating : Multiply the weights by *1-p*



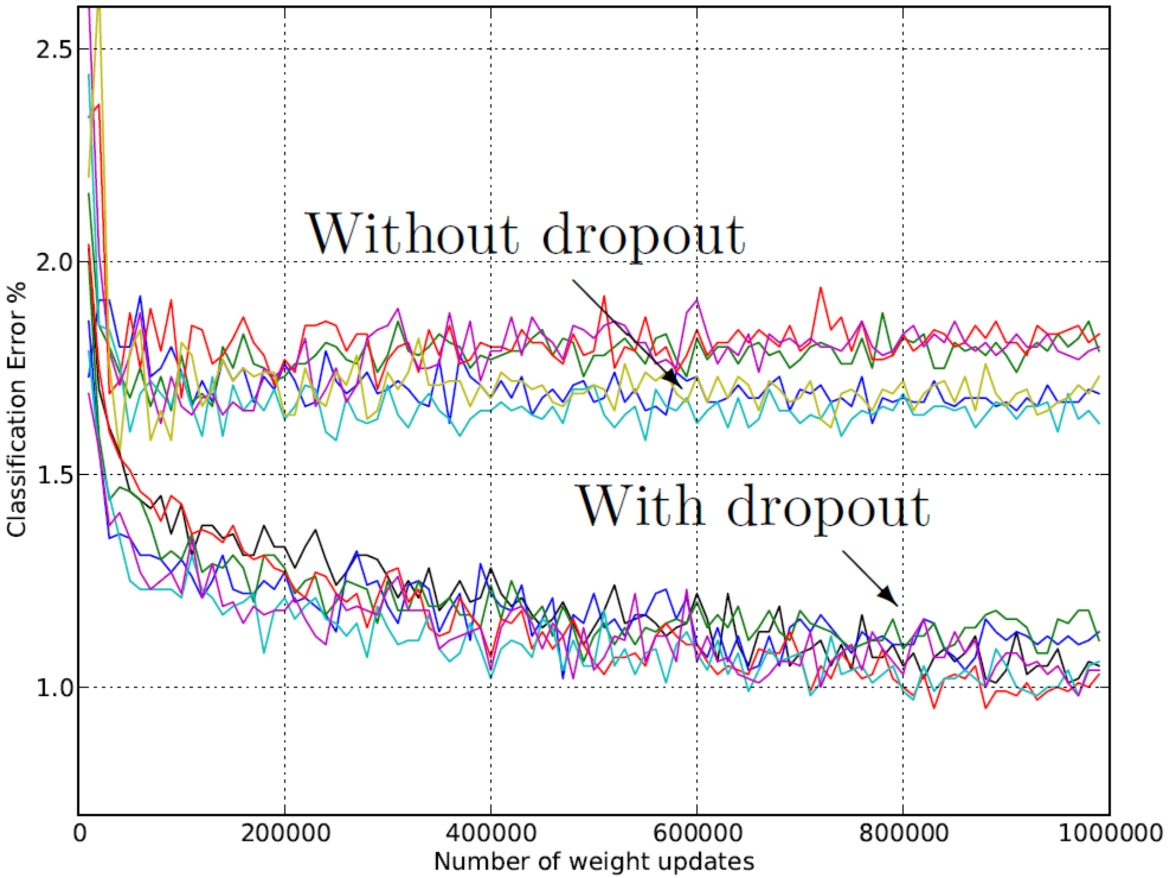
(a) Standard Neural Net (b) After applying dropout.

<u>Source :</u> Nitish Srivastava, Georey Hinton, Alex Krizhevsky , Ilya Sutskever , Ruslan Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overtting*, 2014

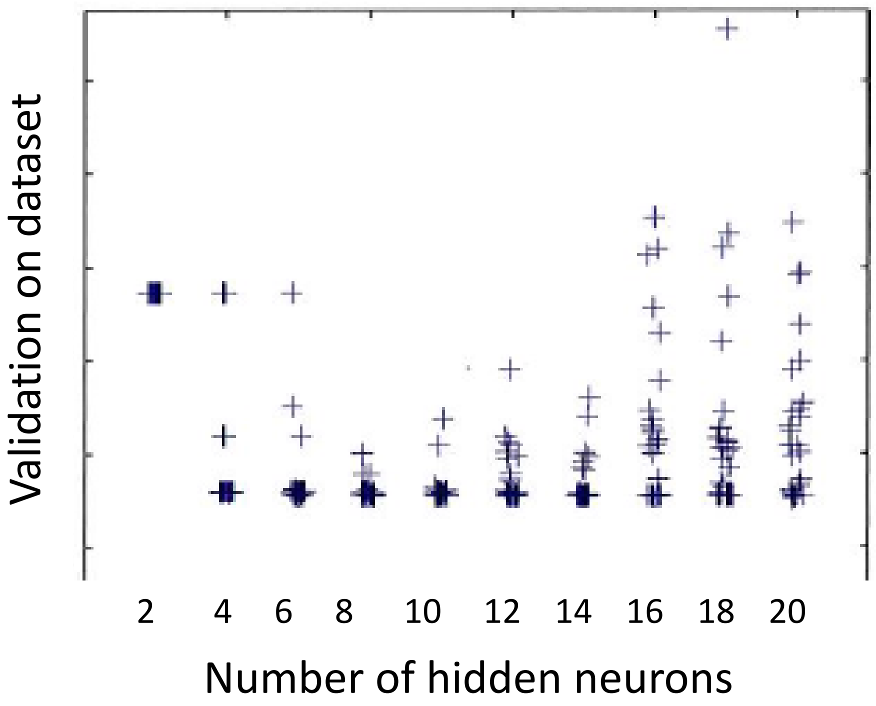
Regularization



CIFAR-10
10 objects : boat, plane, truck etc.



Initialization



Idea :

- Try to initialize the network in a clever way

Goal :

- Avoid local minima?
- Increase final score

Solution :

- Restricted Boltzmann machine (Hinton)
- Stacked Autoencoders (Bengio)

Initialization

Fan-in rule:

$$w_{ij} \sim U[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}]$$

$$b_i = 0$$

Where

- w_{ij} is a edge weight
- w_{ij} is a edge bias
- n_{in} is the number of input edges

Assumption:

- inputs have zero mean
- Input has a one standard deviation

Initialization

Normalized Fan-in rule:

$$w_{ij} \sim U[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}]$$

$$b_i = 0$$

Where

- w_{ij} is a edge weight
- w_{ij} is a edge bias
- n_{in} is the number of input edges

Assumption:

- inputs have zero mean
- Input has a one standard deviation
- Transfer functions are tangents

Goal :

- Enable the activation to keep the input properties through the layer

Source : Xavier Glorot, Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, 2010

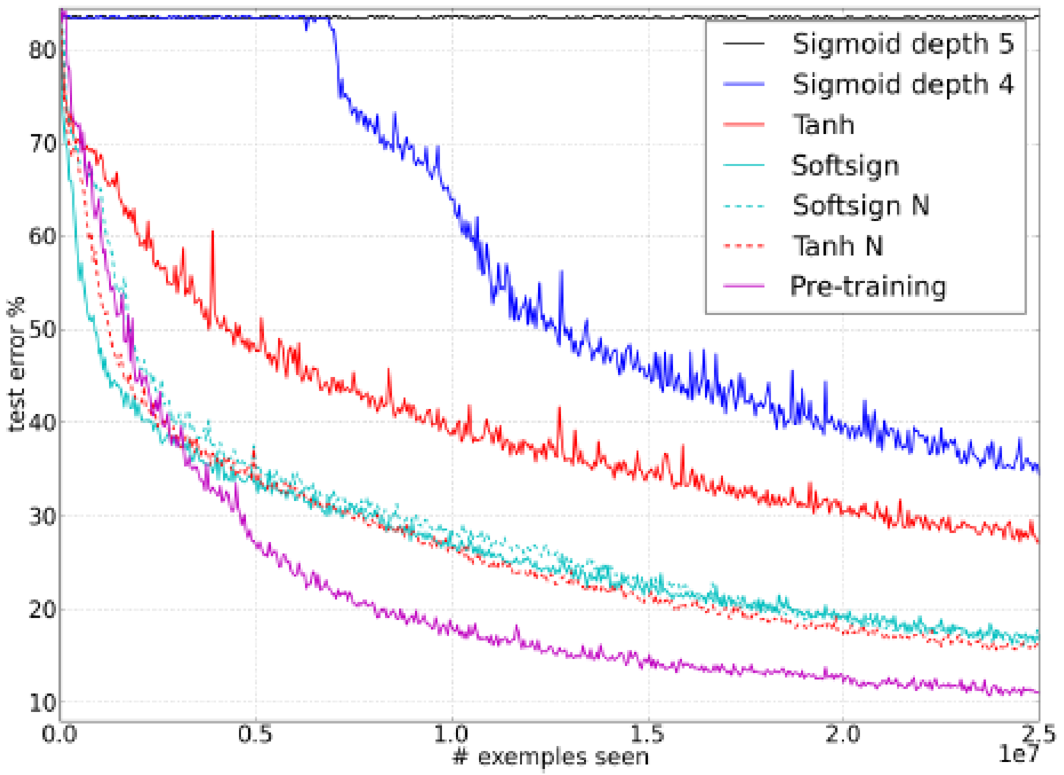
Initialization

Normalized Fan-in rule:

$$w_{ij} \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right]$$
$$b_i = 0$$

Where

- w_{ij} is a edge weight
- w_{ij} is a edge bias
- n_{in} is the number of input edges

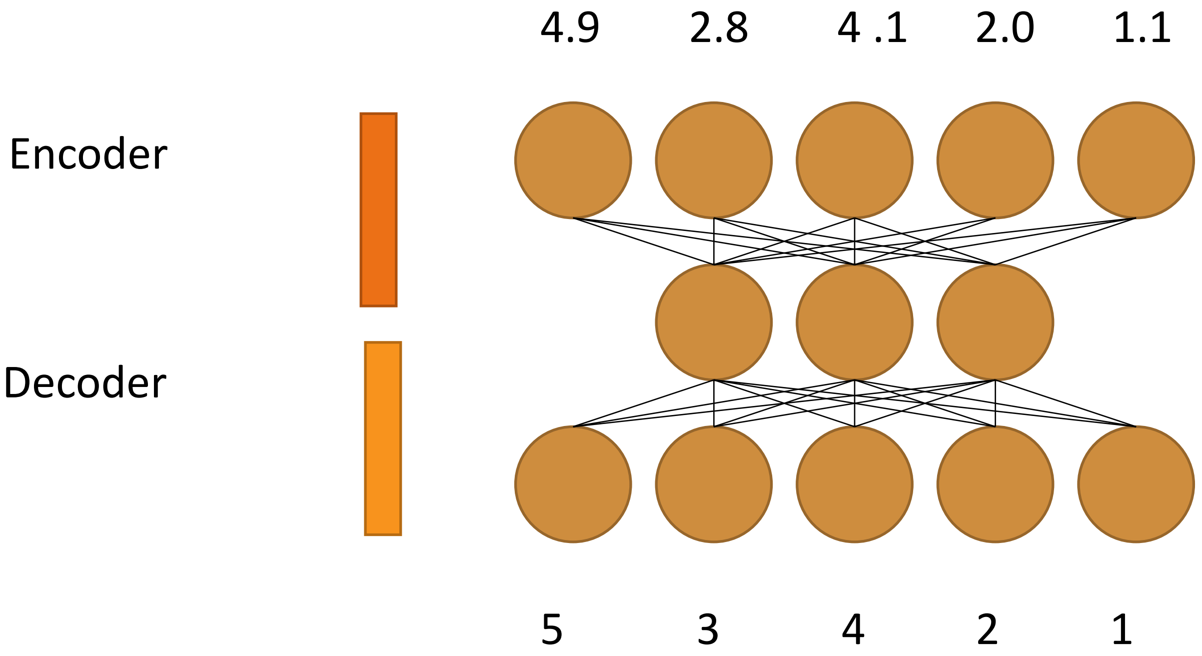


Source : Xavier Glorot, Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, 2010

Initialization

Autoencoders:

Input reconstruction $E = \frac{1}{N} \sum(\mathbf{x_n} - \widehat{\mathbf{x_n}})$

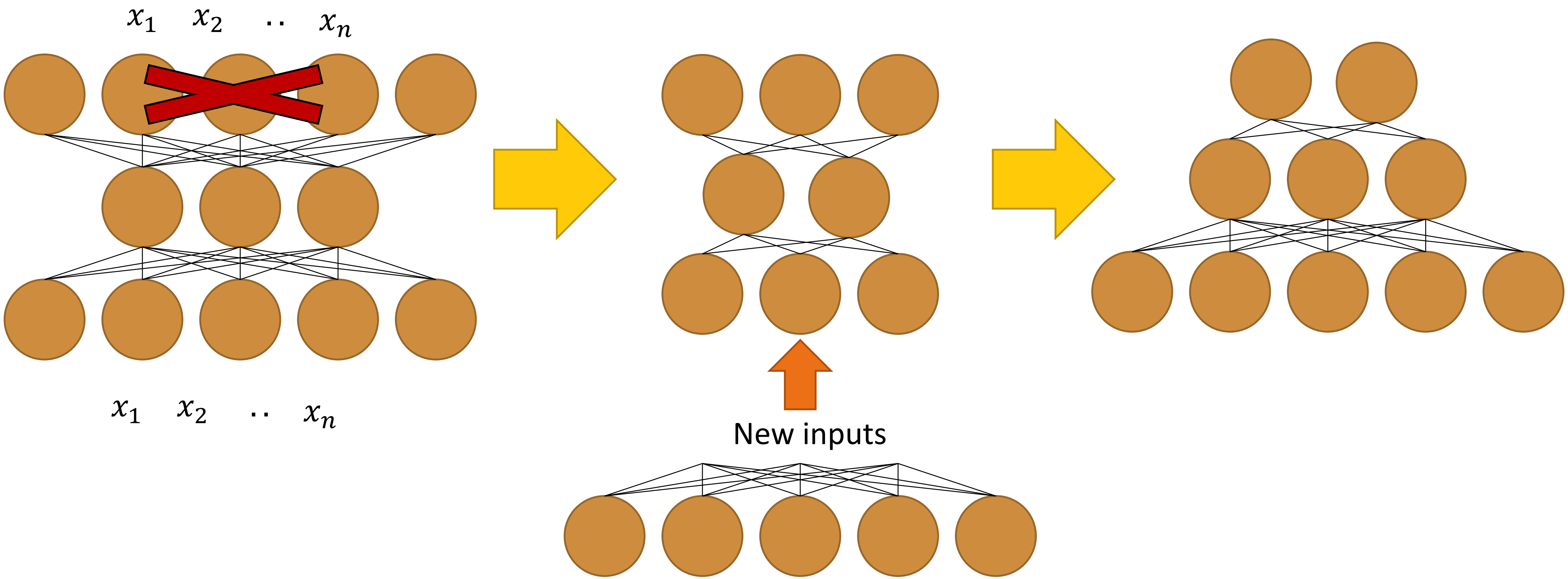


Dimension reduction

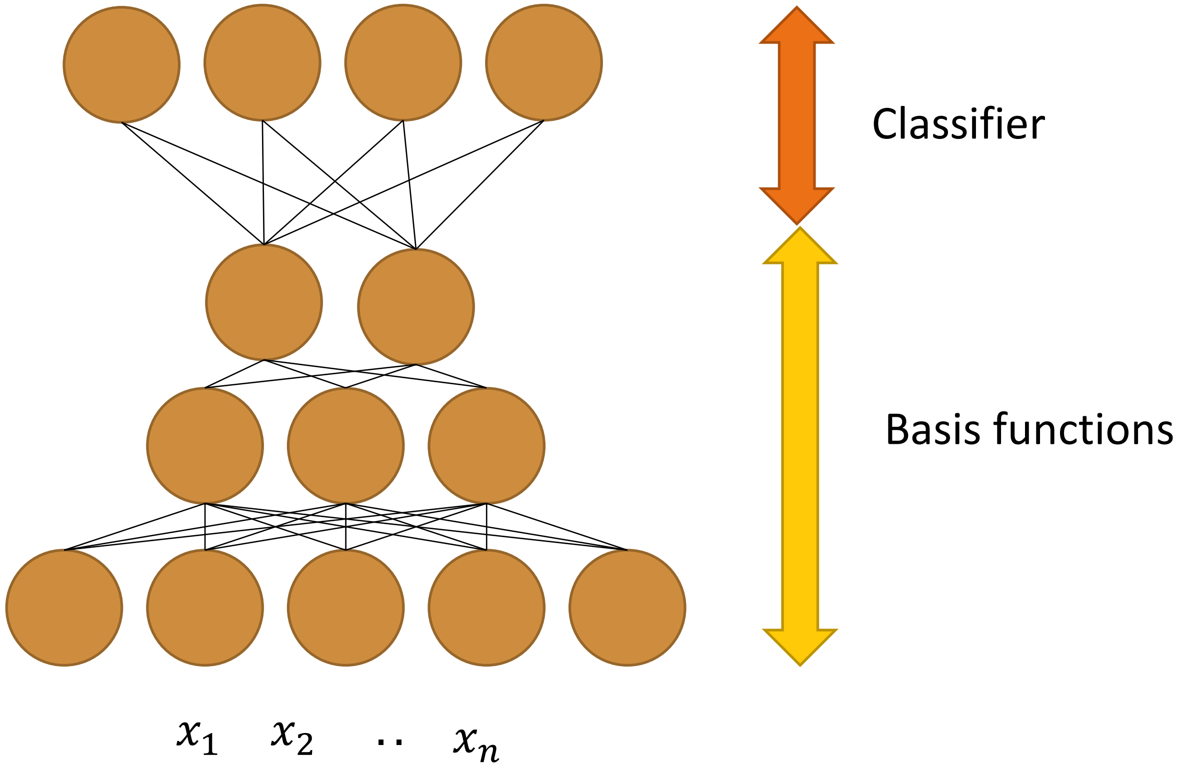
inputs

Initialization

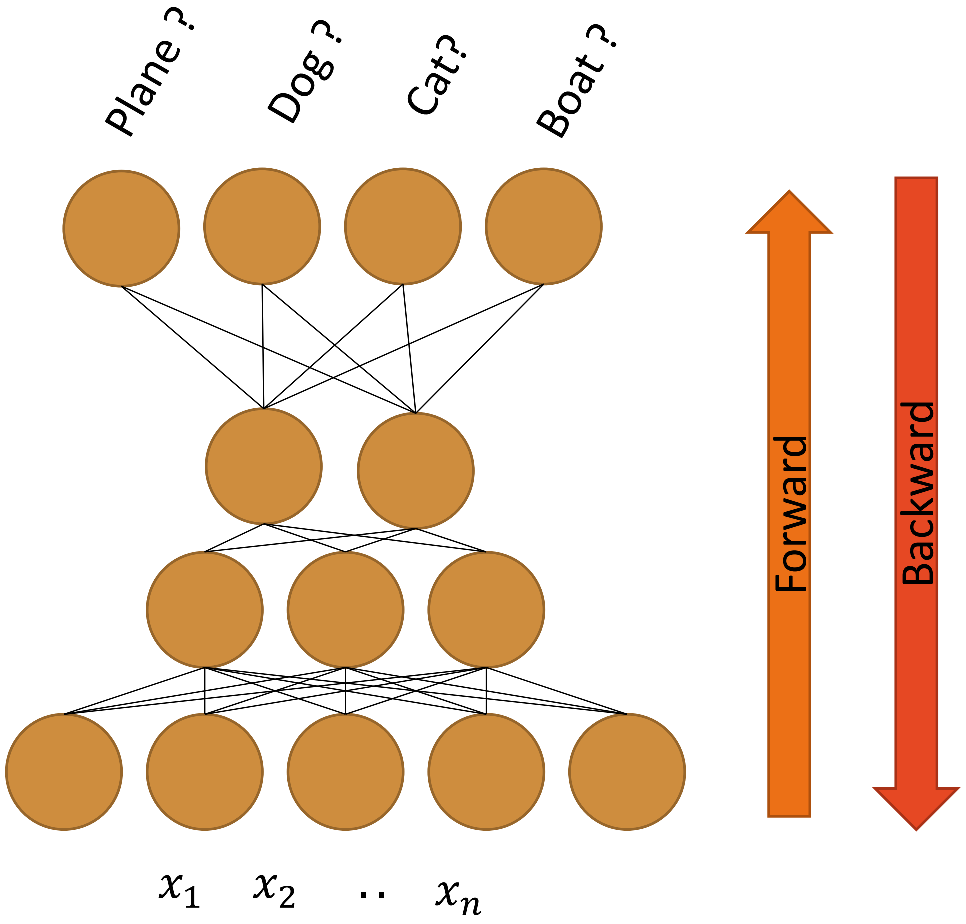
Stacked Autoencoders:



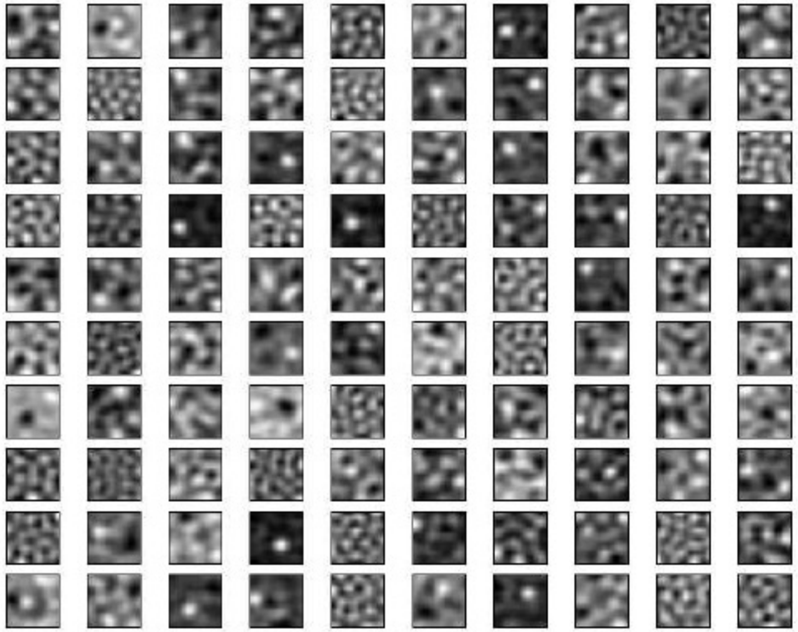
Initialization



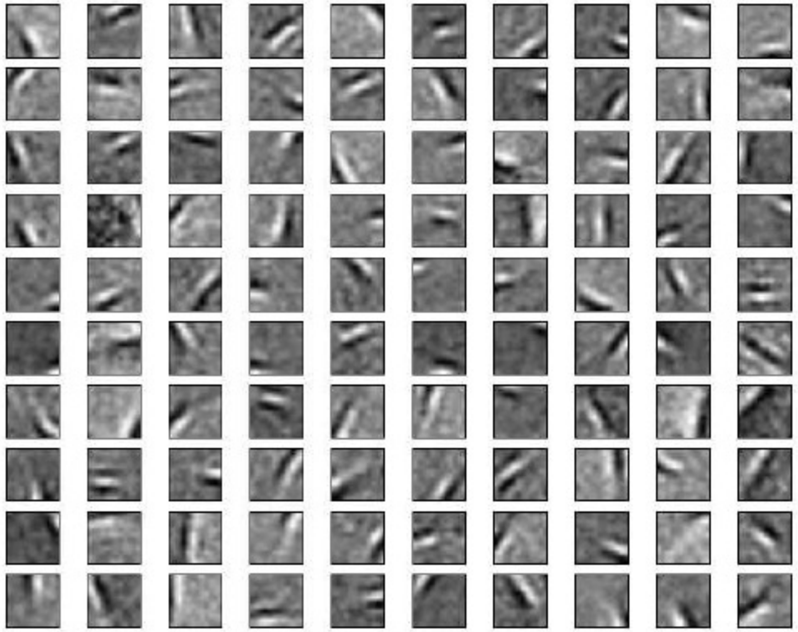
Initialization



Initialization



Input neuron activation

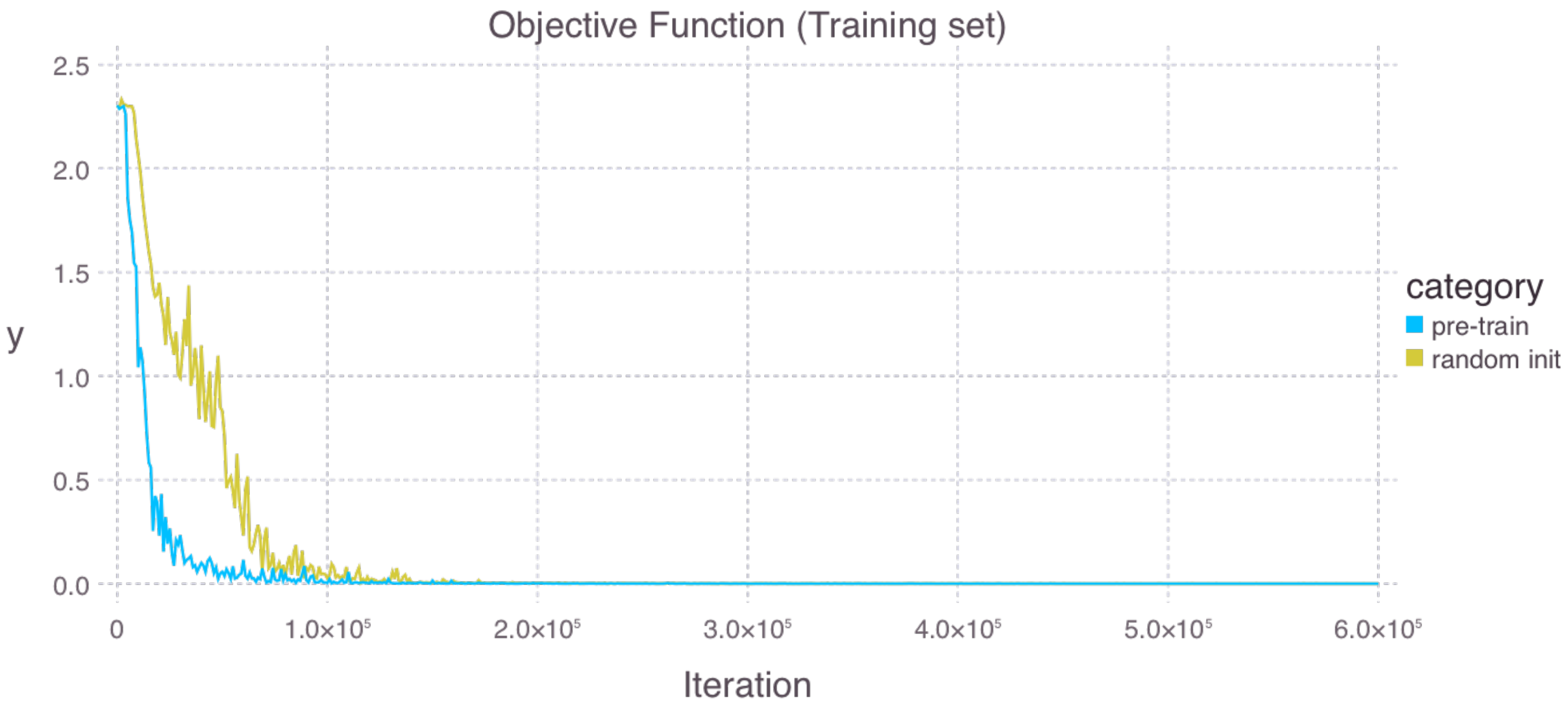


Input neuron activation after stacked autoencoders

<u>Source :</u> Pascal Vincent et al, *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*, 2008

Initialization

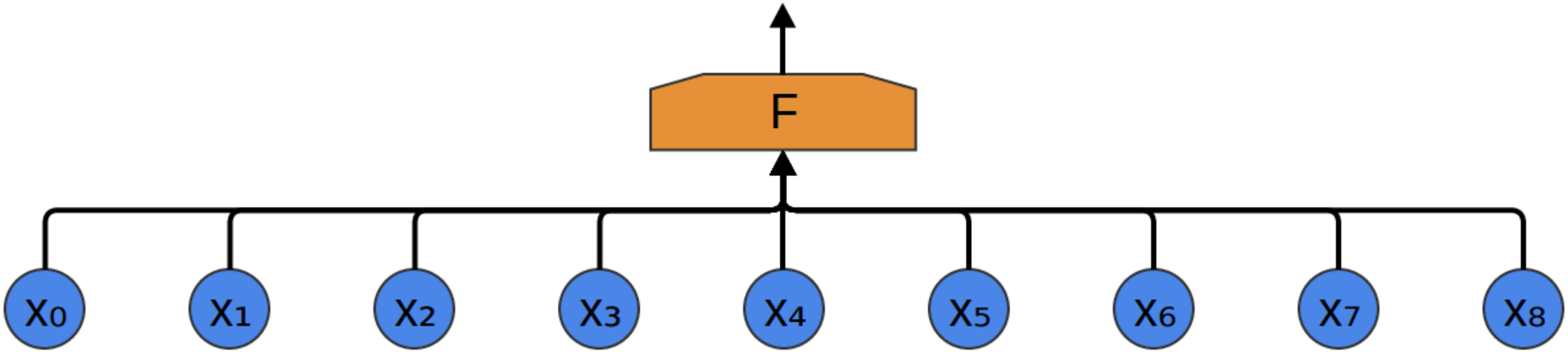
Does initialization can improve the final score?



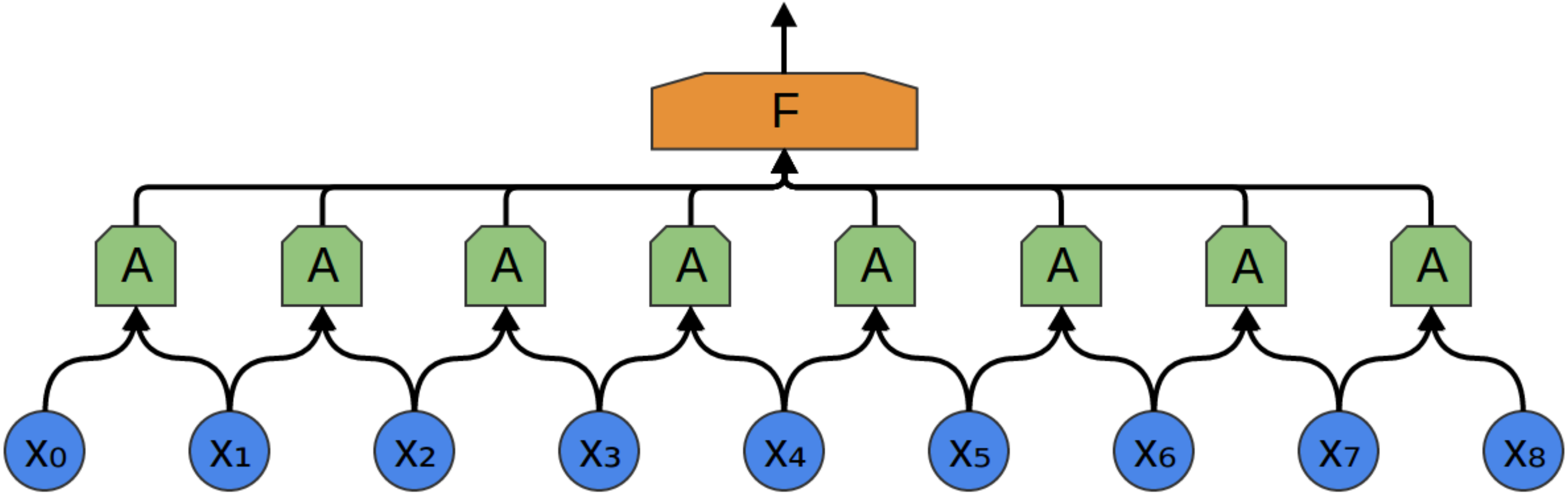
Source NVIDIA – CIFAR 10

Convolutional networks

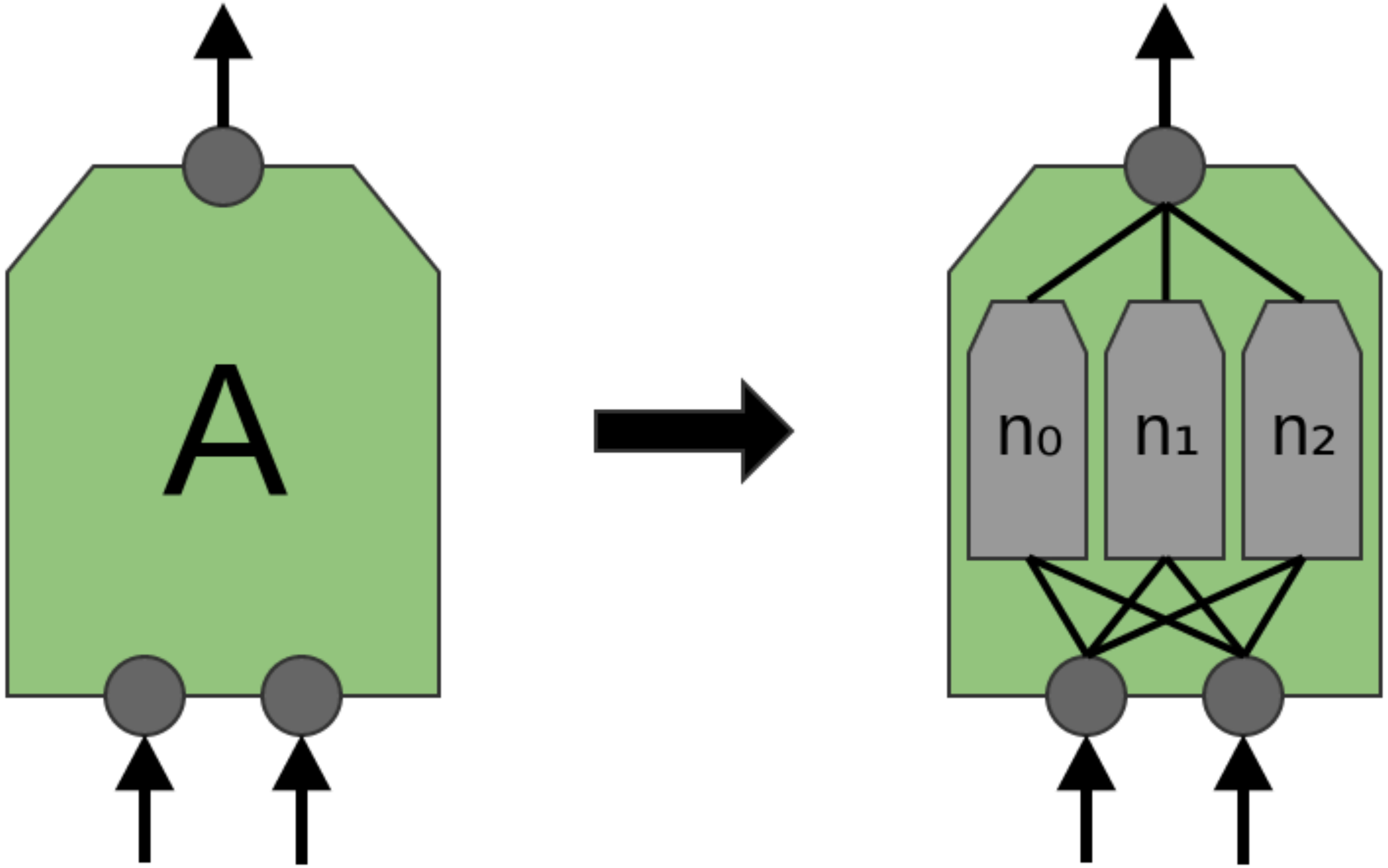
Source : http://colah.github.io/posts/2014-07-Conv-Nets-Modular/



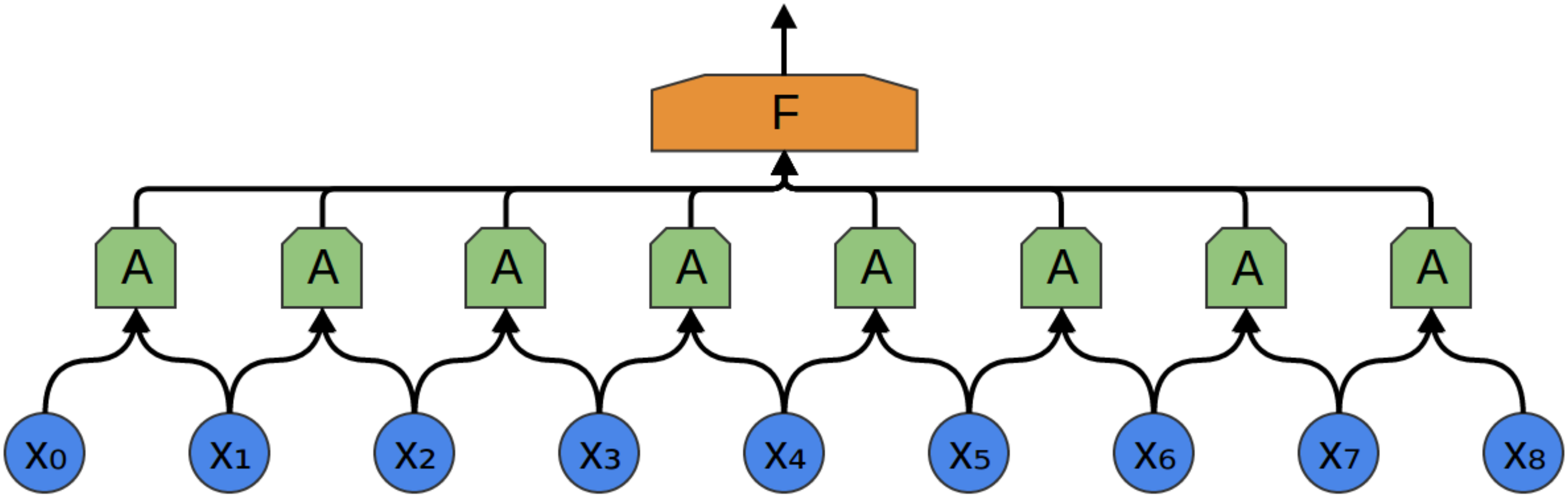
Convolutional networks



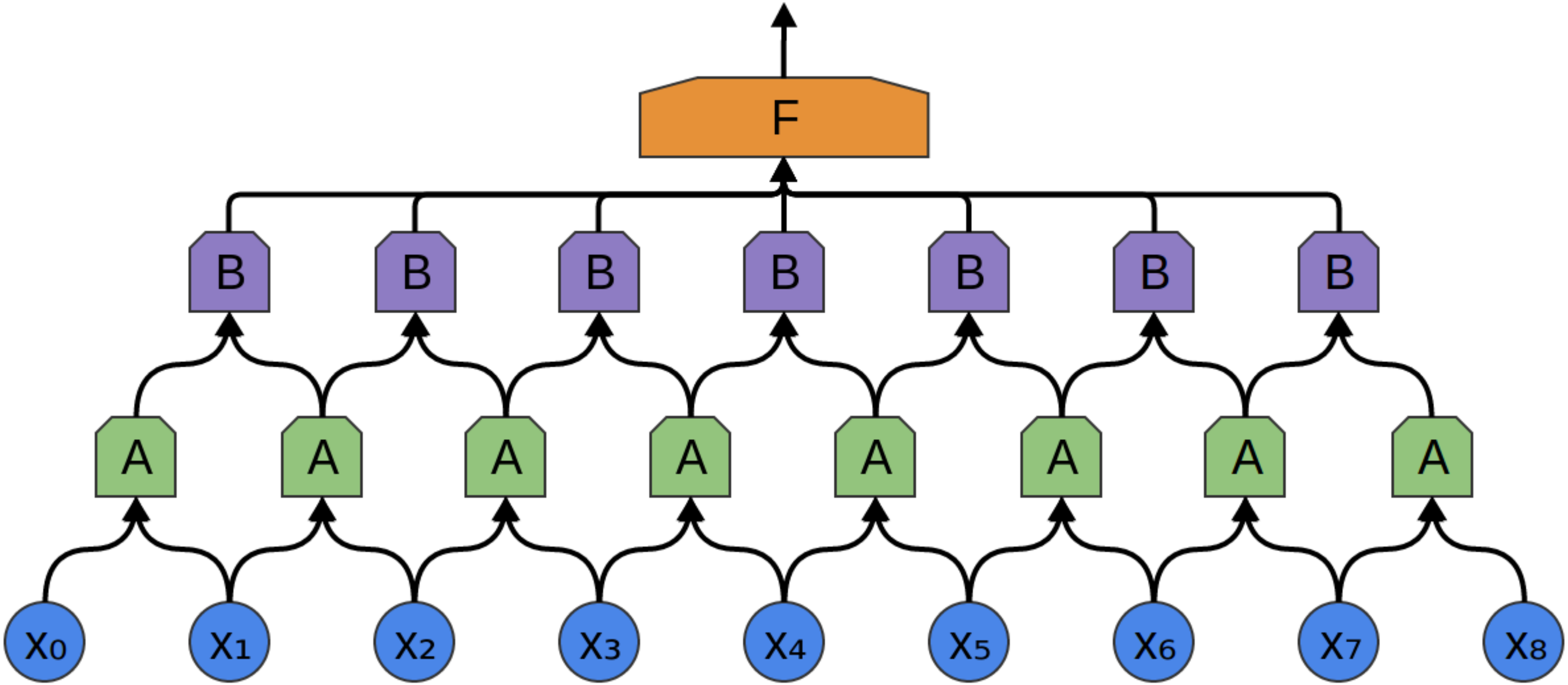
Convolutional networks



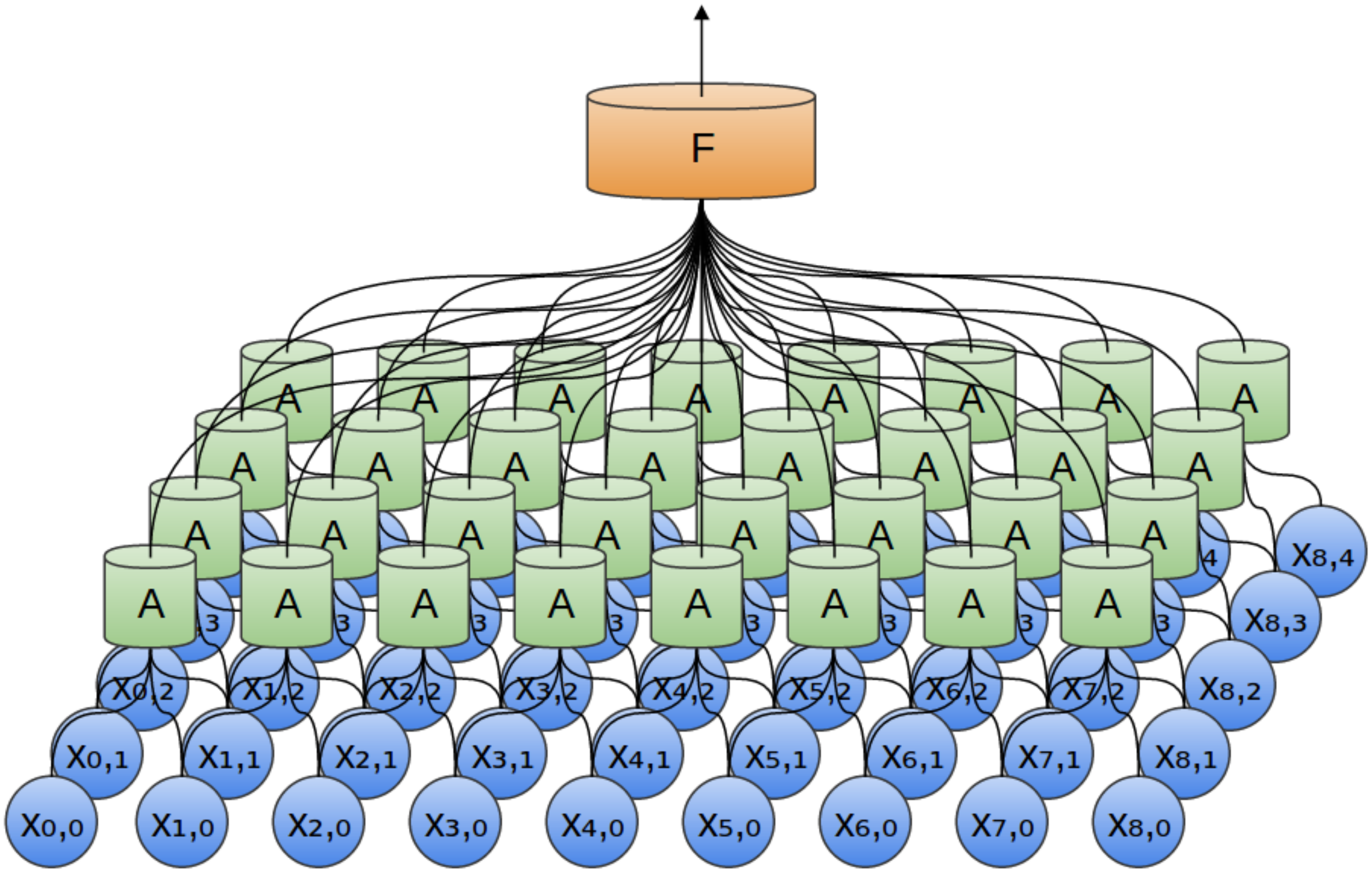
Convolutional networks



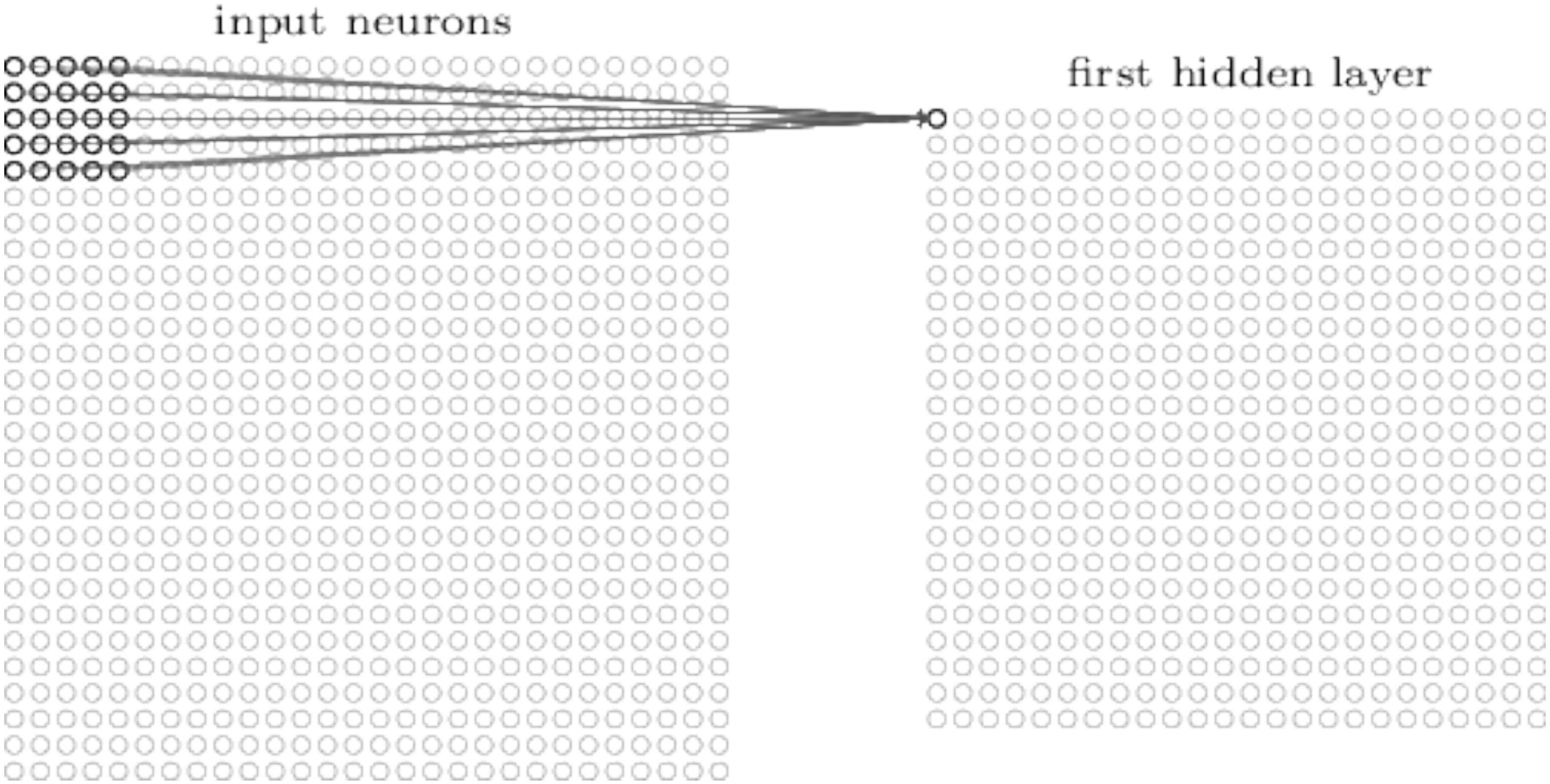
Convolutional networks



Convolutional networks

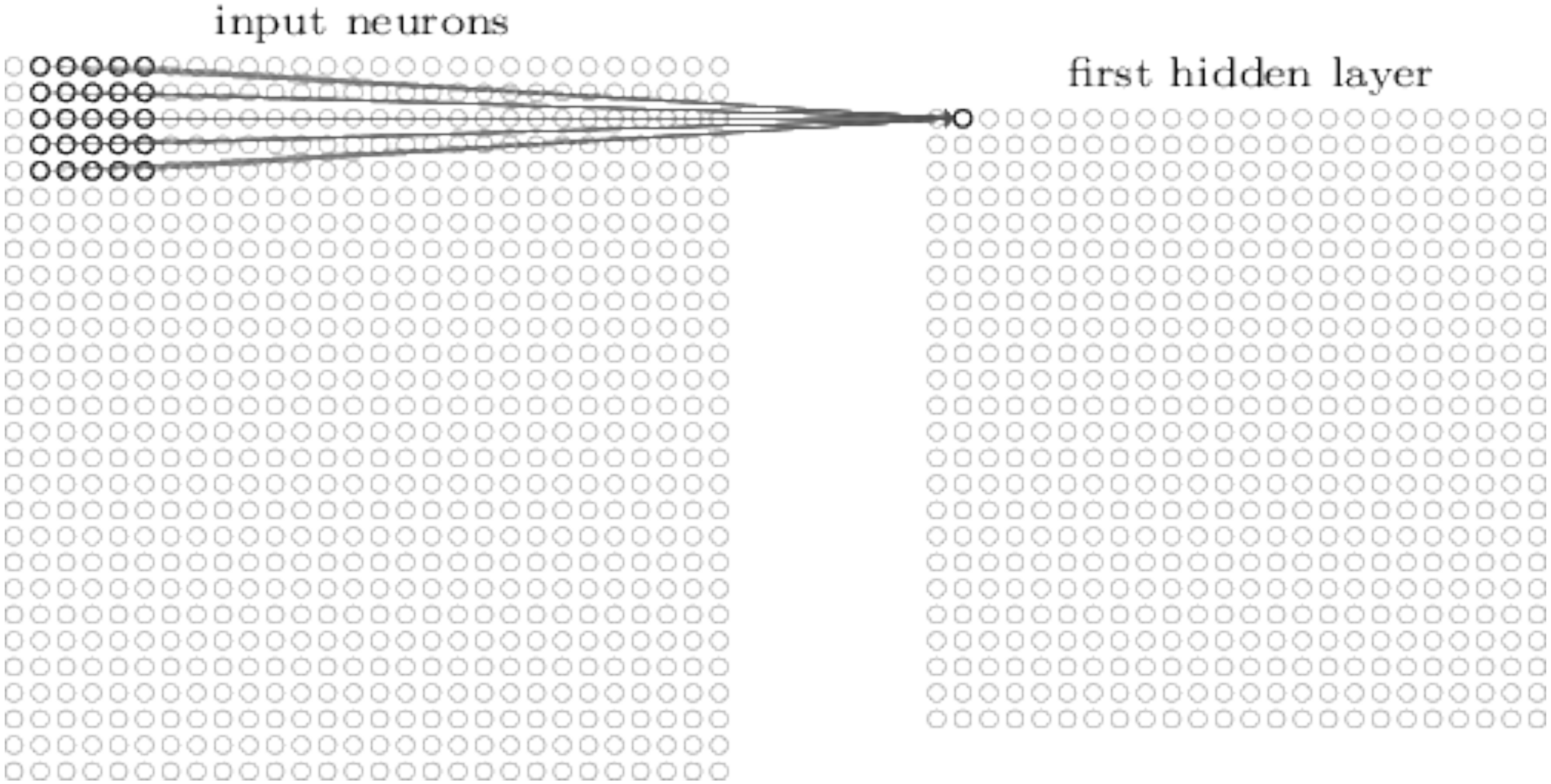


Convolutional networks

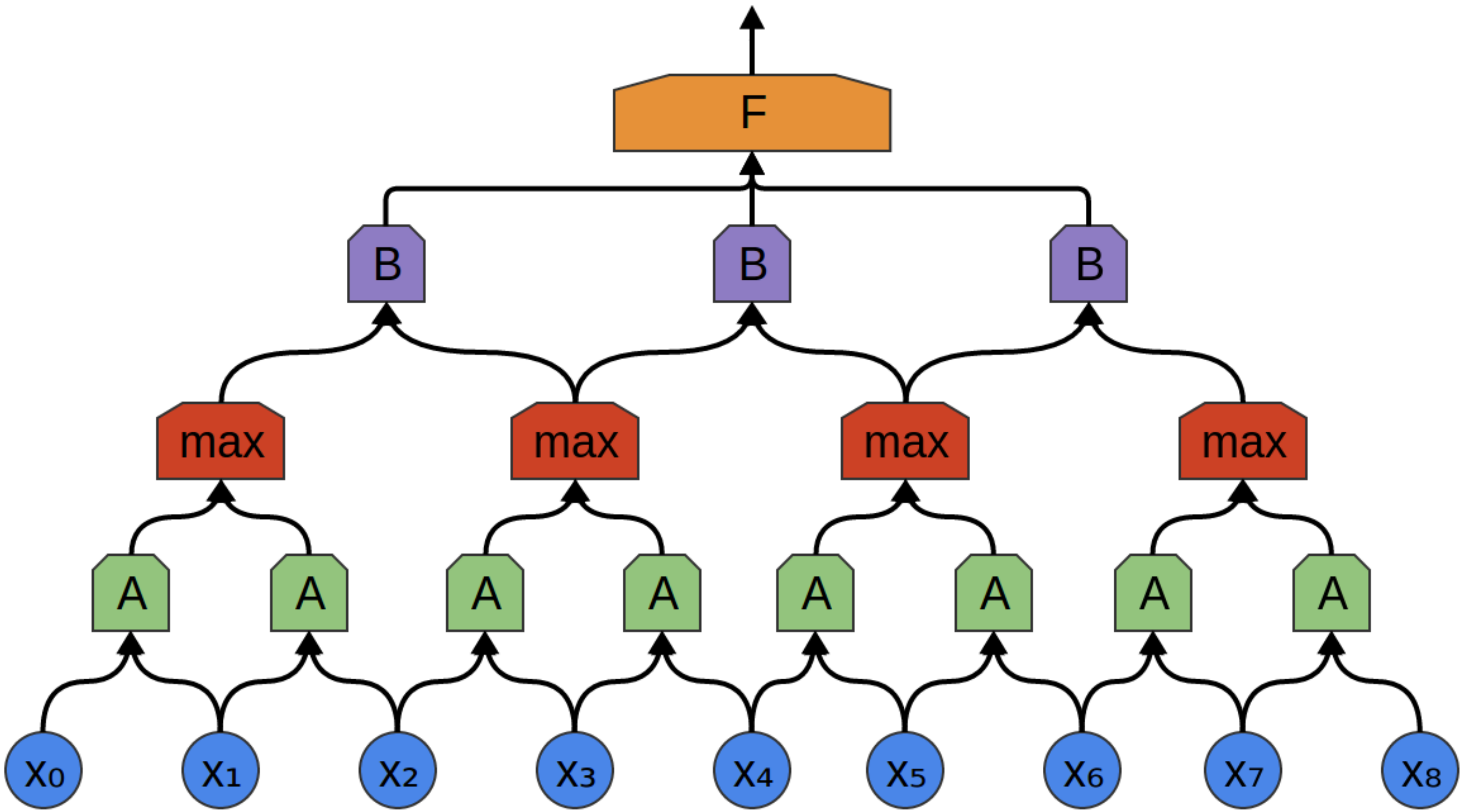


Source : http://neuralnetworksanddeeplearning.com/chap6.html

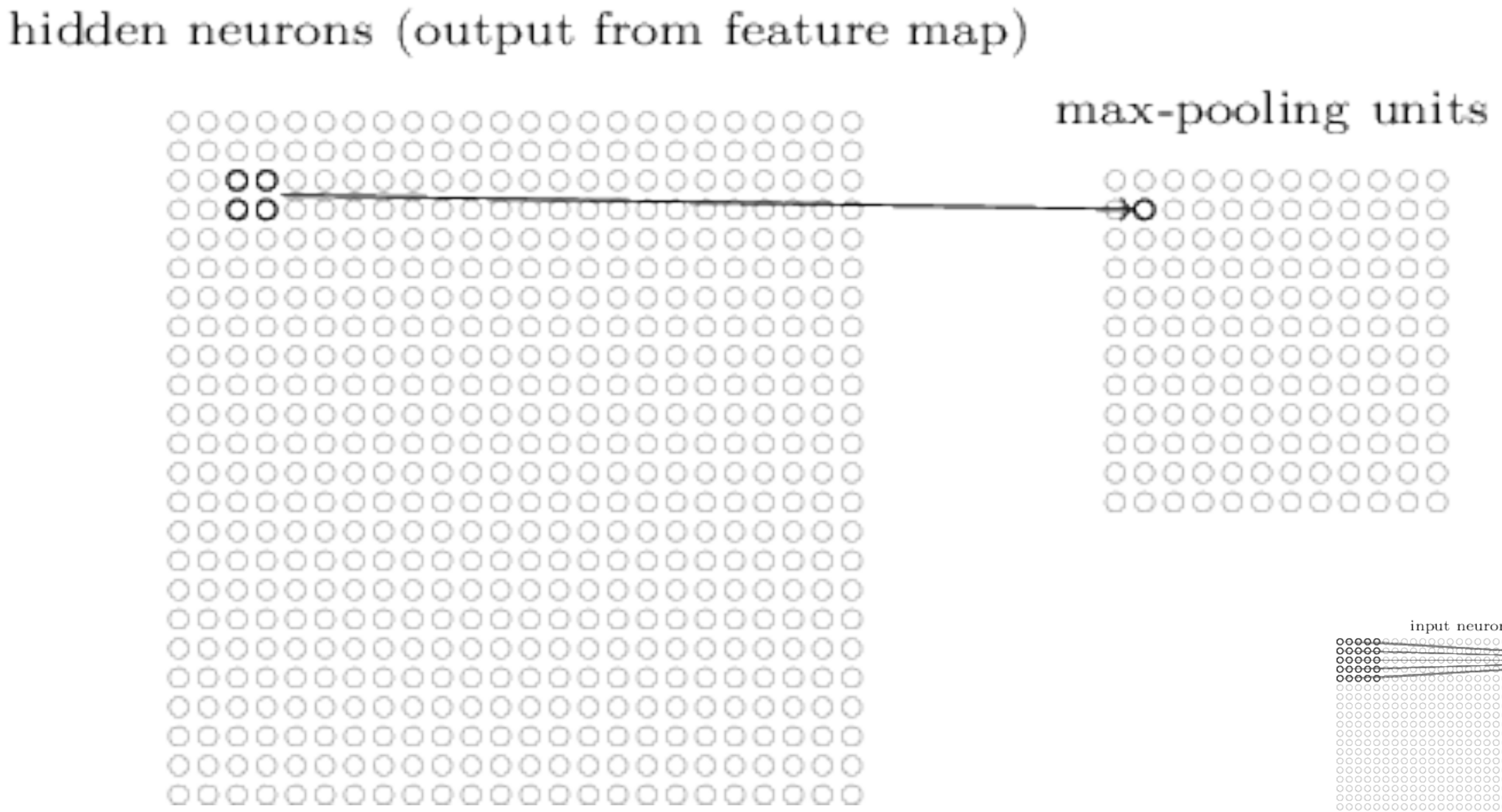
Convolutional networks

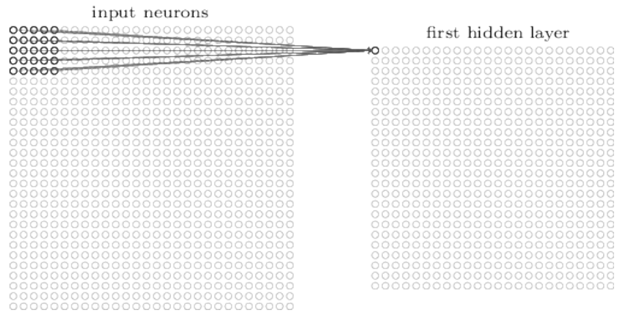


Convolutional networks

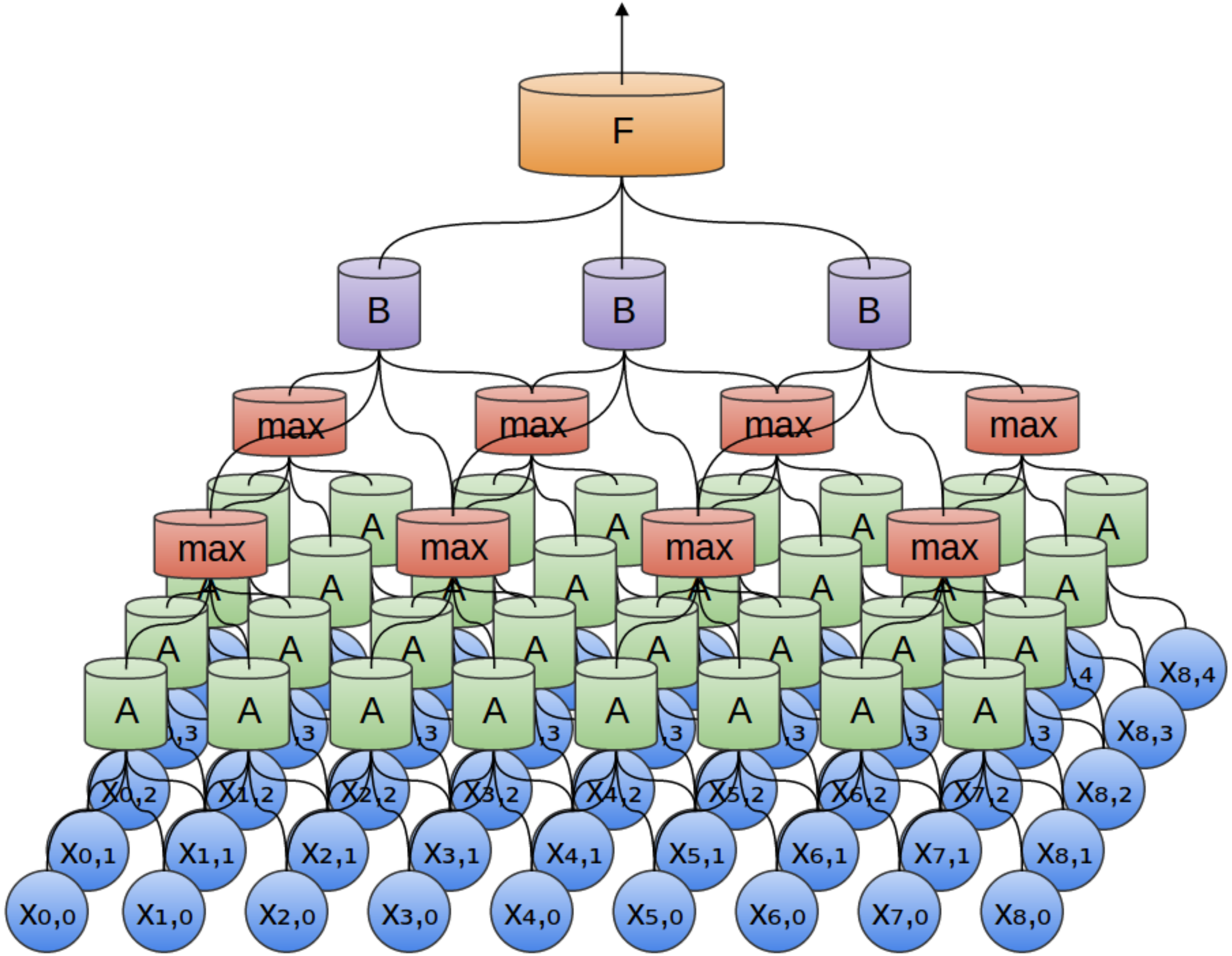


Convolutional networks

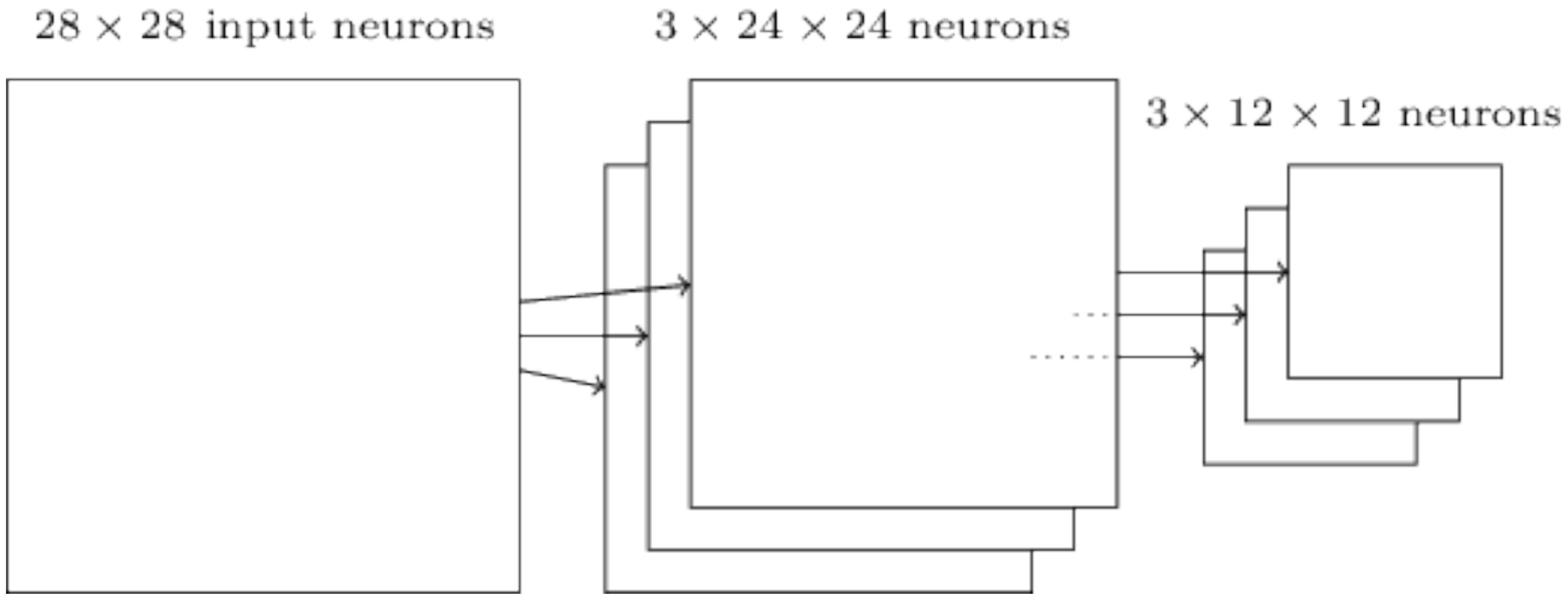


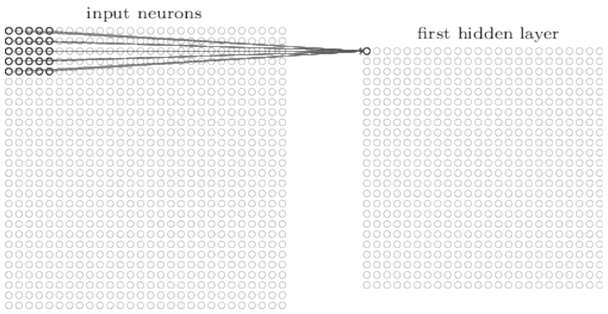


Convolutional networks



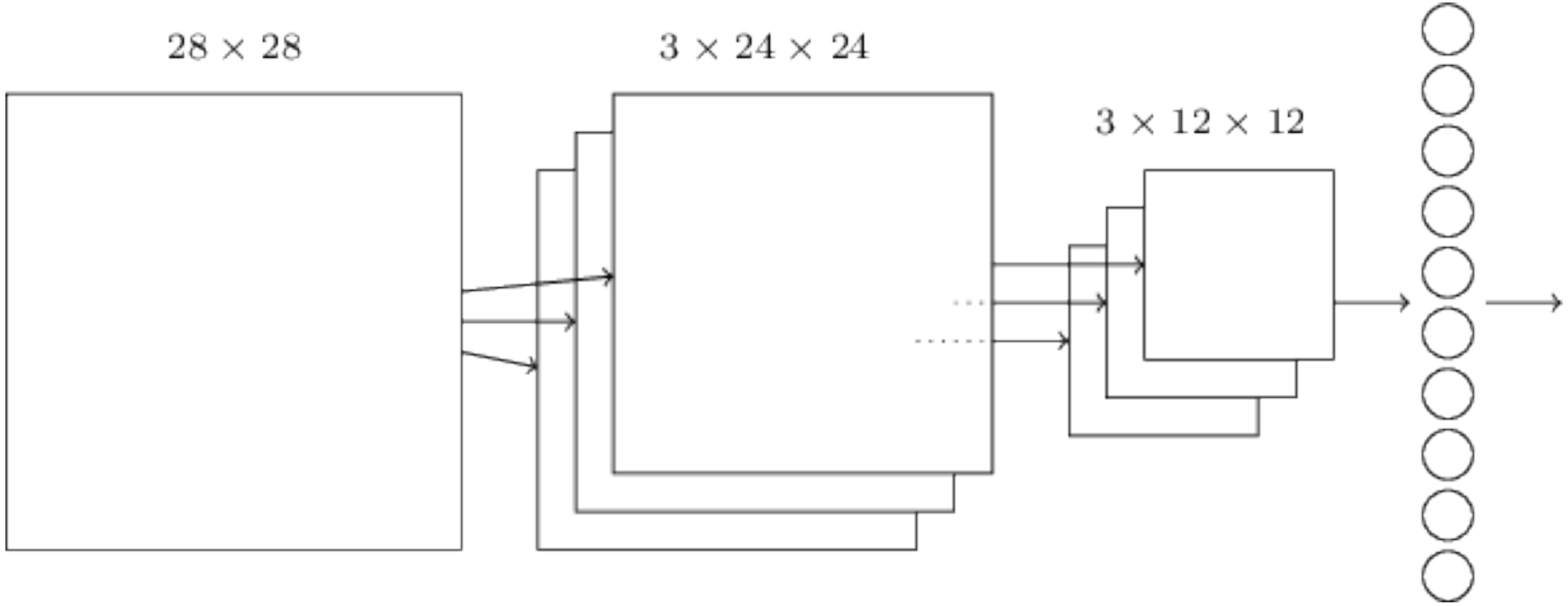
Convolutional networks



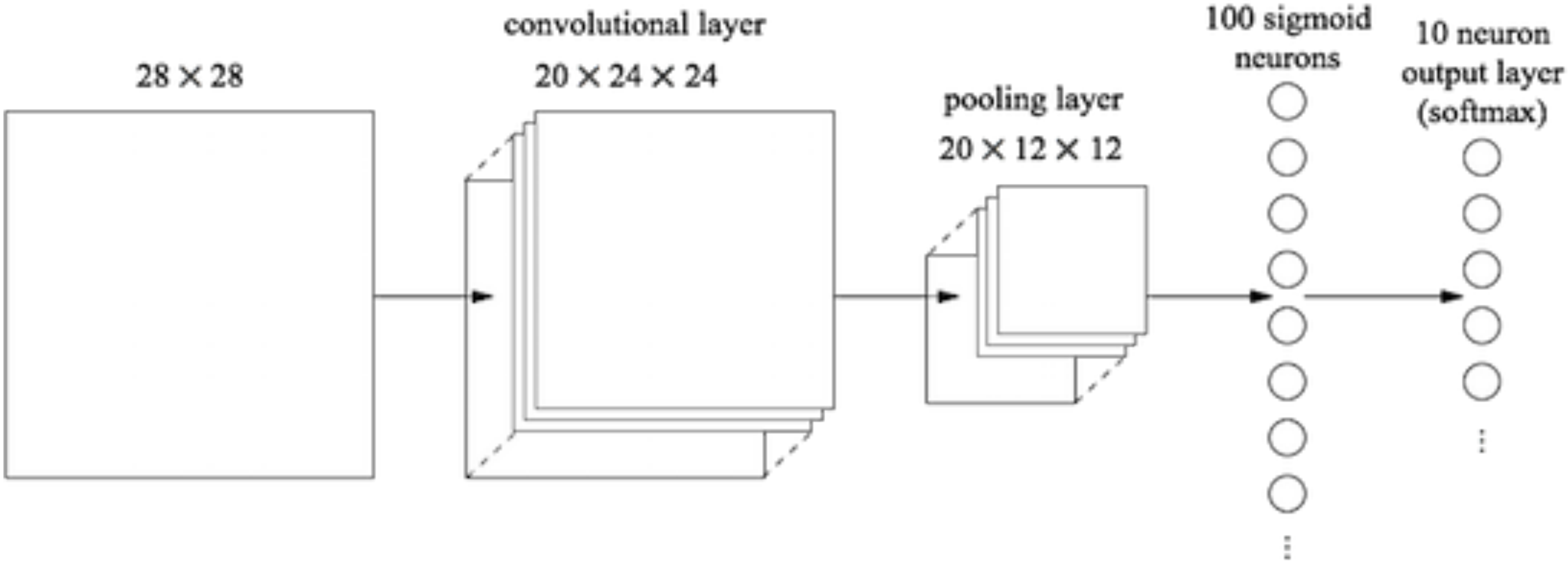


x 3

Convolutional networks

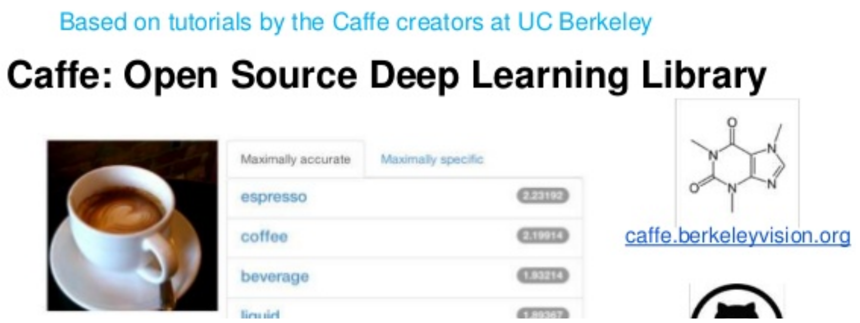


Convolutional networks

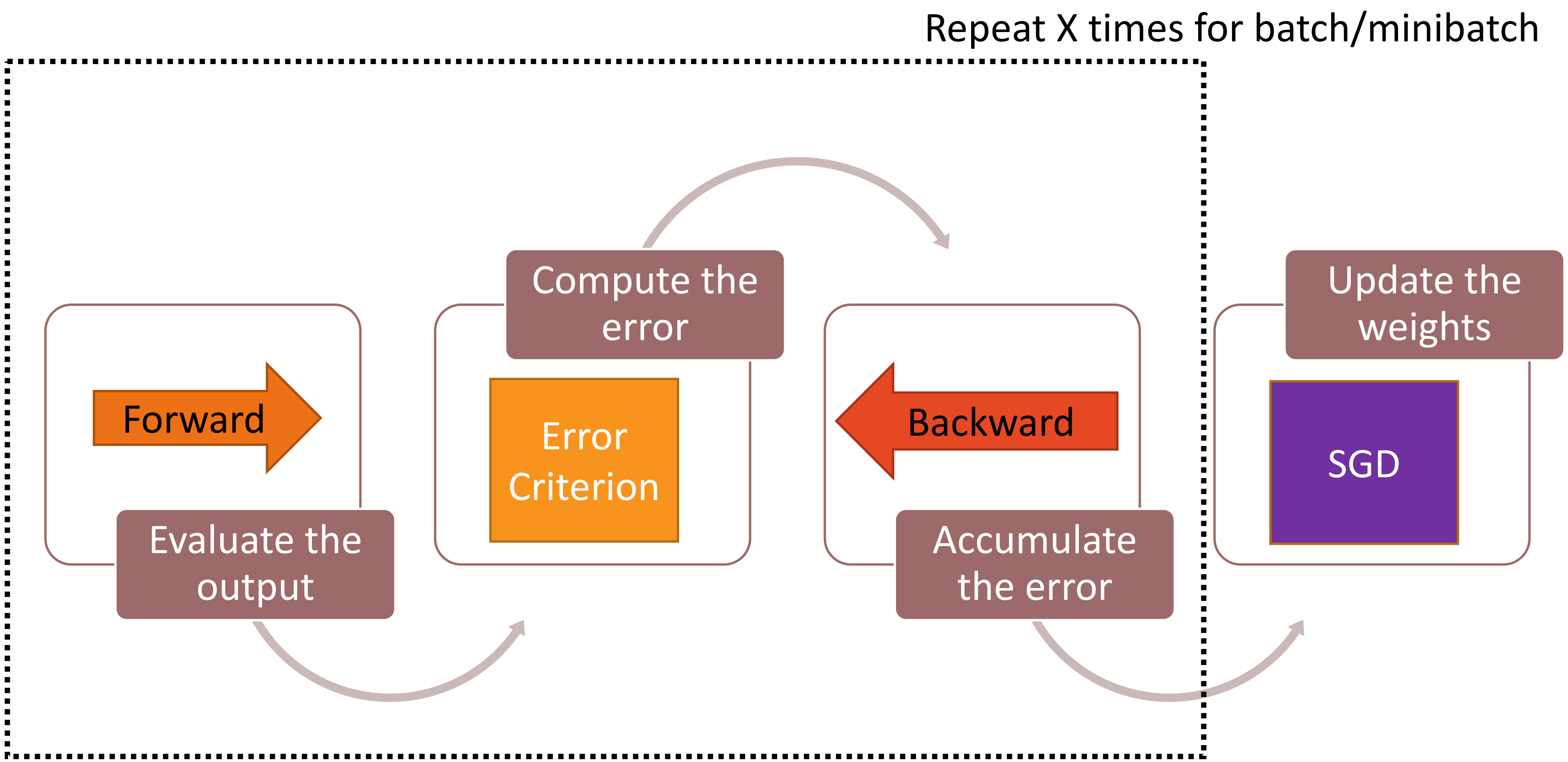


Implementation

Frameworks	Language	Developed by	Paradigm	Suitable for
Thenao	Python	Montreal laboratory	Lambda calculus	Academic
Torch	Lua	Facebook / Deepmind	Object Oriented	Academic
Tensor Flow	C++ / python	Google	Lambda calculus	Industry
Caffe	n/a	Berkeley	Script	Industry



Implementation



Backprogagation of the gradient

$$\frac{\delta E}{\delta w_{kj}^{(2)}} = \frac{\delta E}{\delta a_k^{(2)}} \frac{\delta a_k^{(2)}}{\delta w_{kj}^{(2)}} = \delta_k^{(2)}\, z_j^{(1)}$$

$$a_{\mathbf{k}}^{(2)} = y_k = \sum_j w_{kj}^{(2)} h(a_{\mathbf{j}}^{(1)})$$

$$\delta_k^{(2)} = \frac{\delta E}{\delta a_k^{(2)}} = (y_k - t_k)$$

$$z_j^{(1)} = h(a_{\mathbf{j}}^{(1)})$$

$$E = \sum_k \frac{1}{2} (y_k - t_k)^2$$

$$a_{\mathbf{k}}^{(2)} = \sum_j w_{kj}^{(2)} h(a_{\mathbf{j}}^{(1)})$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = \frac{\delta E}{\delta a_j^{(1)}} \frac{\delta a_j^{(1)}}{\delta w_{ji}^{(1)}} = \delta_j^{(1)}\, z_i^{(0)}$$

$$\delta_j^{(1)} = \frac{\delta E}{\delta a_j^{(1)}} = \sum\nolimits_k \frac{\delta E}{\delta a_k^{(2)}} \frac{\delta a_k^{(2)}}{\delta a_j^{(1)}}$$

$$\frac{\delta a_k^{(2)}}{\delta a_j^{(1)}} = w_{kj}^{(2)} h'\left(a_{\mathbf{j}}^{(1)}\right)$$

$$\delta_j^{(1)} = h'\left(a_{\mathrm{j}}^{(1)}\right) \sum_k w_{kj}^{(2)} \delta_k^{(2)}$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = (1 - h^2(a_{\mathbf{j}}^{(1)})) \sum_k w_{kj}^{(2)} (y_k - t_k) \quad x_i$$

$$\frac{\delta E}{\delta w_{ji}^{(1)}} = \delta_j^{(1)}\, z_i^{(0)}$$

NB : $if\ h(x) = \tanh(x)\ \ then\ h'(x)$ = 1 - $h(x)^2$