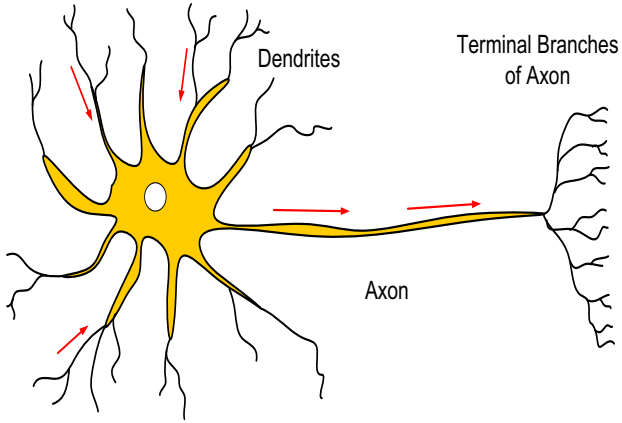
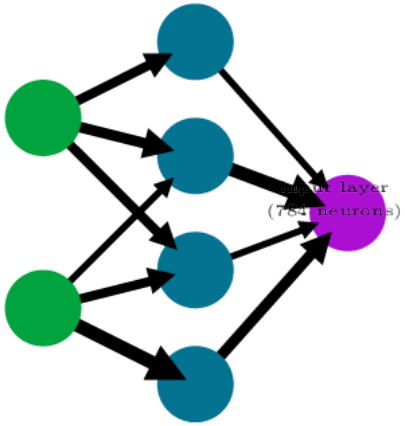
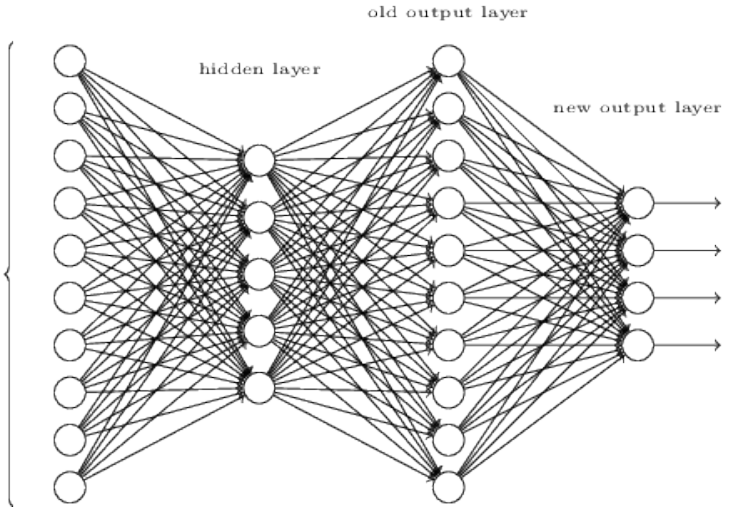


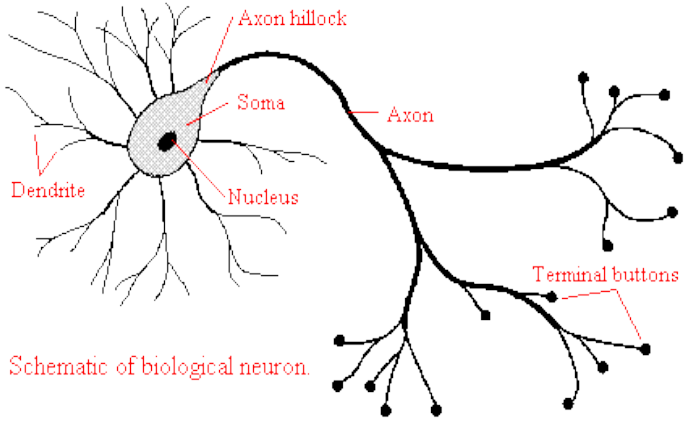


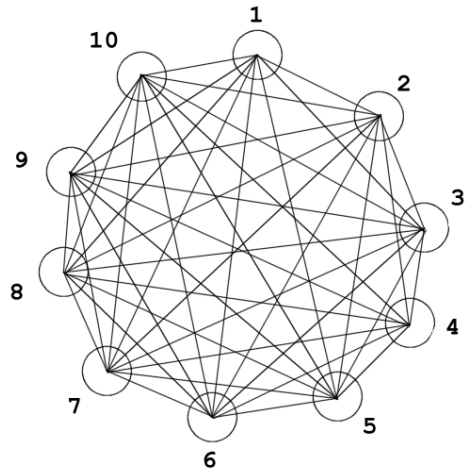


Artificial Neural Network

Anna Helena Reali Costa

PCS





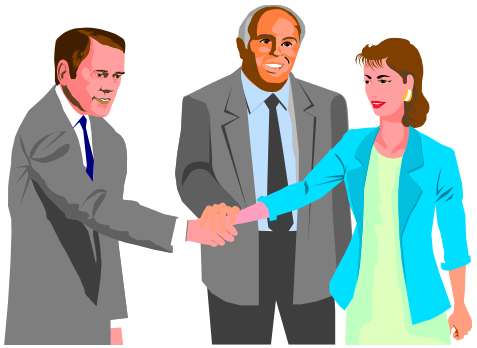
ANN

- A different style of computation: **parallel distributed processing**
- A **universal computational architecture**: the same structure carries out many different functions
- It can learn new knowledge, therefore it is **adaptive**

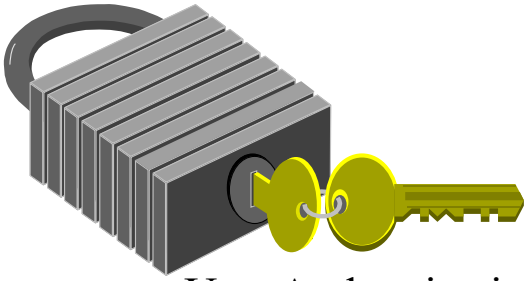
History

- McCulloch and Pitts introduced the artificial neuron in 1943.
  - Simplified model of a biological neuron
- Fell out of favor in the late 1960's
  - Perceptron limitations (Minsky and Papert)
- Resurgence in the mid 1980's
  - Nonlinear Neuron Functions
  - Backpropagation training (Werbos)
- Currently resounding success with Deep NN

Applications



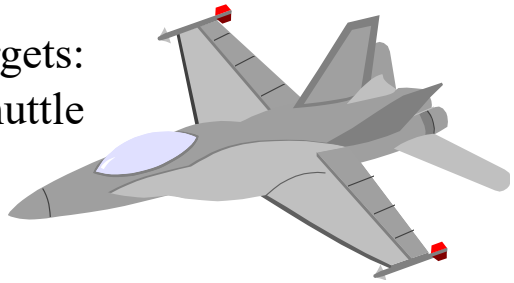
Face / Speech Recognition

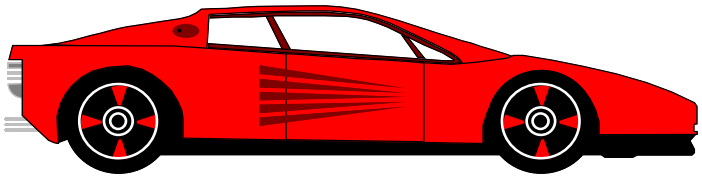


User Authentication.

Oil exploration: Lithology, etc.

Identification of military targets: B-52, Boeing 747, Space Shuttle





Autonomous Vehicle Navigation

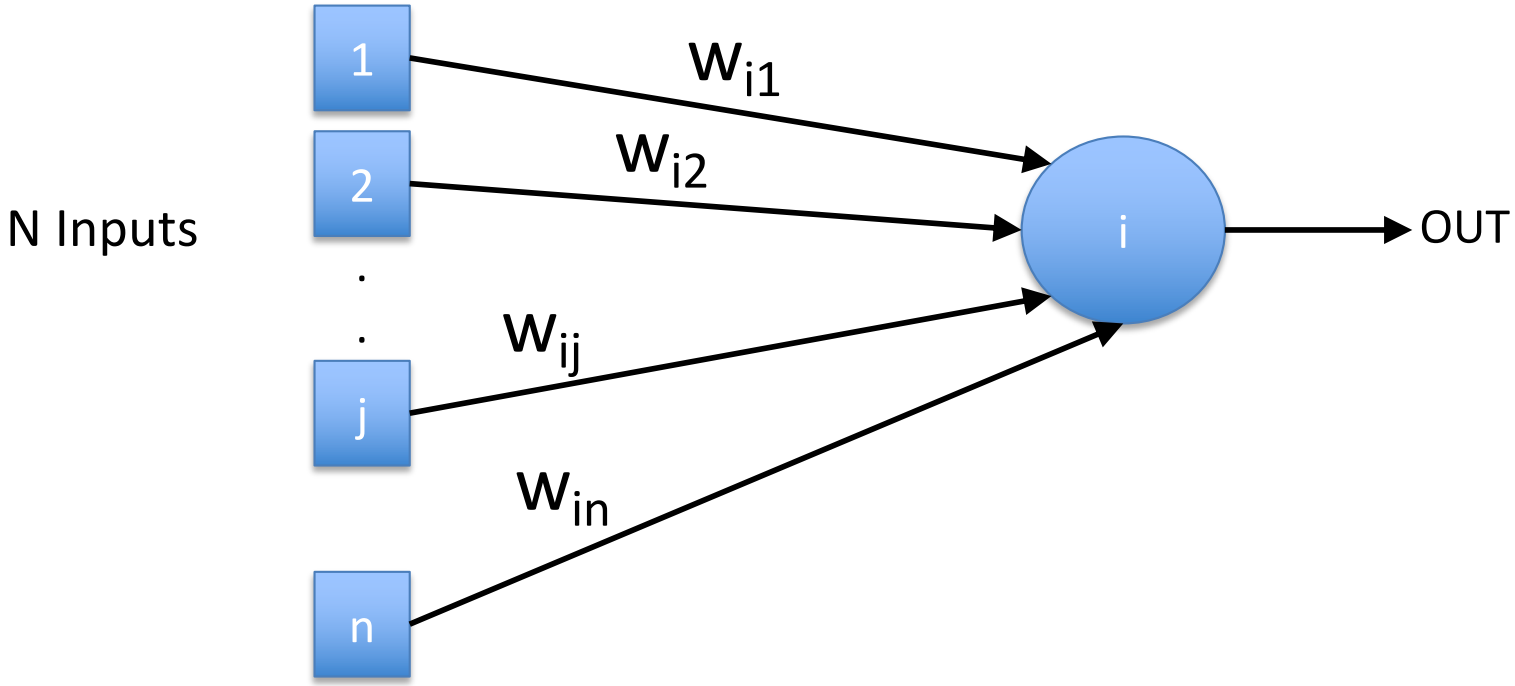
Prediction on Financial Market

Design of an ANN

- The design of an ANN involves determining the following elements:
  - Neurons and **activation function**.
  - Connections and arrangement of neurons: **topology** (network architecture).
  - Synaptic **weights**: values (in the case of learned weights) or a **training algorithm** to be used and its parameters.
  - **Recall**: procedure to be used for the network to calculate the outputs for given new inputs

- Unfortunately, there is no single "recipe" ...

Perceptron (Frank Rosenblatt, 1958)

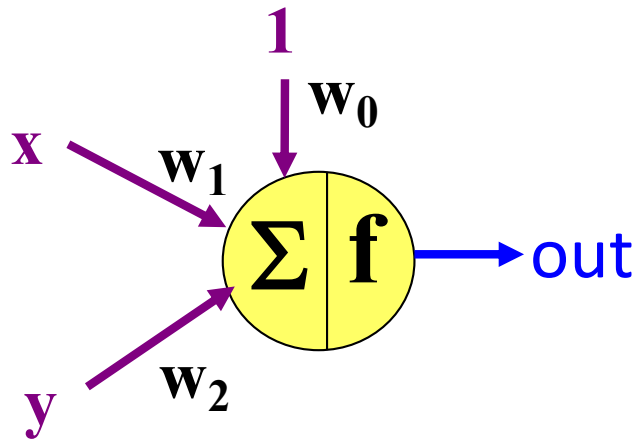


$$OUT = f(net) = \begin{cases} 1 : net > 0 \\ -1 : net \leq 0 \end{cases} \quad with \;\; net = \sum_j w_{ij} In_j$$

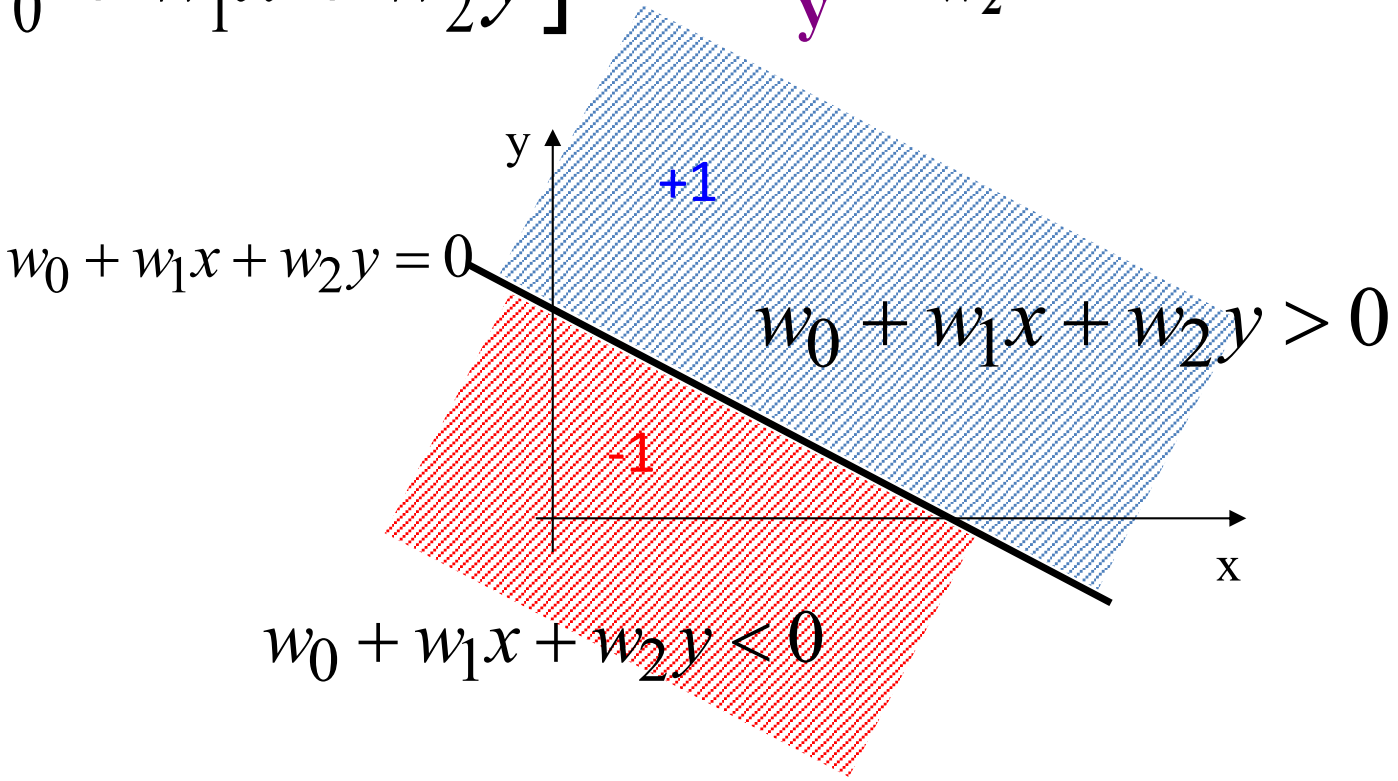
Activation function

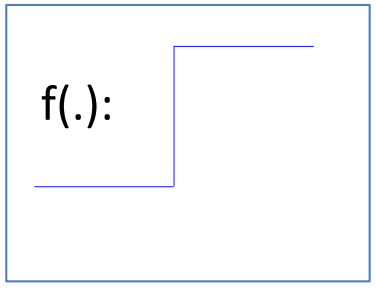
Perceptron: example

- Two inputs: x, y
- Two weights: w_1, w_2
- **Bias** input: w_0



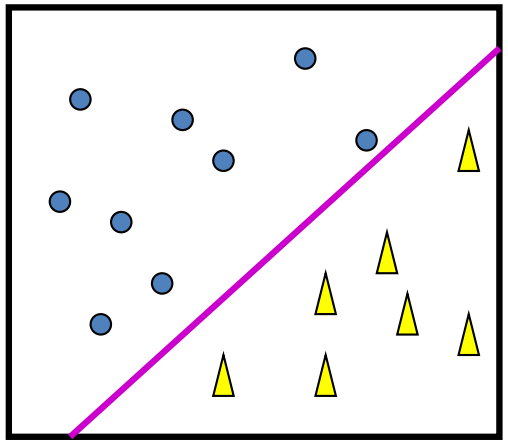
$$out = f\left[w_0 + w_1 x + w_2 y\right]$$





Representational power of Perceptrons

Perceptron can represent any **linearly separable** function.

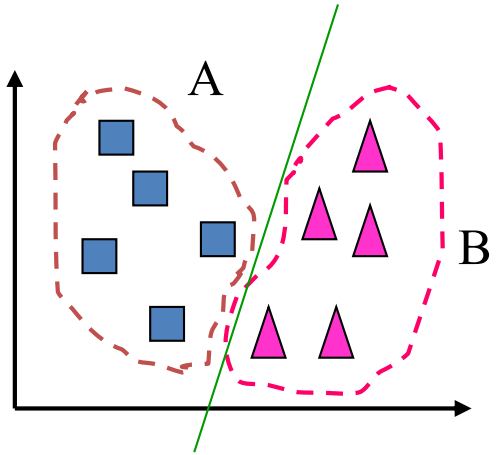


It represents a hyperplane decision surface in the n-dimensional space of instances.

1 neuron: 2 classes +1 and –1

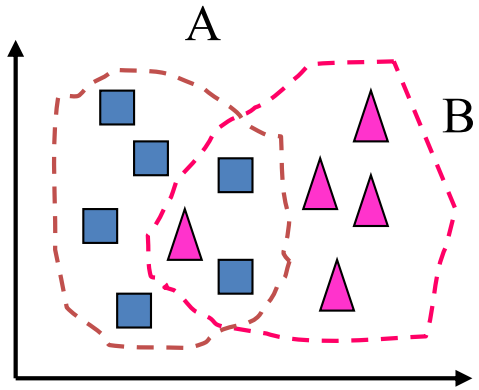
2 neurons: 4 classes +1+1, +1-1, -1+1, -1-1

N neurons: 2^N classes



← OK

Not OK →



Exercise

- Design a perceptron (Figure 1) which calculates the logical implication function $y = x_1 \rightarrow x_2$, described in Table 1.

Table 1

x_1	x_2	y
0	0	1
0	1	1
1	0	0
1	1	1

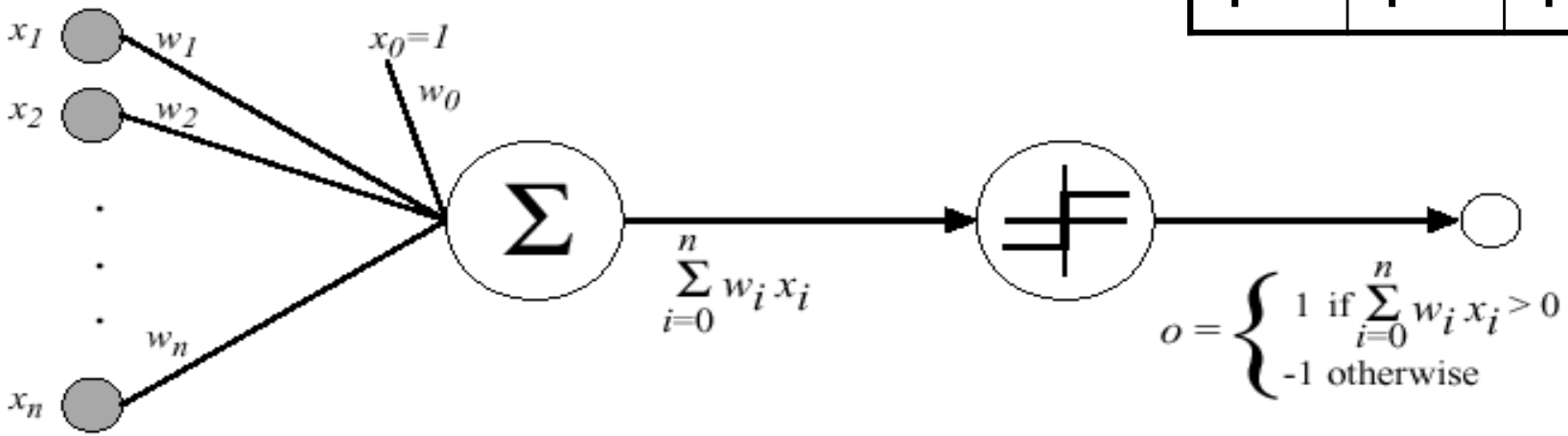


Figure 1 – Perceptron.

Exercise

- Design a perceptron (Figure 1) which calculates the logical implication function $y = x_1 \rightarrow x_2$, described in Table 1.

Table 1

x_1	x_2	y
0	0	1
0	1	1
1	0	0
1	1	1

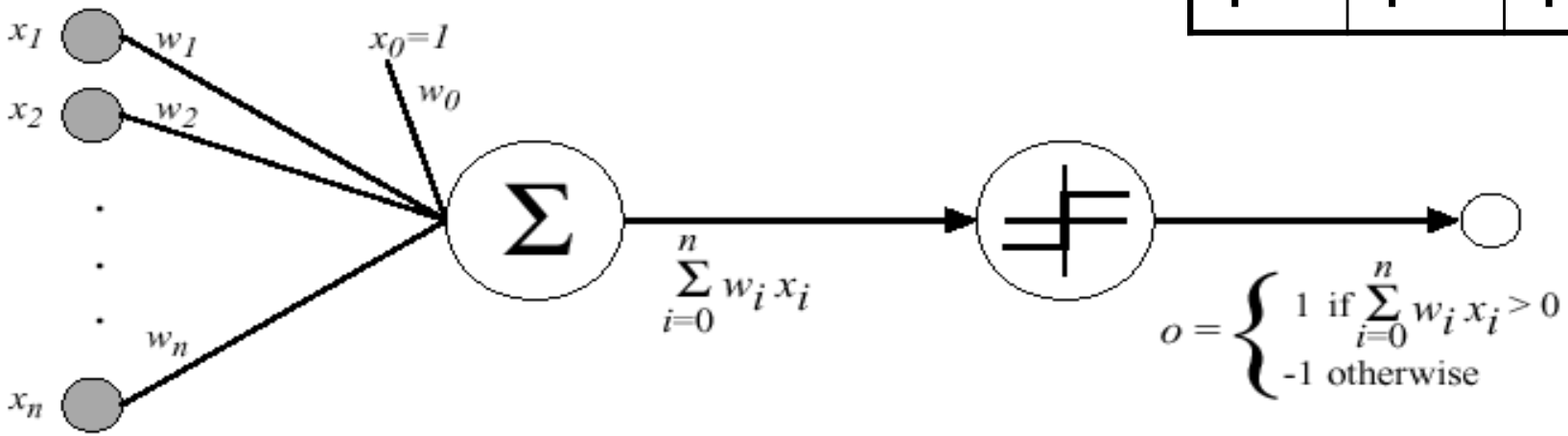


Figure 1 – Perceptron.

Uma resposta: $w_0 = 1$, $w_1 = -2$, $w_2 = 2$
(0 representa NL0, 1 representa NL1)

The Perceptron training rule

Delta Rule

- Let ***t*** be the target output for the current training example, ***o*** be the output generated by the perceptron, and **η** be a positive constant called the learning rate:

Delta Rule:

$$w_i \leftarrow w_i + \Delta w_i$$

$$and$$

$$\Delta w_i = \eta (t - o) x_i$$

The Perceptron training algorithm

- Initialize weight matrix **w**
- For each training sample i, with $(\mathbf{x}_i, s(\mathbf{x}_i)) = (\mathbf{x}_i, \mathbf{t}_i)$:
  - Use current **w** to calculate $s'(\mathbf{x}_i) = \mathbf{o}_i$
  - If $|\mathbf{t}_i - \mathbf{o}_i| > \varepsilon$ then update:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta.(\mathbf{t}_i - \mathbf{o}_i).\mathbf{x}_i$$

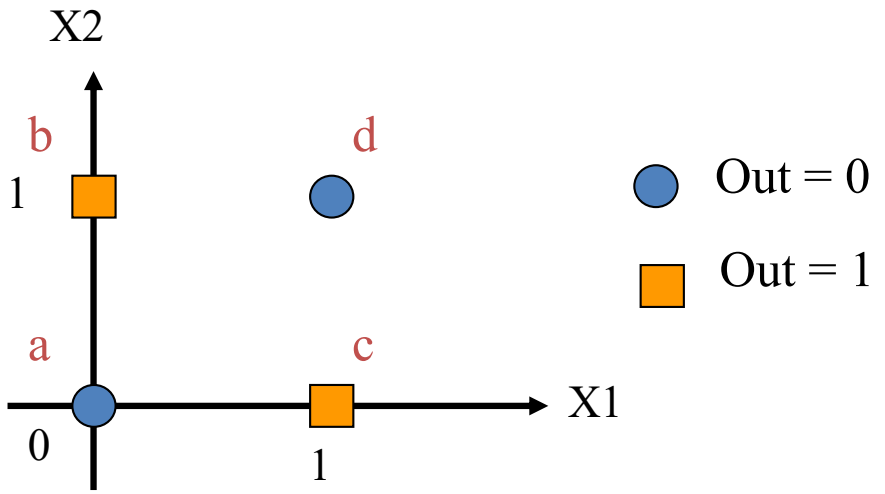
- Stop when $\varepsilon \geq |\mathbf{t}_i - \mathbf{o}_i|$ for all samples $(\mathbf{x}_i, \mathbf{t}_i)$

If $|ti - oi| = |\delta| \leq \varepsilon$ ➔ **w** does not change
If $\delta > 0$ and $|\delta| > \varepsilon$ ➔ **w** increases (because o_i is very small)
If $\delta < 0$ and $|\delta| > \varepsilon$ ➔ **w** decreases (because o_i is very large)

Limitations of Perceptron with a single layer

- Unfortunately, various functions of interest are not linearly separable.
- For example, the Perceptron can not represent XOR (exclusive OR).

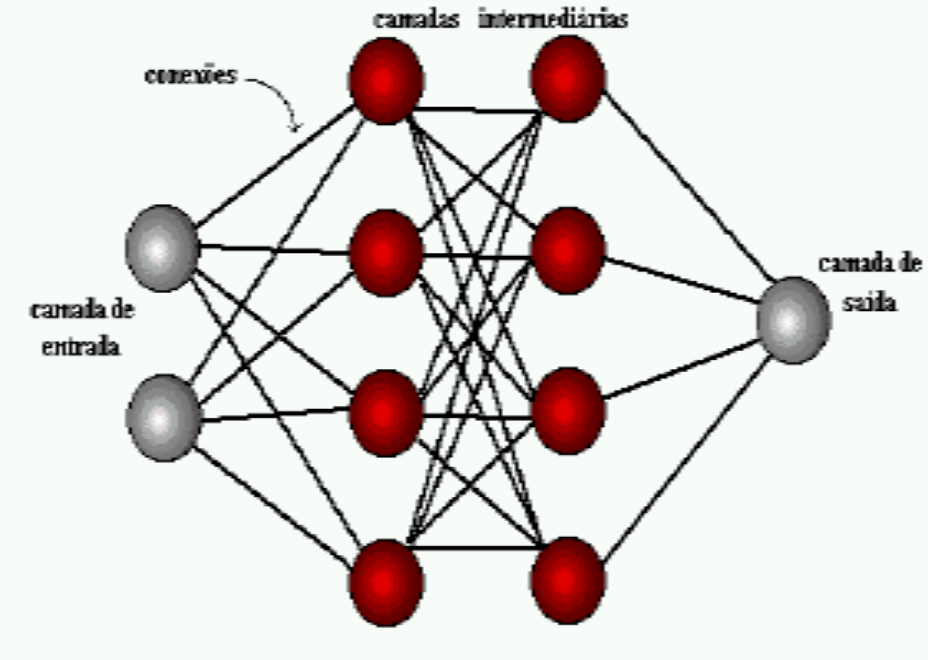
	X1	X2	XOR
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	0



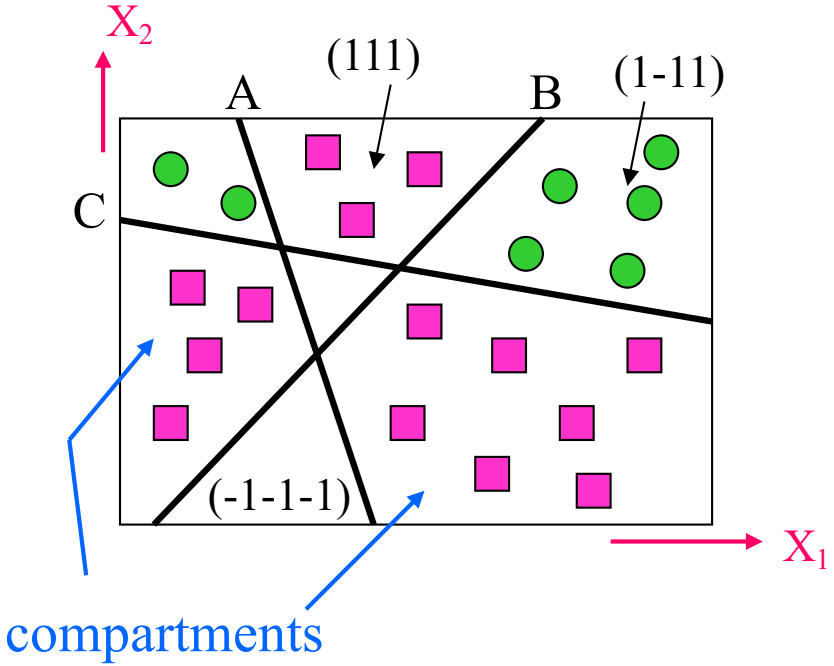
MLP: Multilayer Perceptrons

- Sets of perceptrons arranged in several layers.
- At least one hidden layer.

- An intermediate layer is sufficient to approximate any continuous function.
- Two intermediate layers are sufficient to approximate any mathematical function.

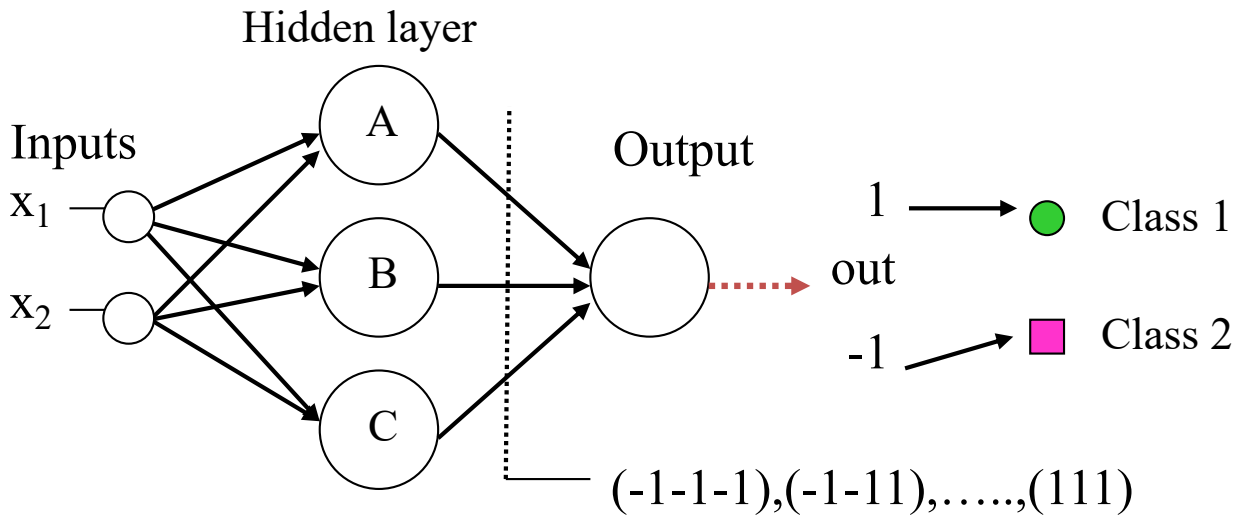


MLP: Multi-layer Perceptrons



Classification requires 3 straight lines that create 7 compartments and two decision regions: one for ■ and one for ●

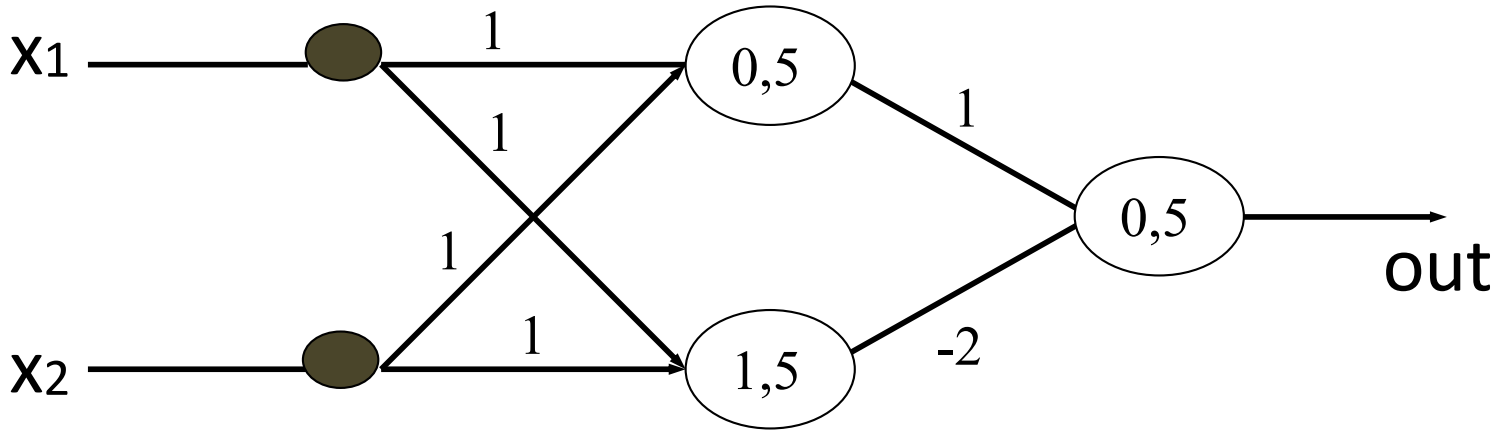
Solution:



Example of an MLP

- $x_1 = x_2$ = binary inputs.
- $w_1 = w_2 = w_3 = w_4 = w_5 = 1$ e $w_6 = -2$
- f_1 (**x**.**w**) = 1 if the activation level ≥ 0.5,
  0 otherwise.
- f_2 (**x**.**w**) = 1 if the activation level $\geq 1,5$,
  0 otherwise.

x_1	x_2	Out
0	0	
0	1	
1	0	
1	1	

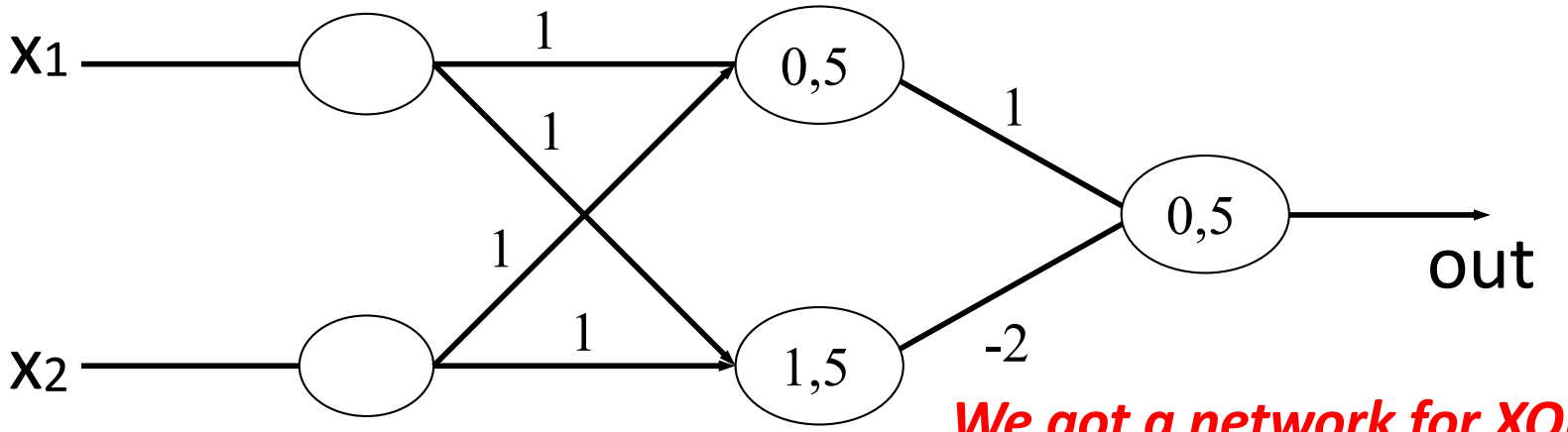


Make the recall of this network, filling the table.

Example of an MLP

- $x_1 = x_2$ = binary inputs.
- $w_1 = w_2 = w_3 = w_4 = w_5 = 1$ e $w_6 = -2$
- f_1 (**x**.**w**) = 1 if the activation level ≥ 0.5,
  0 otherwise.
- f_2 (**x**.**w**) = 1 if the activation level $\geq 1,5$,
  0 otherwise.

x_1	x_2	Out
0	0	0
0	1	1
1	0	1
1	1	0

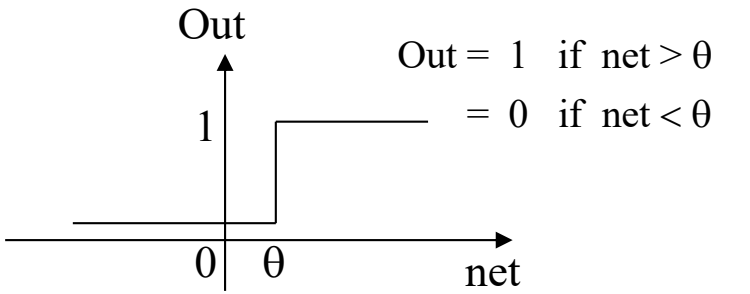


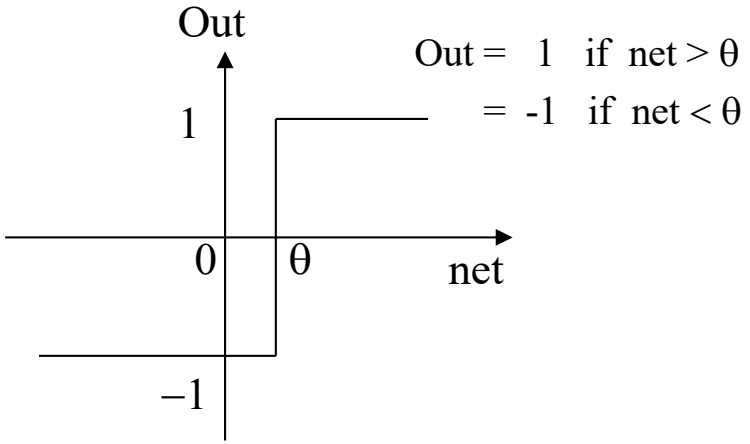
We got a network for XOR !!!

We now need an algorithm to train the weights ...

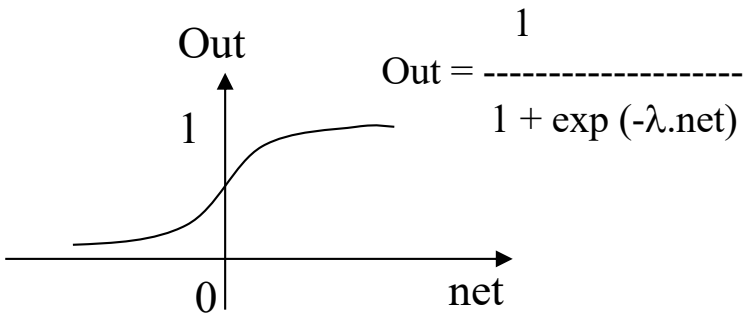
Types of Activation Function

- **Linear-Threshold functions:**

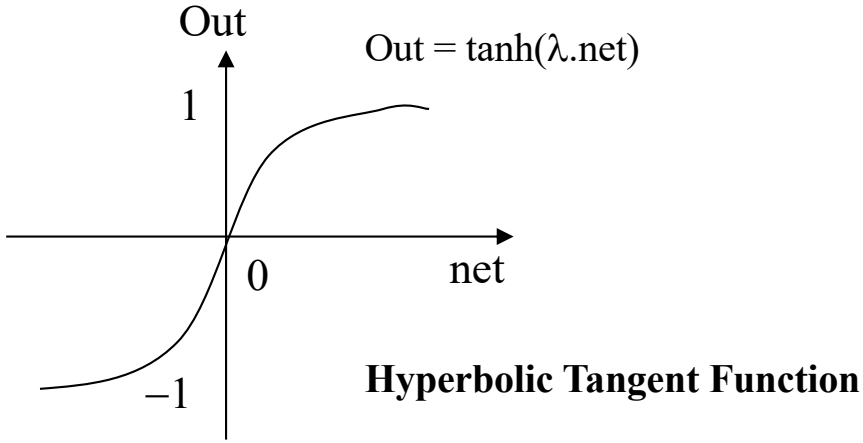




- **Sigmoid functions:**



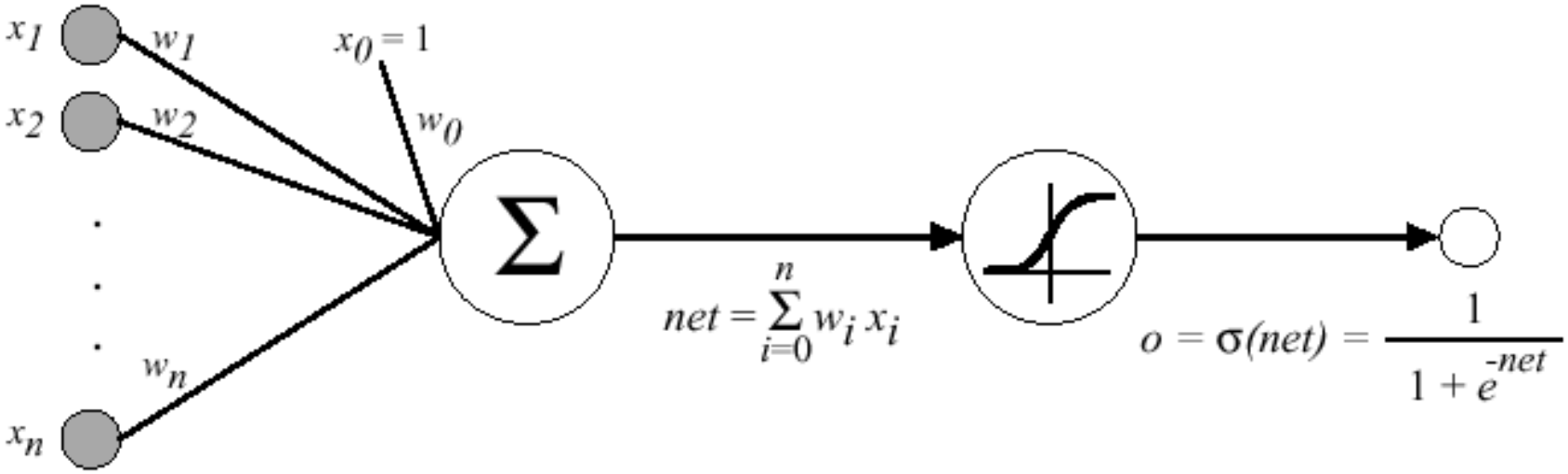
Logistics function



Hyperbolic Tangent Function

MLP / Sigmoid

$$\sigma(net) = \frac{1}{1+e^{-net}} \quad \text{or} \quad \sigma(net) = \tanh(net)$$



$$\sigma(x) = \frac{1}{1+e^{-x}} \Rightarrow \sigma'(x) = \sigma(x)\left(1-\sigma(x)\right)$$

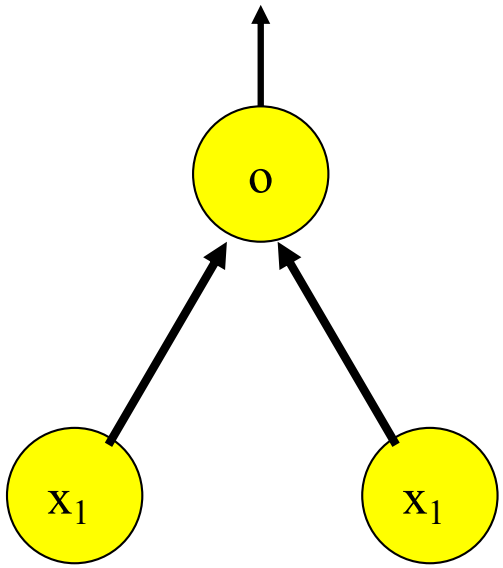
$$\sigma(x) = \tanh(x) \Rightarrow \sigma'(x) = \left(1-\sigma^2(x)\right)/2$$

Interesting property ...

derivative function

MLP training

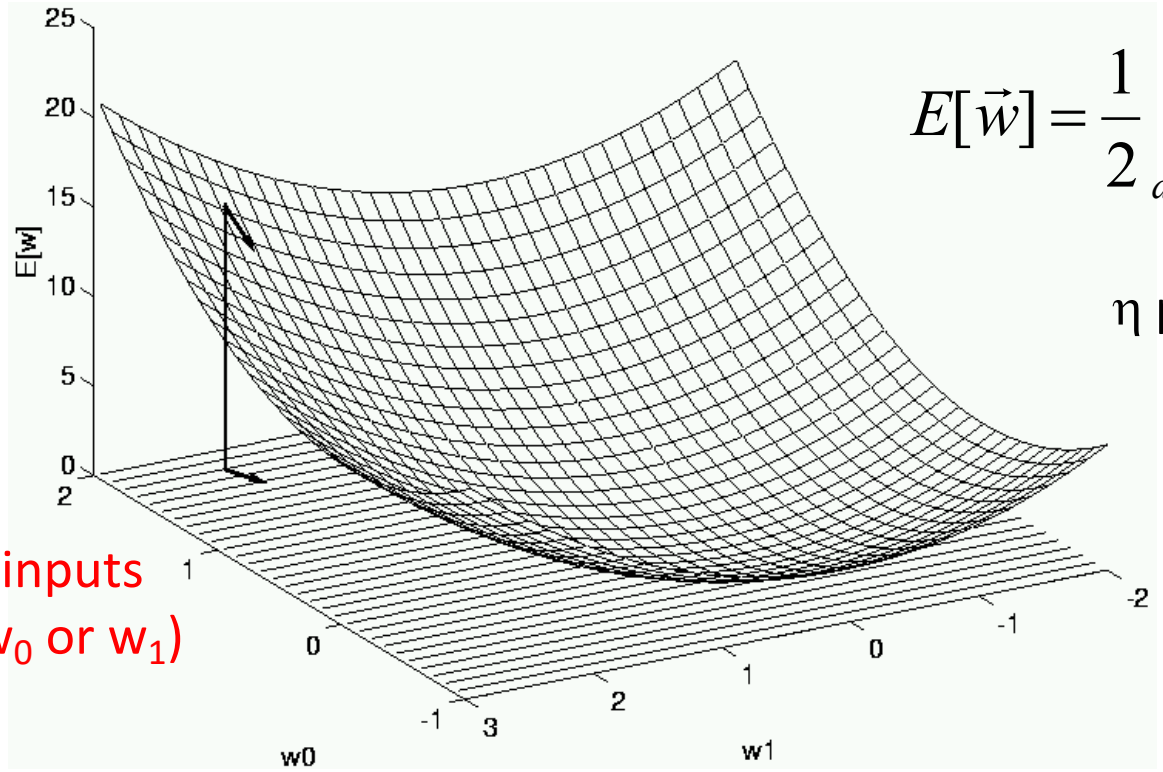
- **Key idea**: use of **gradient descent** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- Learn w_i's that minimize **squared error**:



$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

D = training data

Gradient Descent



$$E[\vec{w}] = \frac{1}{2}\sum_{d\ \varepsilon\ D}(t_d - o_d)^2$$

η positive

1 neuron with two inputs
(and two weights w_0 or w_1)

Gradient:

$$\nabla E[\vec{w}] = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n}\right]$$

Rule:

$$\Delta\vec{w} = -\eta\ \nabla E[\vec{w}] \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Because it is desired to move the weight vector in the direction that the error E decreases

Gradient Descent (one layer)

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_d (t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d \frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_d 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

$$= \sum_d (t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \sigma(\vec{w}\cdot\vec{x}_d))$$

$$= \sum_d (t_d - o_d)(-\frac{\partial}{\partial w_i}\sigma(\vec{w}\cdot\vec{x}_d))$$

σ is the logistics function

$$= -\sum_d (t_d - o_d)\sigma(\vec{w}\cdot\vec{x}_d)(1-\sigma(\vec{w}\cdot\vec{x}_d))x_{i,d}$$

o_d o_d

Gradient Descent: **multiple** Outputs

The error should now be redefined to k outputs:

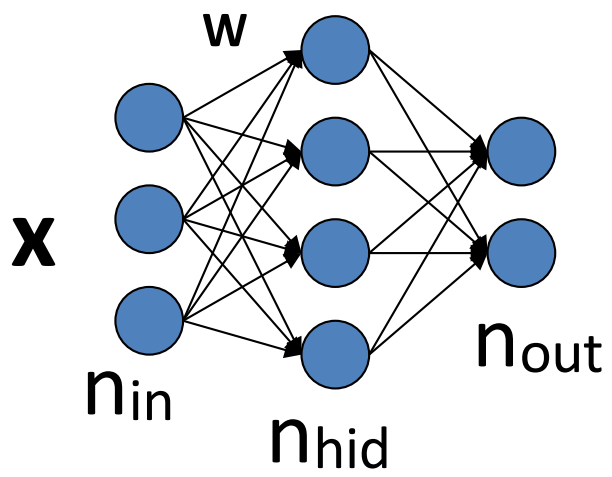
$$E[\vec{w}] = \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

D = training data

Outputs

The *Backpropagation* algorithm

- A feed-forward network with one hidden layer with n_{hid} sigmoidal units, n_{out} outputs (sigmoidal units), n_{in} inputs, several training samples $< x, t >$



t (target output)
o (net output)

Input parameters of the backpropagation algorithm:

$\{<x_1, t_1>, <x_2, t_2>....\}, \eta, n_{hid}, n_{in}, n_{out}$

The *Backpropagation* algorithm

- Initialize w to small random numbers
- Until the termination condition is met, do:
  - For each <x,t> in training-examples, do:

//Propagate the input forward:

1. Propagate the input forward and compute each o_k.

//Propagate the error backward:

2. For each output unit k, calculate the error δ_k:

$$\delta_k \leftarrow o_k (1 - o_k) (t_k - o_k)$$

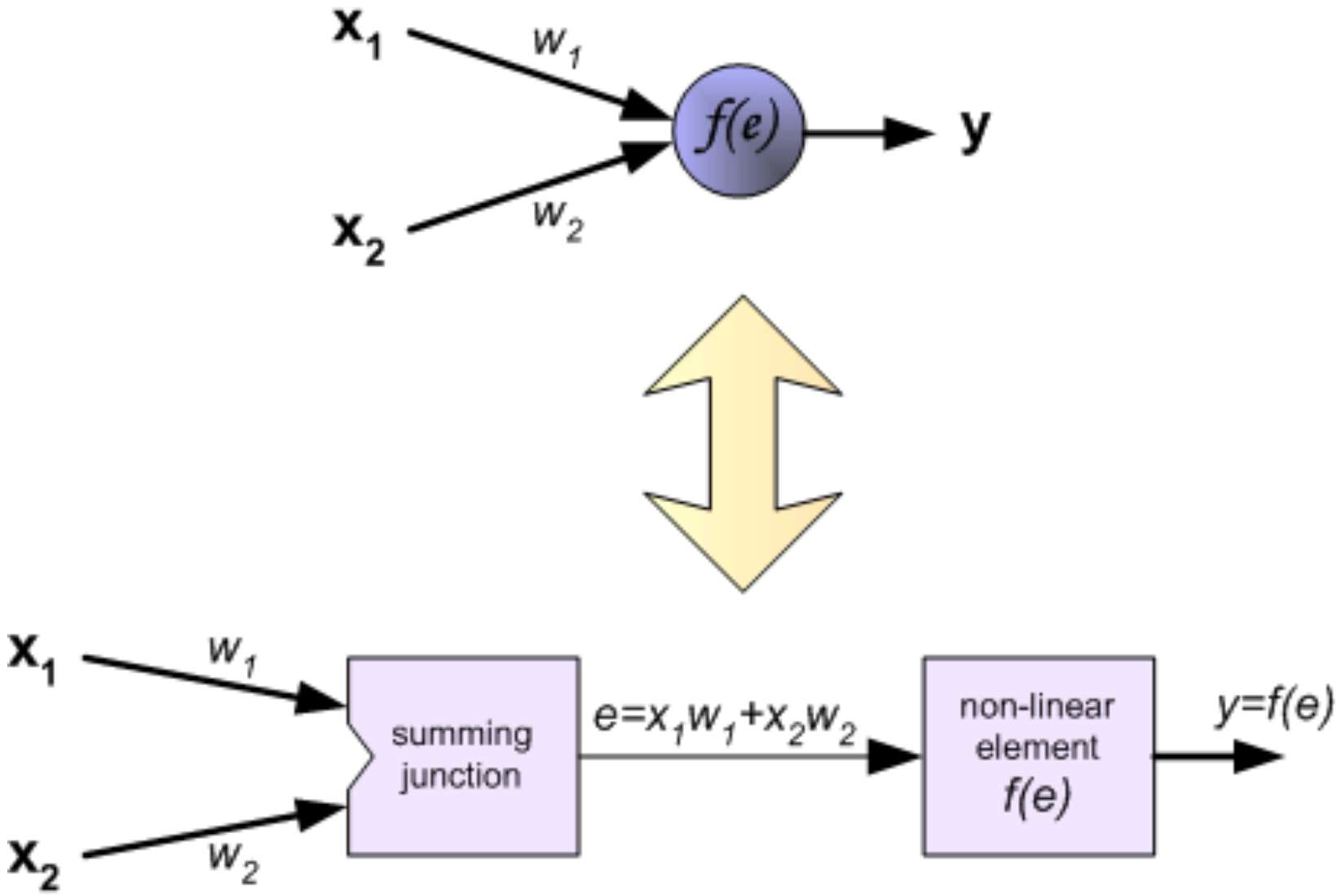
3. For each hidden unit h, calculate the error δ_h:

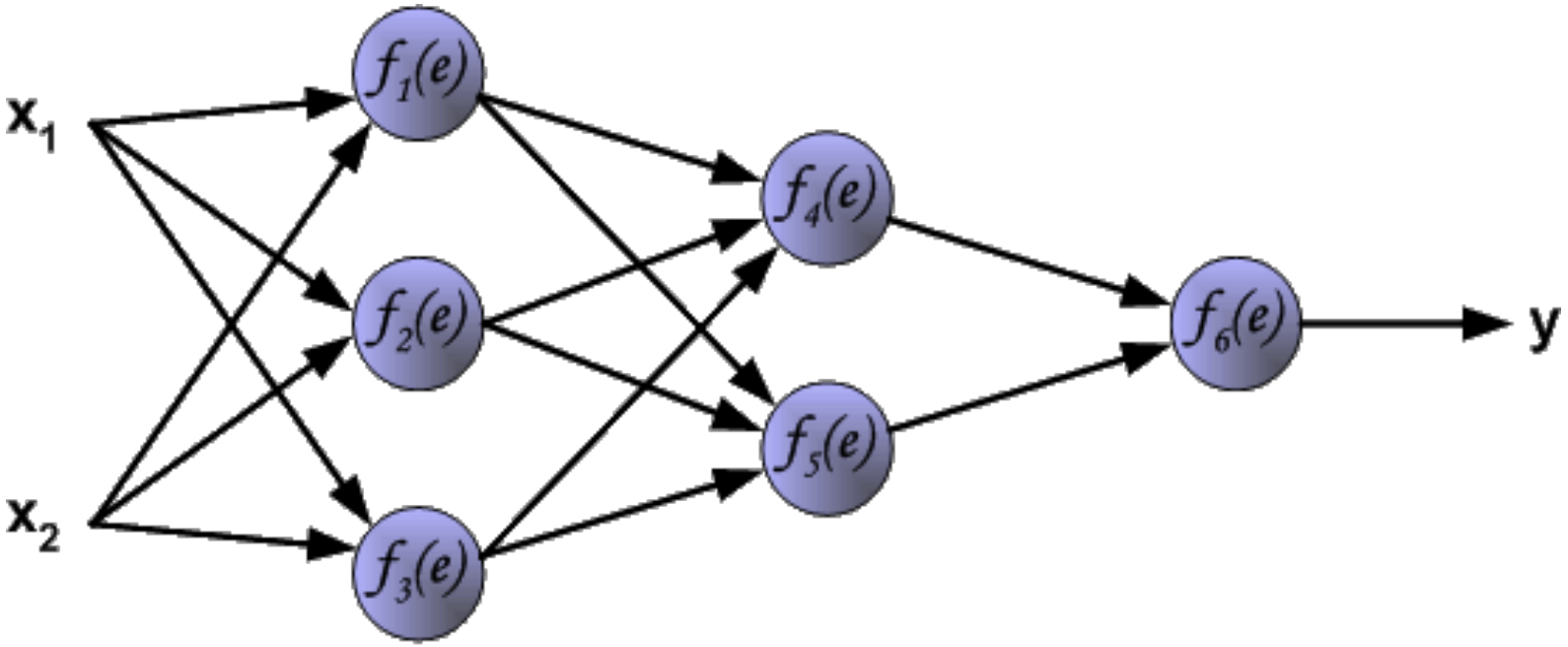
$$\delta_h \leftarrow o_h (1 - o_h) \sum_{k \in saidas} w_{hk} . \delta_k$$

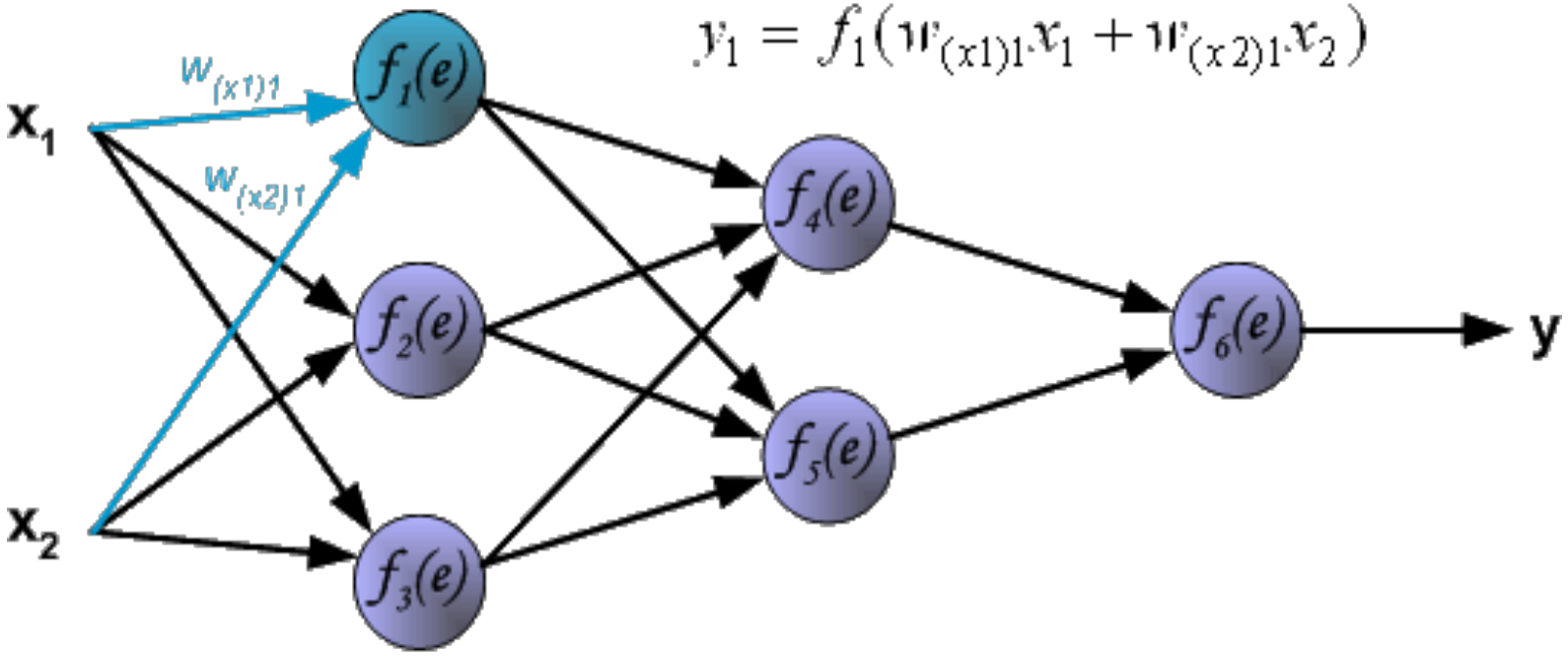
4. Update each network weight w_{ij}:

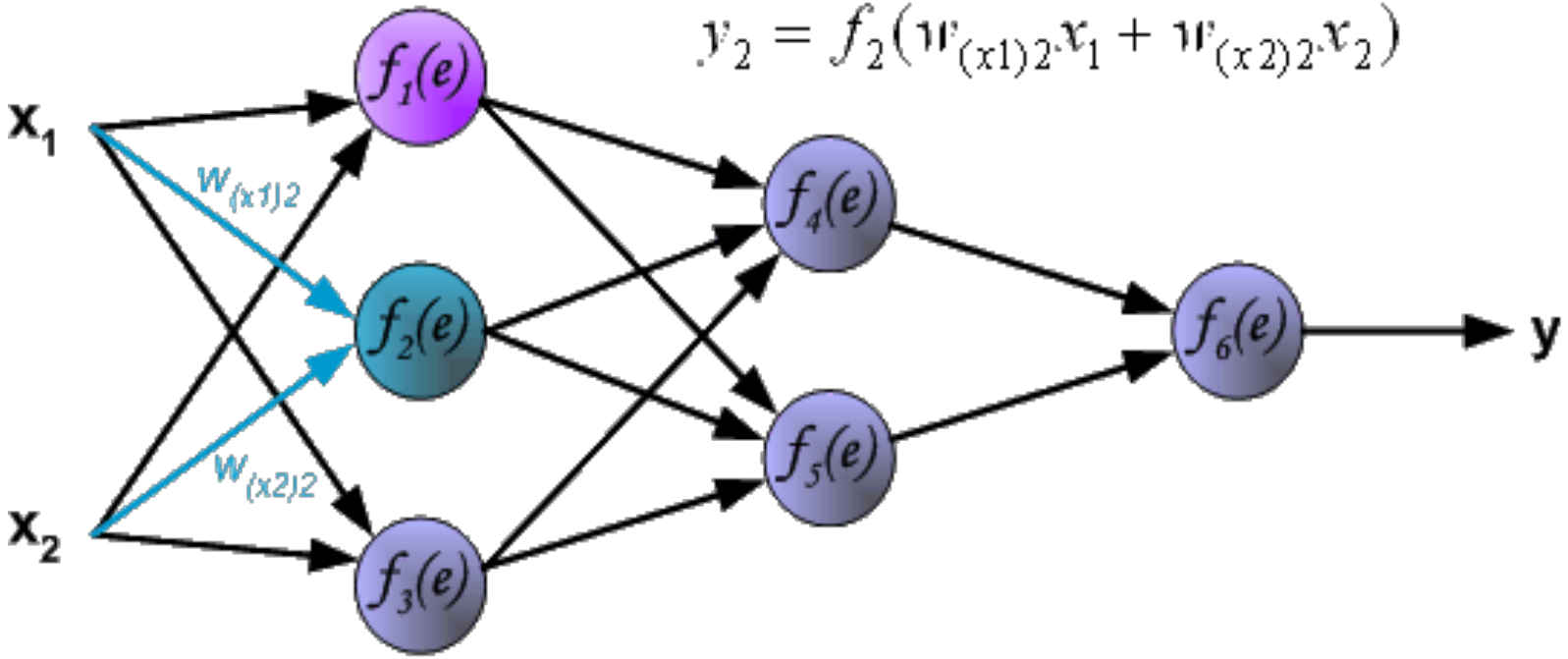
$$w_{ij} \leftarrow w_{ij} + \eta \delta_j x_{ij}$$

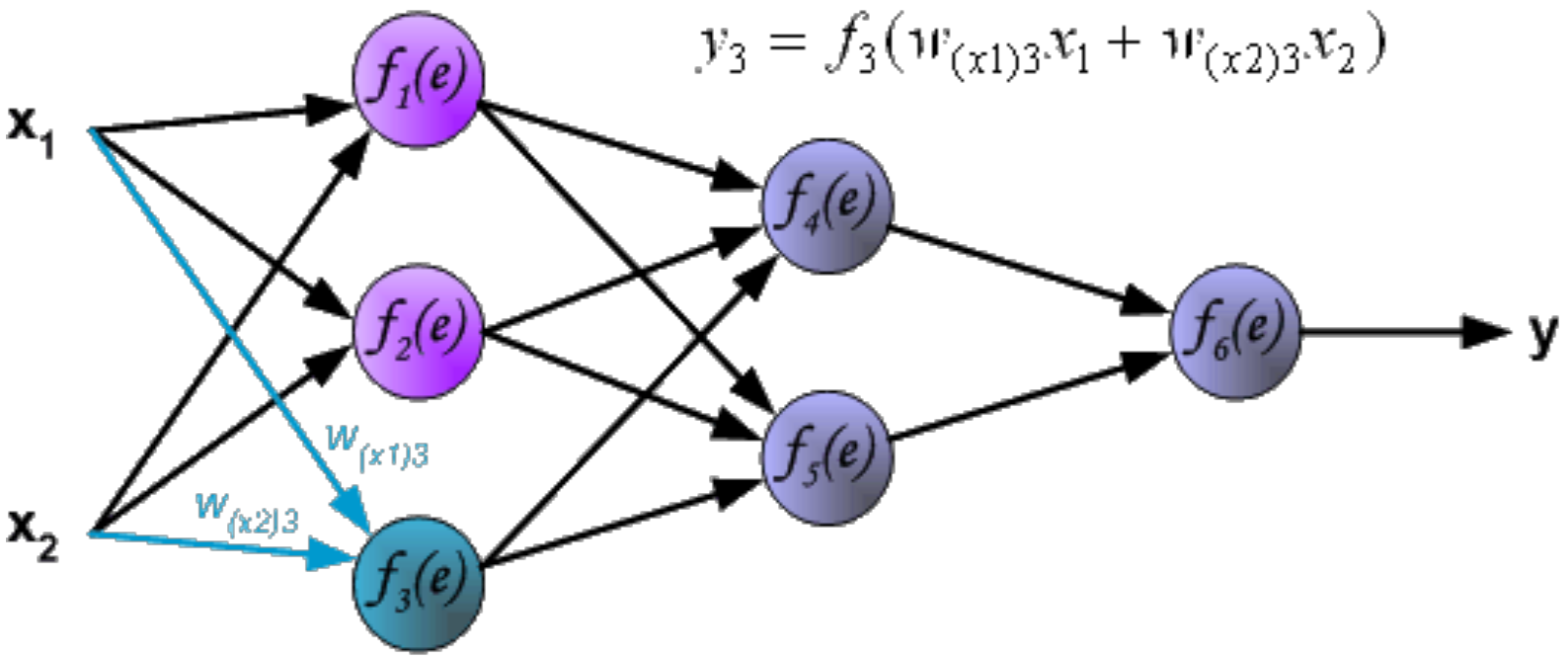
Input from i to j

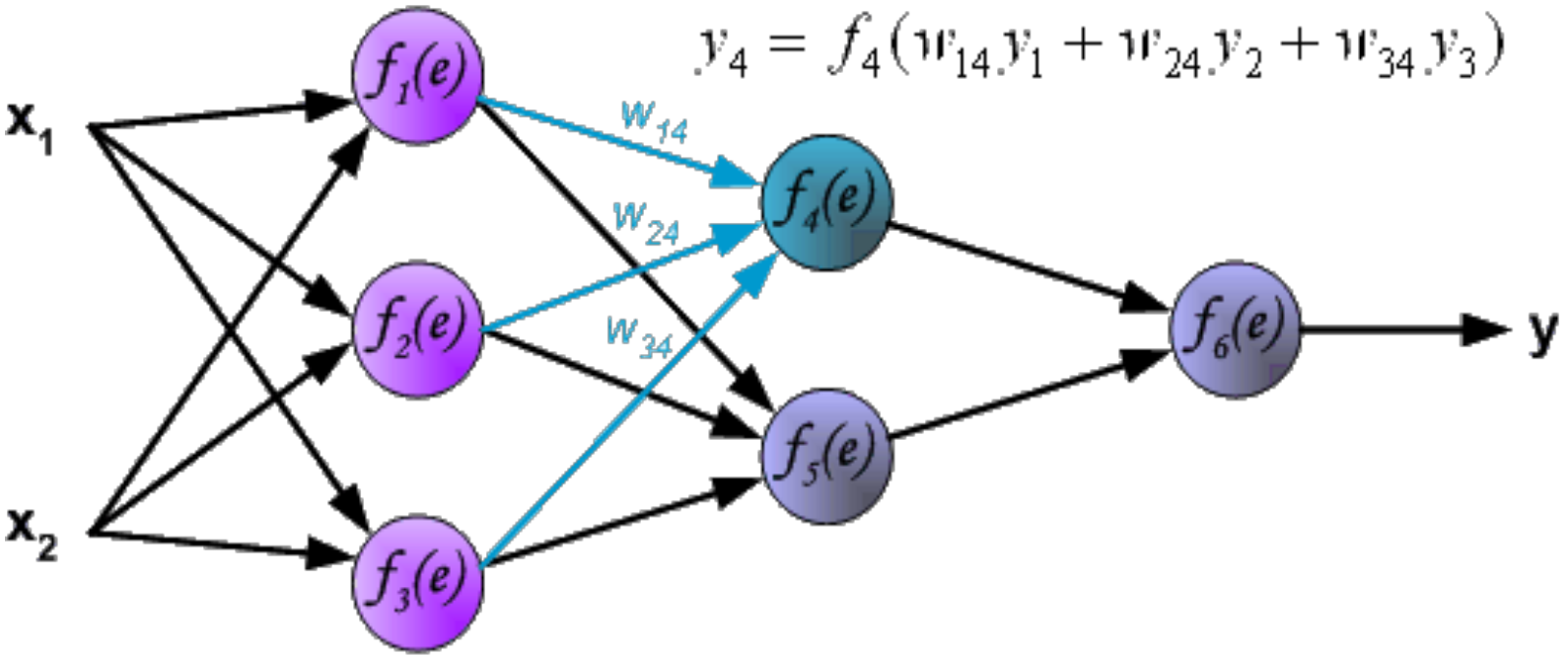

x1
w1
f(e)
y
x2
w2
x1
w1
summing
junction
e=x1w1+x2w2
non-linear
element
f(e)
y=f(e)
x2
w2

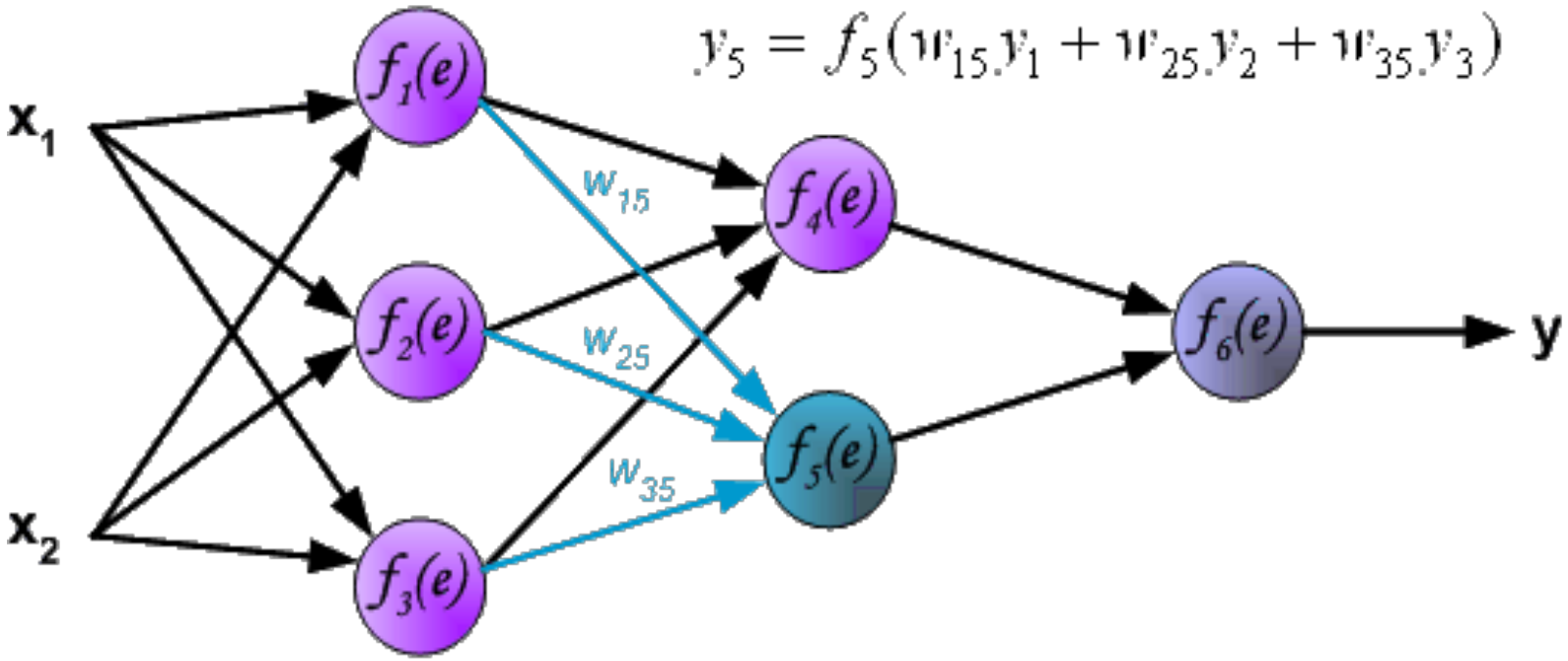

x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y

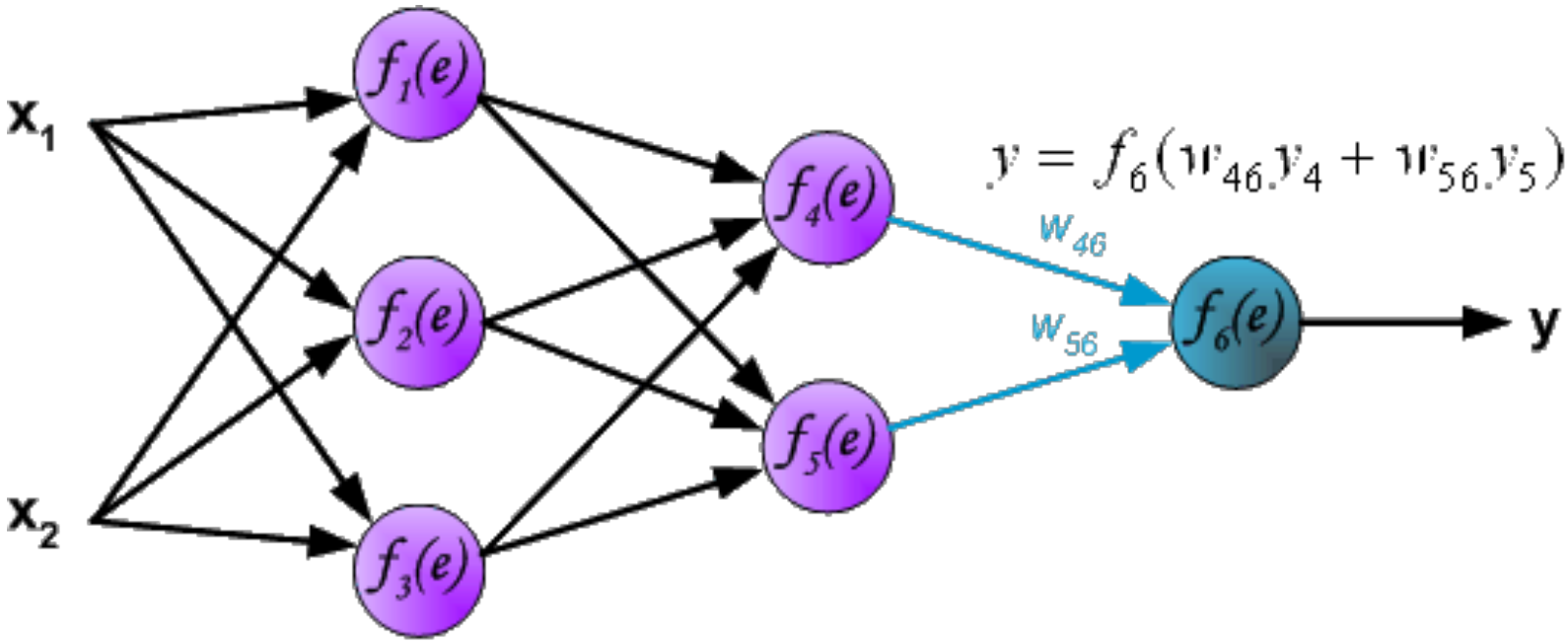

$y_1 = f_1(w_{(x1)1}x_1 + w_{(x2)1}x_2)$
x_1
x_2
$w_{(x1)1}$
$w_{(x2)1}$
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
y

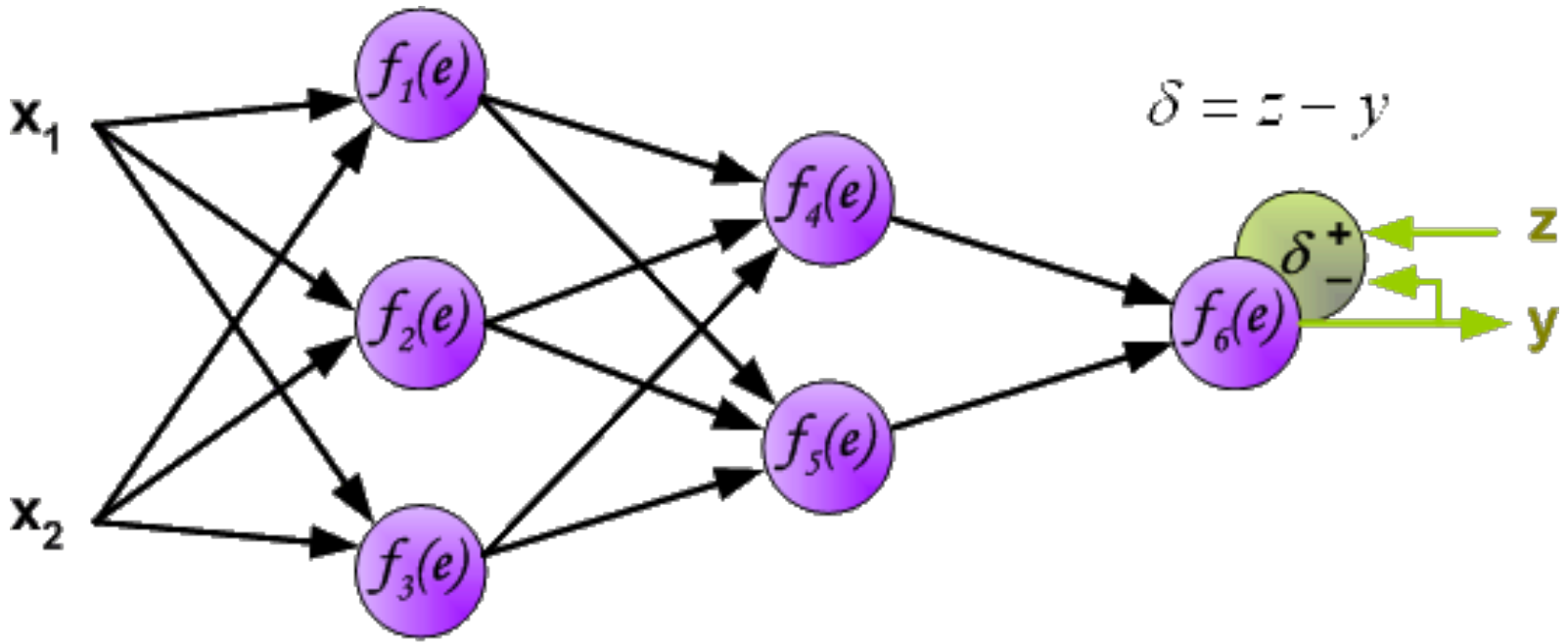

$y_2 = f_2(w_{(x1)2}x_1 + w_{(x2)2}x_2)$
x1
x2
w(x1)2
w(x2)2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y

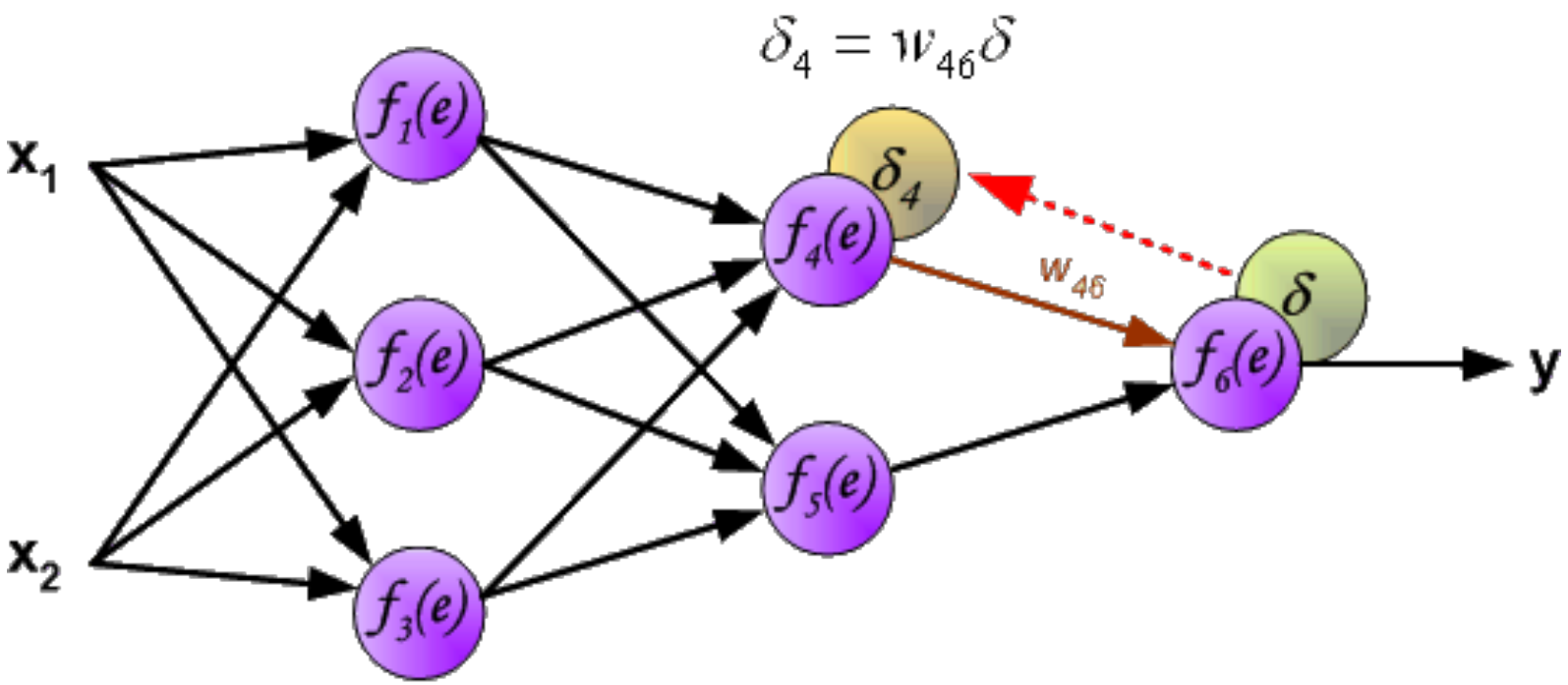

$y_3 = f_3(w_{(x1)3}x_1 + w_{(x2)3}x_2)$
x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y
W(x1)3
W(x2)3

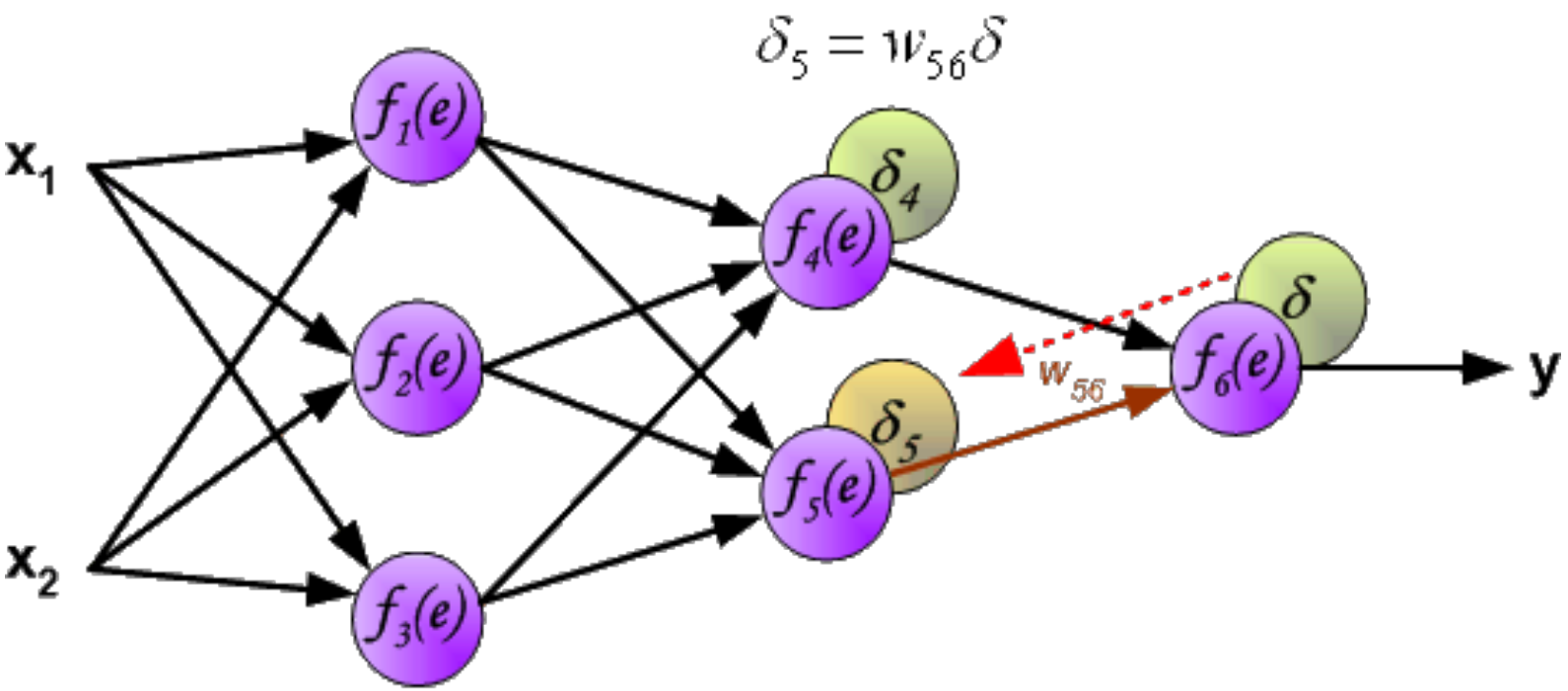

$y_4 = f_4(w_{14}.y_1 + w_{24}.y_2 + w_{34}.y_3)$
x1
x2
f1(e)
f2(e)
f3(e)
w14
w24
w34
f4(e)
f5(e)
f6(e)
y

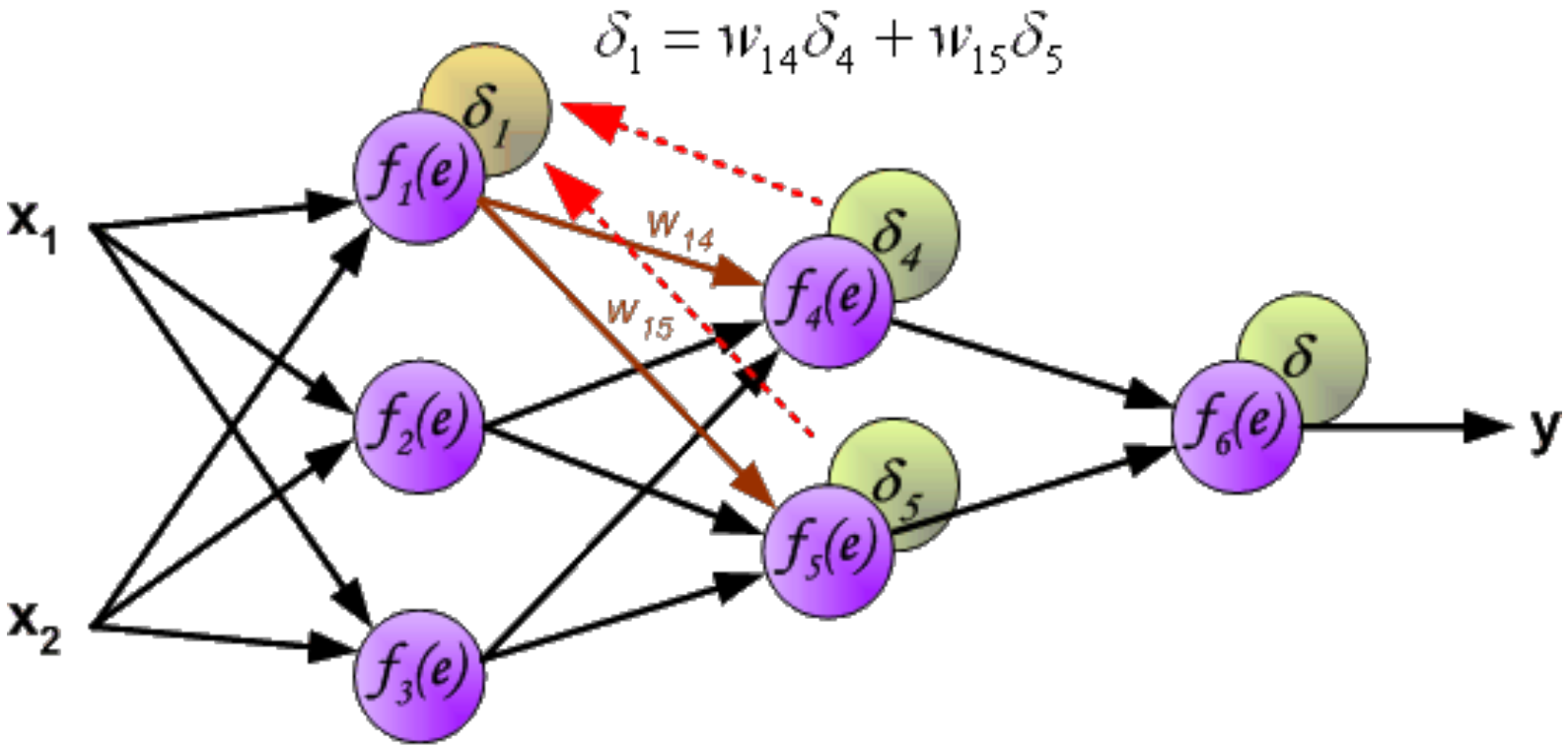

$y_5 = f_5(w_{15}.y_1 + w_{25}.y_2 + w_{35}.y_3)$
x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
w15
w25
w35
y

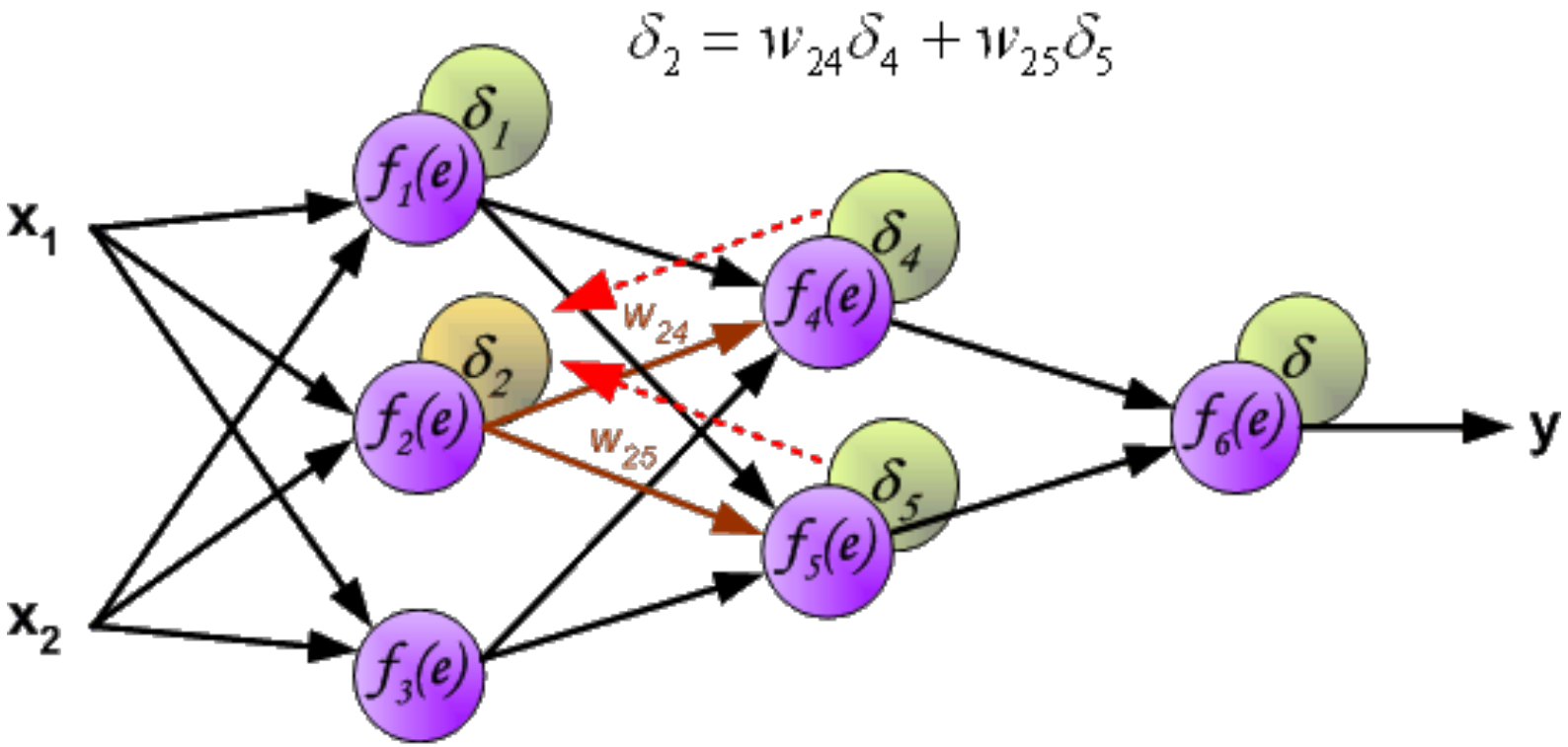

x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
y = f6(w46.y4 + w56.y5)
w46
w56
y

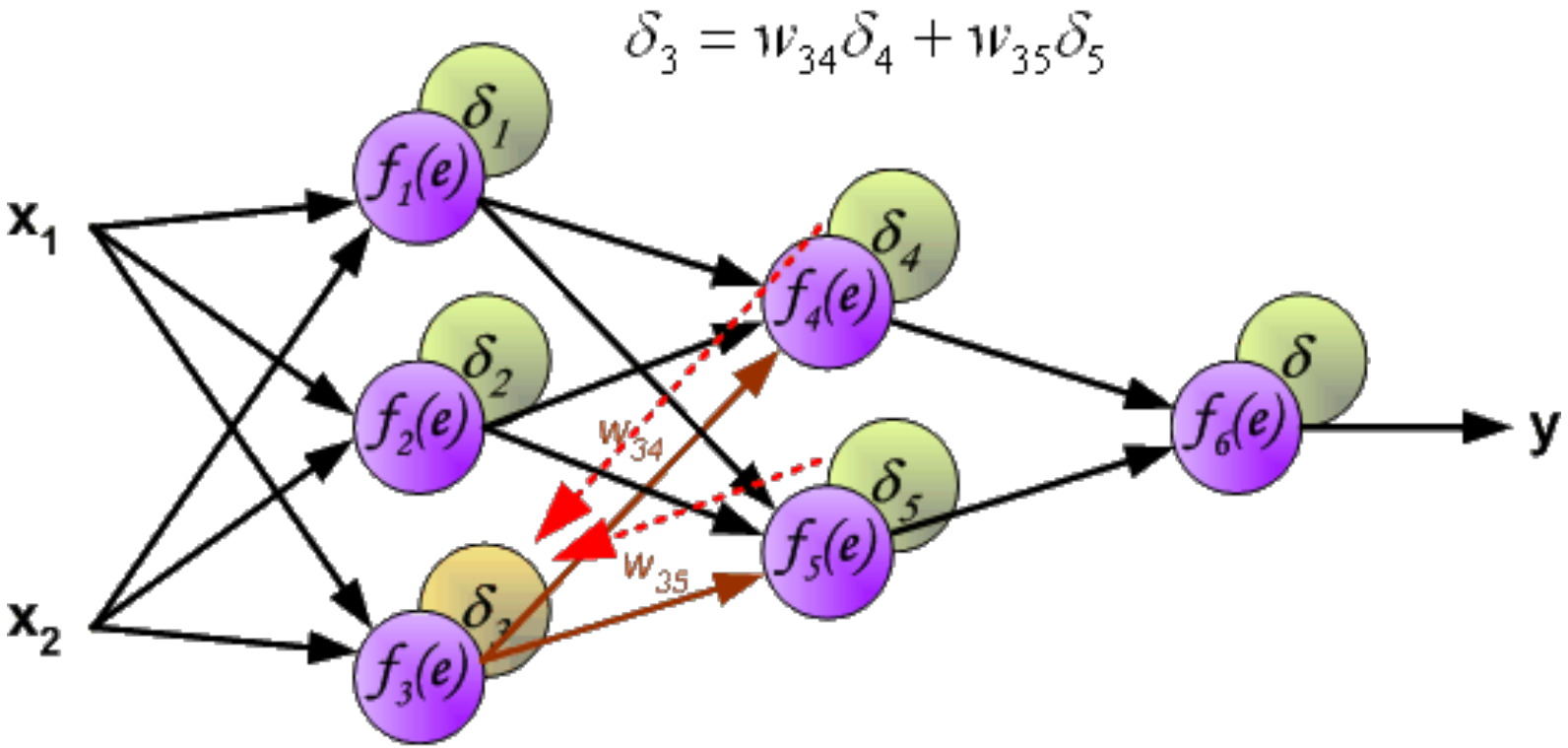

x_1
x_2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
$\delta = z - y$
δ + −
z
y


$\delta_4 = w_{46}\delta$
x1
x2
f1(e)
f2(e)
f3(e)
f4(e)
f5(e)
f6(e)
δ4
δ
w46
y


$\delta_5 = w_{56}\delta$
x1
x2
$f_1(e)$
$f_2(e)$
$f_3(e)$
$f_4(e)$
$f_5(e)$
$f_6(e)$
δ_4
δ_5
δ
w_{56}
y

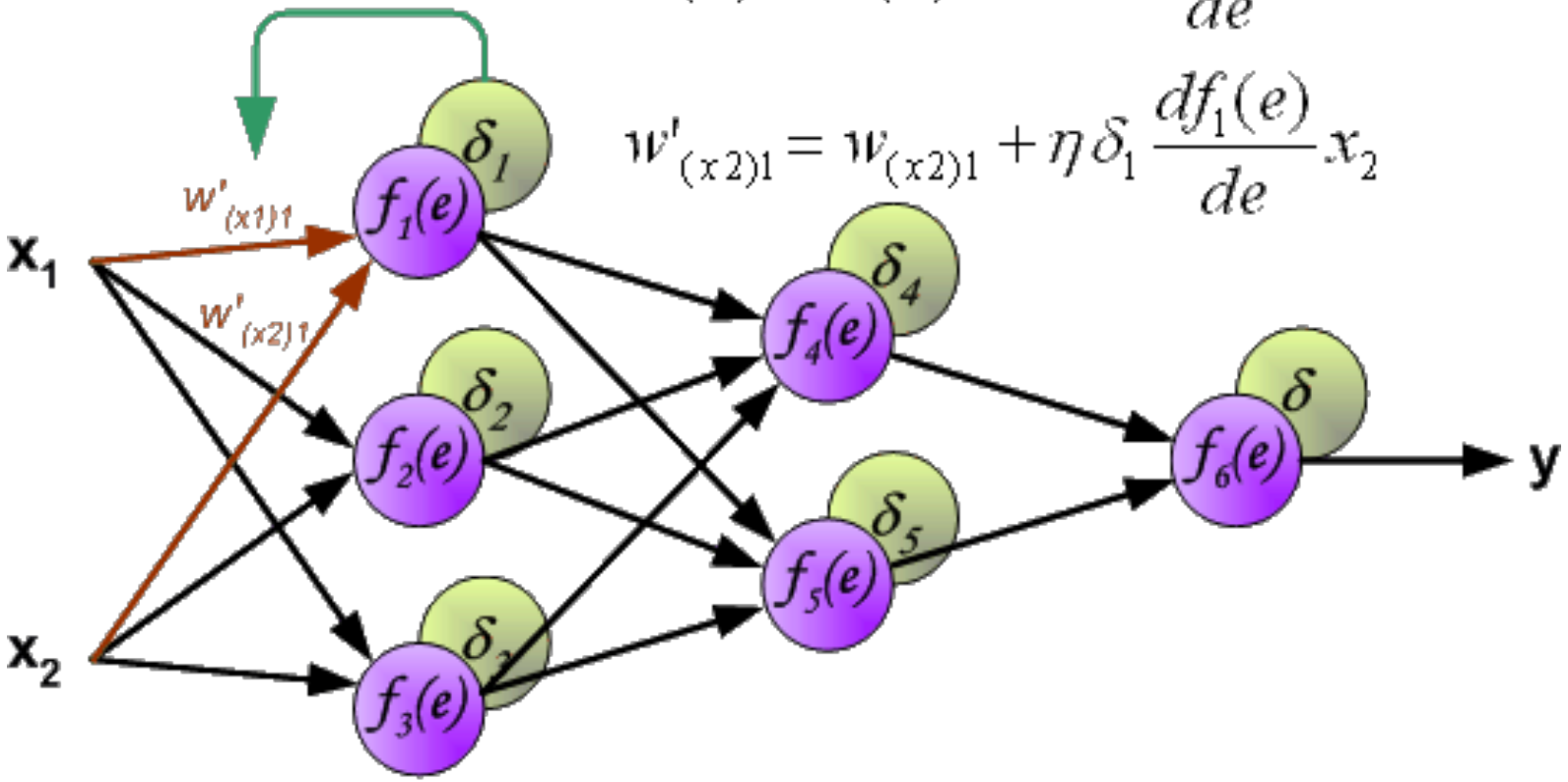

$\delta_1 = w_{14}\delta_4 + w_{15}\delta_5$
δ_1
$f_1(e)$
w_{14}
w_{15}
δ_4
$f_4(e)$
δ
$f_6(e)$
y
x_1
$f_2(e)$
δ_5
$f_5(e)$
x_2
$f_3(e)$


$\delta_2 = w_{24}\delta_4 + w_{25}\delta_5$
x1
x2
δ1
f1(e)
δ2
f2(e)
f3(e)
δ4
f4(e)
w24
w25
δ5
f5(e)
δ
f6(e)
y


$\delta_3 = w_{34}\delta_4 + w_{35}\delta_5$
x1
x2
δ1
f1(e)
δ2
f2(e)
δ3
f3(e)
δ4
f4(e)
δ5
f5(e)
δ
f6(e)
y
w34
w35

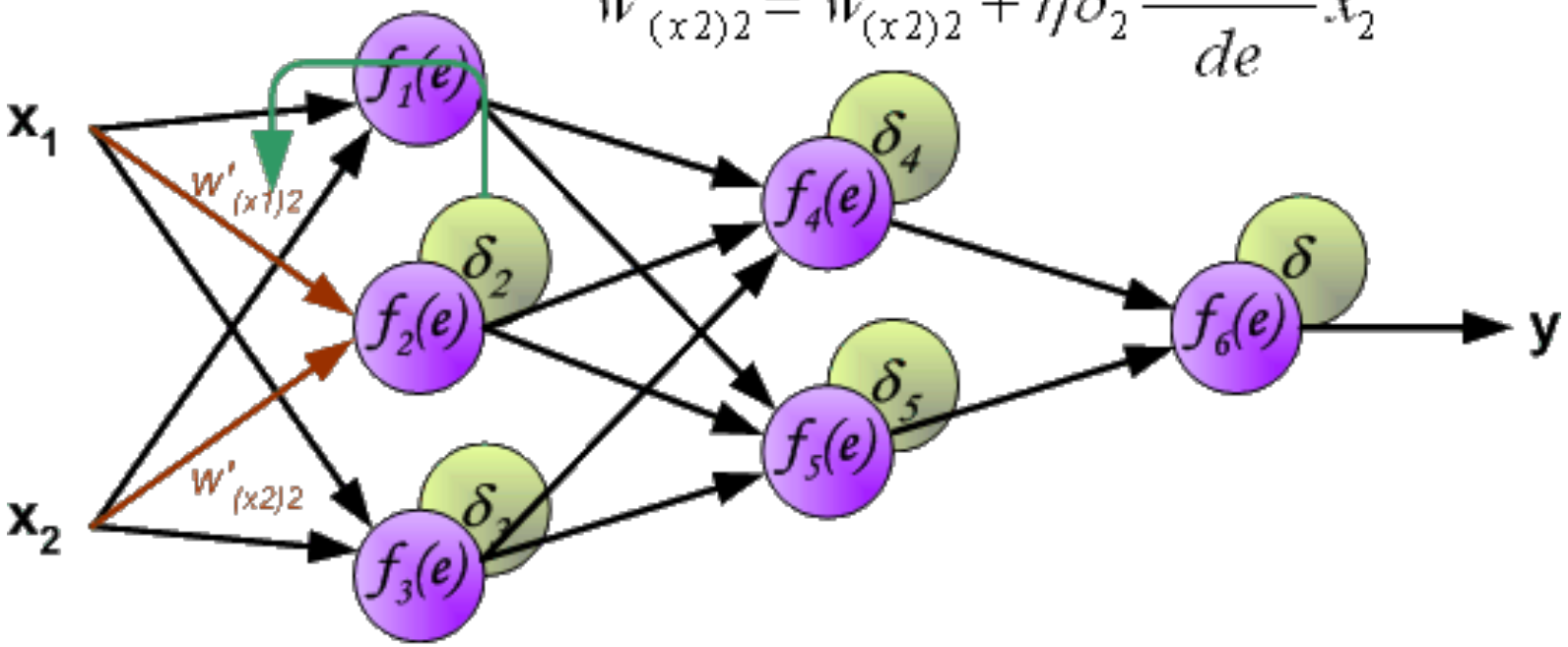
$$w'_{(x1)1} = w_{(x1)1} + \eta \delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta \delta_1 \frac{df_1(e)}{de} x_2$$



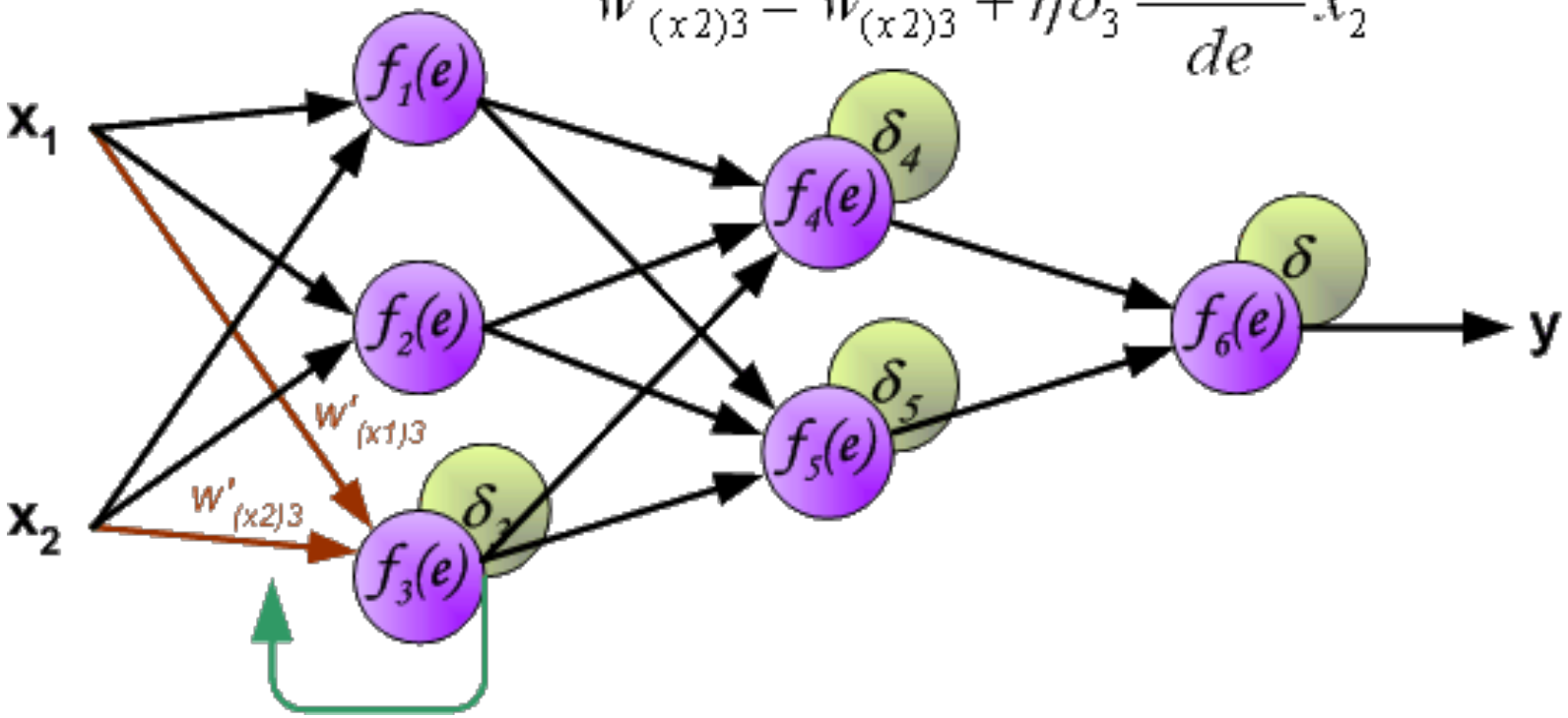
$$w'_{(x1)2} = w_{(x1)2} + \eta \delta_2 \frac{df_2(e)}{de} x_1$$

$$w'_{(x2)2} = w_{(x2)2} + \eta \delta_2 \frac{df_2(e)}{de} x_2$$



$$w'_{(x1)3} = w_{(x1)3} + \eta \delta_3 \frac{df_3(e)}{de} x_1$$

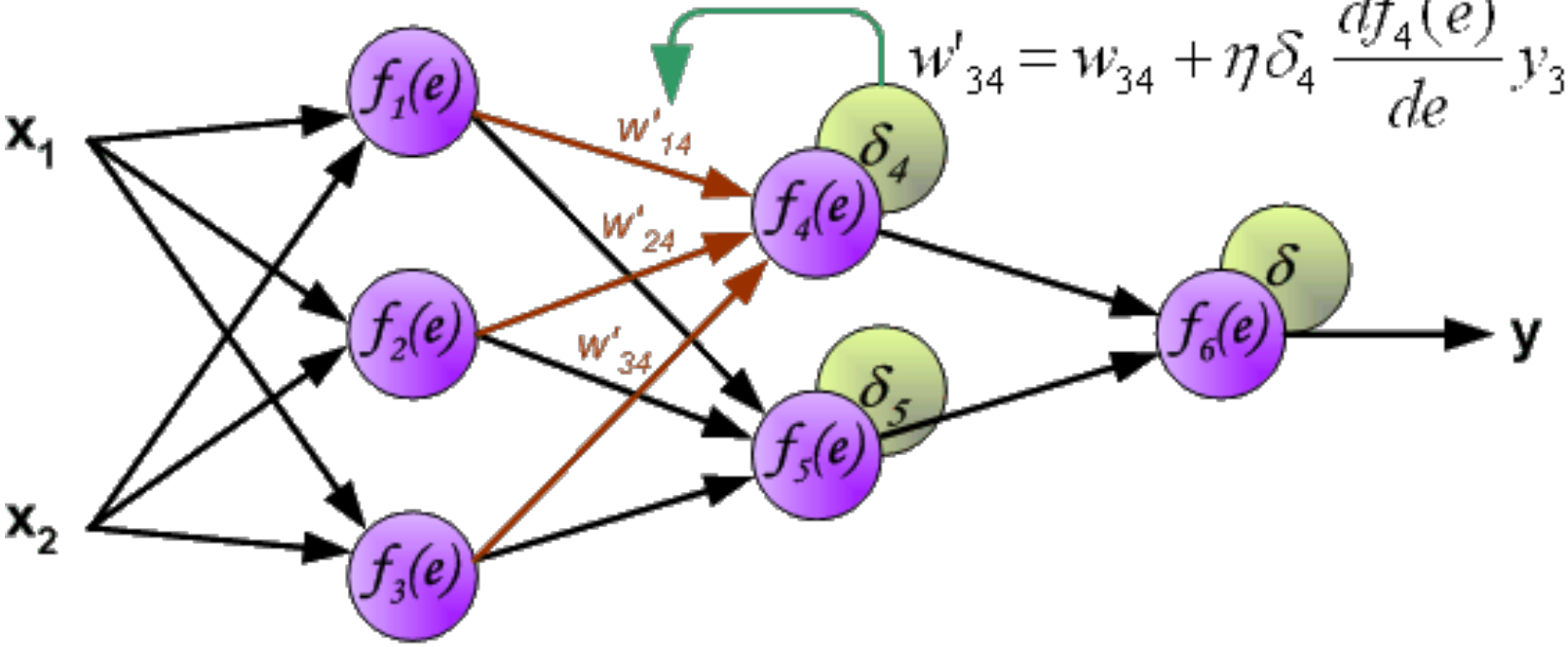
$$w'_{(x2)3} = w_{(x2)3} + \eta \delta_3 \frac{df_3(e)}{de} x_2$$



$$w'_{14} = w_{14} + \eta\delta_4 \frac{df_4(e)}{de} y_1$$

$$w'_{24} = w_{24} + \eta\delta_4 \frac{df_4(e)}{de} y_2$$

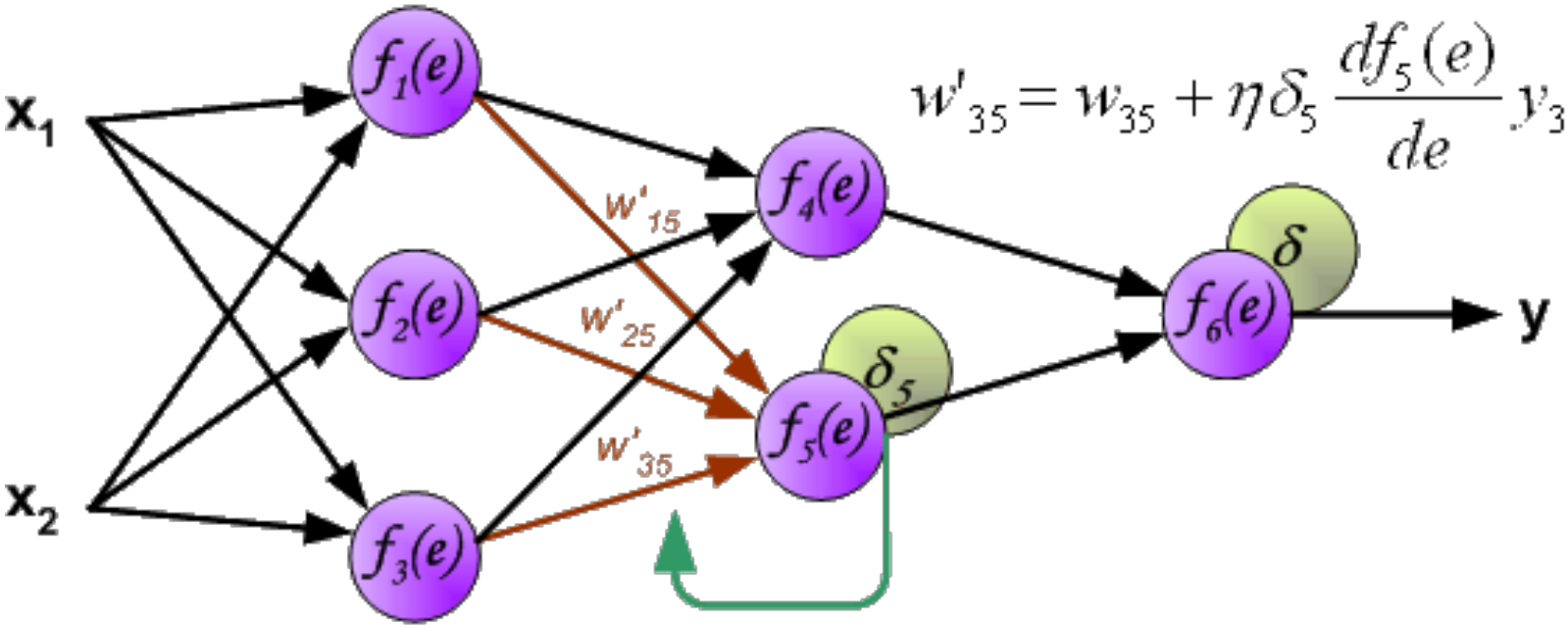
$$w'_{34} = w_{34} + \eta\delta_4 \frac{df_4(e)}{de} y_3$$



$$w'_{15} = w_{15} + \eta \delta_5 \frac{df_5(e)}{de} y_1$$

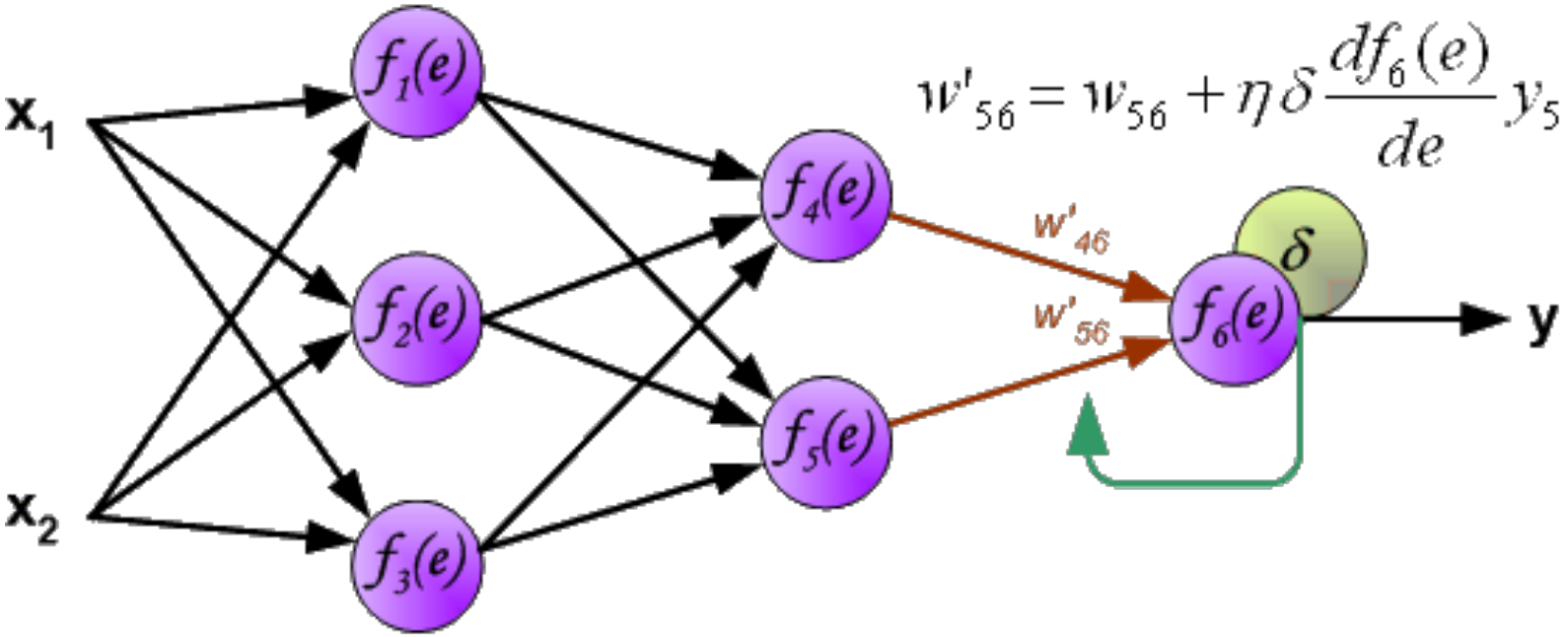
$$w'_{25} = w_{25} + \eta \delta_5 \frac{df_5(e)}{de} y_2$$

$$w'_{35} = w_{35} + \eta \delta_5 \frac{df_5(e)}{de} y_3$$

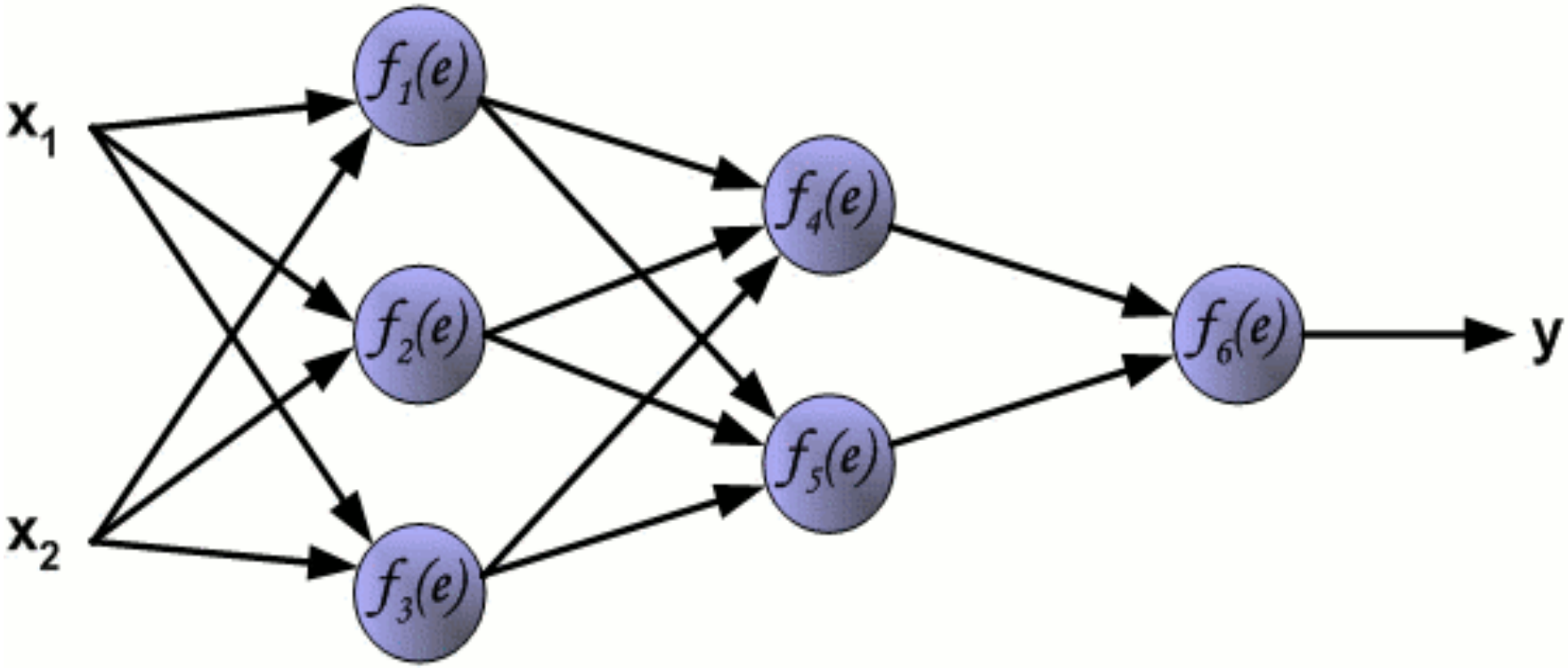


$$w'_{46} = w_{46} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_4$$

$$w'_{56} = w_{56} + \eta\,\delta\,\frac{df_6(e)}{de}\,y_5$$



Backpropagation: animation

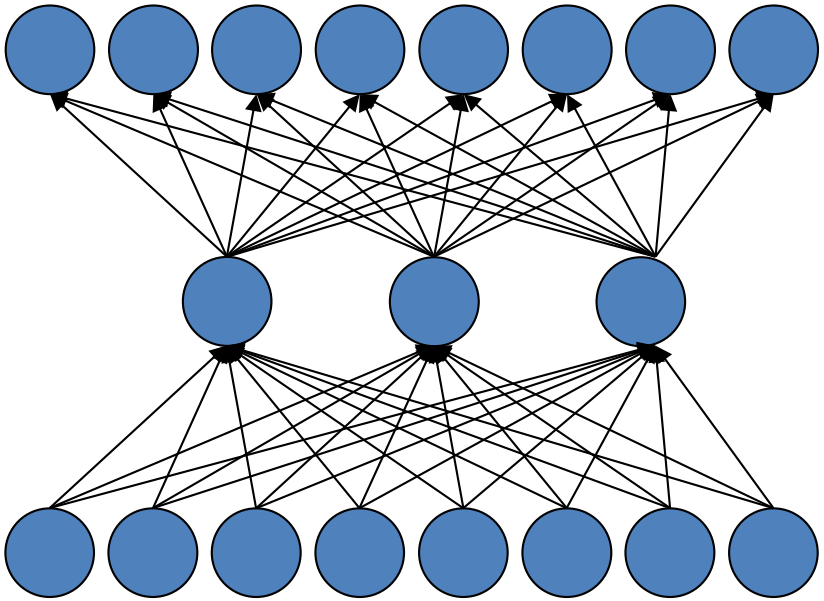


http://www.trapexit.org/images/b/ba/Animate_ANN.gif

Example:

initial weights: (-0.1 a +0.1); $\eta = 0.3$

Entrada		Saída
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

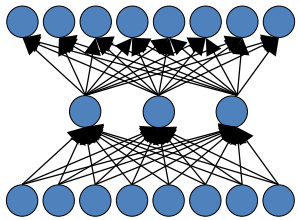


Learning the representation (inner layer)

1 0 0
0 1 1

Entrada		Saída hs		Saída
10000000	→	.89 .04 .08	→	10000000
01000000	→	.15 .99 .99	→	01000000
00100000	→	.01 .97 .27	→	00100000
00010000	→	.99 .97 .71	→	00010000
00001000	→	.03 .05 .02	→	00001000
00000100	→	.01 .11 .88	→	00000100
00000010	→	.80 .01 .98	→	00000010
00000001	→	.60 .94 .01	→	00000001

Intermediate representation "discovers" binary code !!!!



Application × ANN type

- There are many other types of ANN:
  - For classification or prediction tasks, usually feed-forward networks (such as MLP) are used.
  - For clustering tasks, the types of network used are: Simple Competitive Networks, Adaptive Resonance Theory (ART) networks, Kohonen Self-Organizing Maps (SOM).
  - In association tasks, an ANN can be trained to "remember" a number of patterns; the type of ANN usually used for this task is Hopfield network.

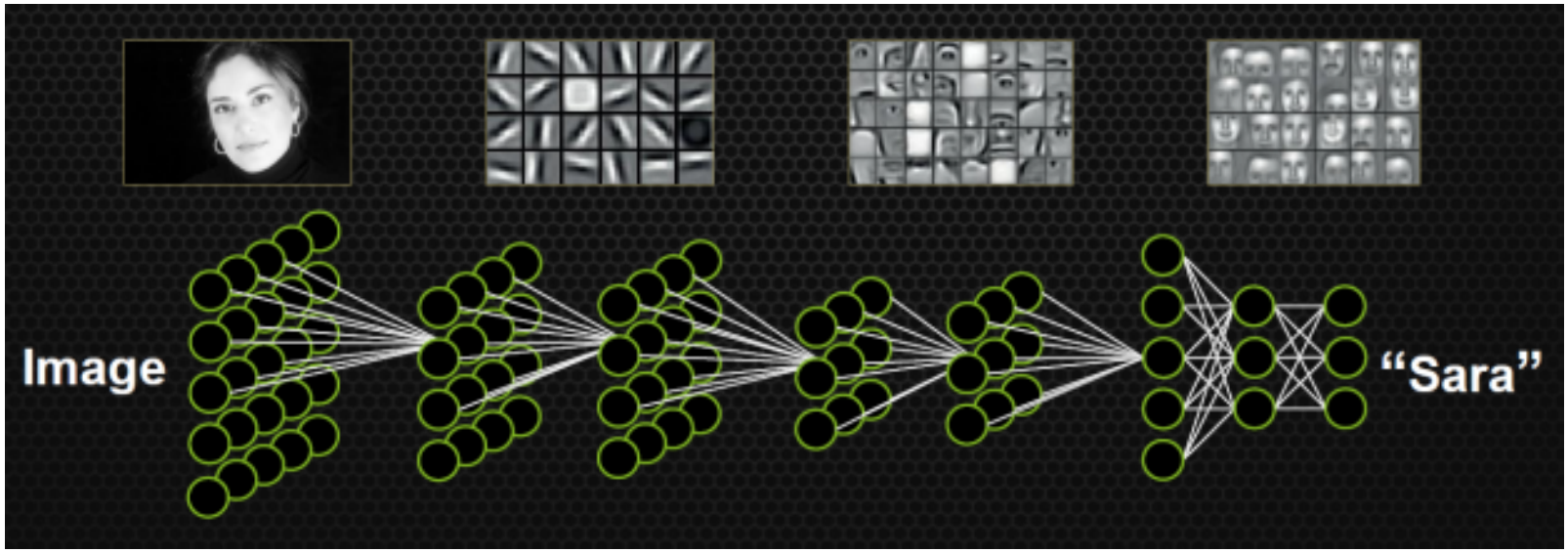
Recent advances and future applications of ANNs

- Integration of fuzzy logic into neural networks
- Pulsed neural networks
- Hardware specialized for neural networks
- Deep Learning:
  - enable much deeper (and larger) networks (5 to 10 hidden layers): have the ability of building up a complex hierarchy of concepts.
  - one can show that there are functions which a k-layer network can represent compactly (with a number of hidden units that is **polynomial** in the number of inputs), that a (k − 1)-layer network cannot represent unless it has an **exponentially** large number of hidden units.

References

- Russel, S.; Norvig, P. Artificial Intelligence: a modern approach. 2nd.edition. Prentice Hall, 2003. Cap. 20.5.
- Mitchell, T.M. Machine Learning. WCB/McGraw-Hill, 1997. Cap.4.
- *Simon Haykin. Neural Networks: A Comprehensive Foundation.*
- *Livro em Português: Braga, Ludermir e Carvalho. Redes Neurais Artificiais. LTC.*

There are very interesting stuff on the web.



Deep Learning

Anna Helena Reali Costa

“more intuitive” applications

Face recognition

“Baseball player is throwing ball in game."

Generating natural language descriptions

Dog or Mop?

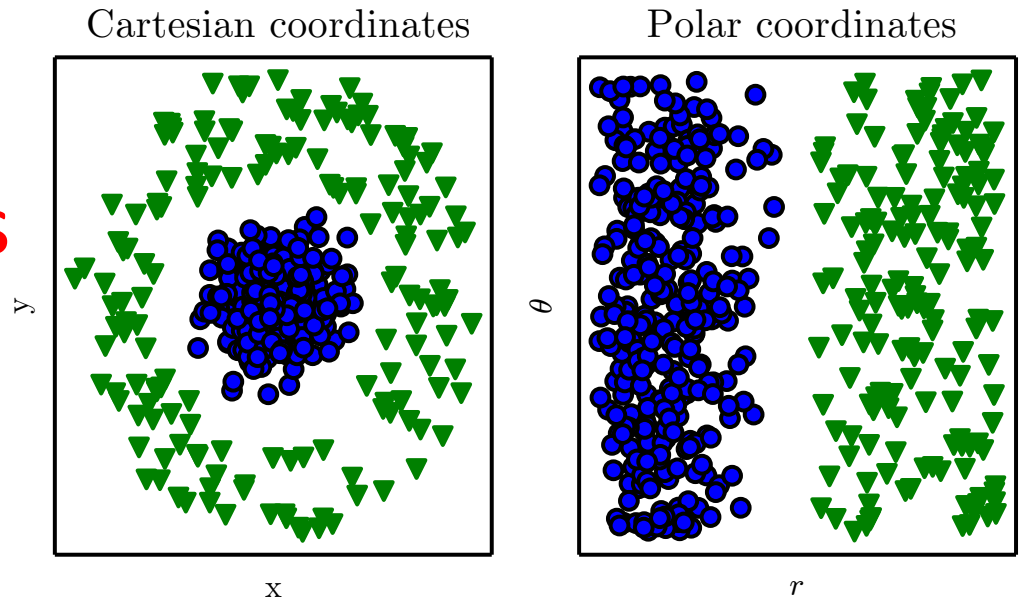
Chiuaua or Muffin?

Deep Learning Applications

Machine Learning depends on Representation

- The performance of simple machine learning algorithms depends heavily on the **representation** of the data they are given (***features***!).

Representations Matter!!

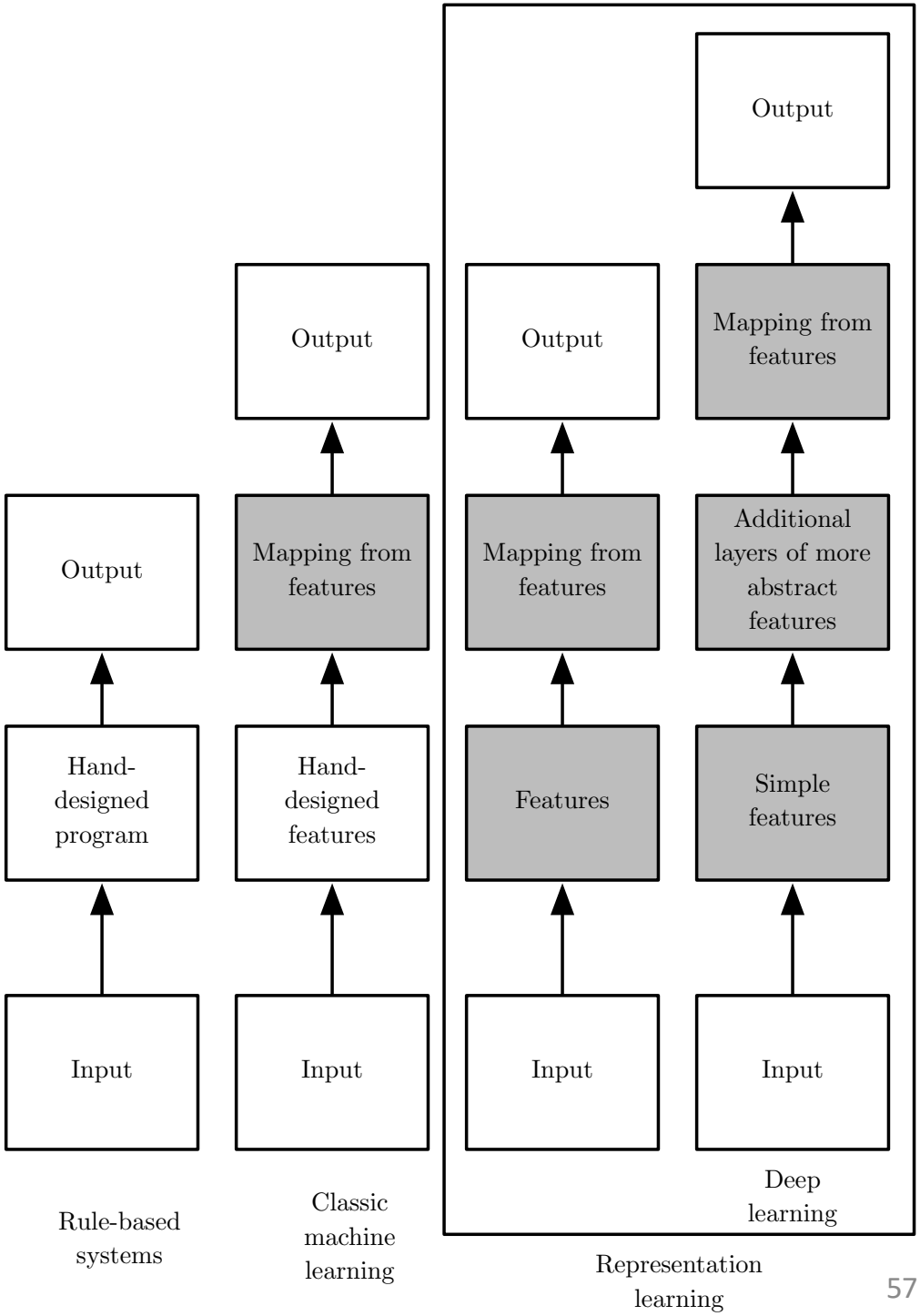


Representation Learning

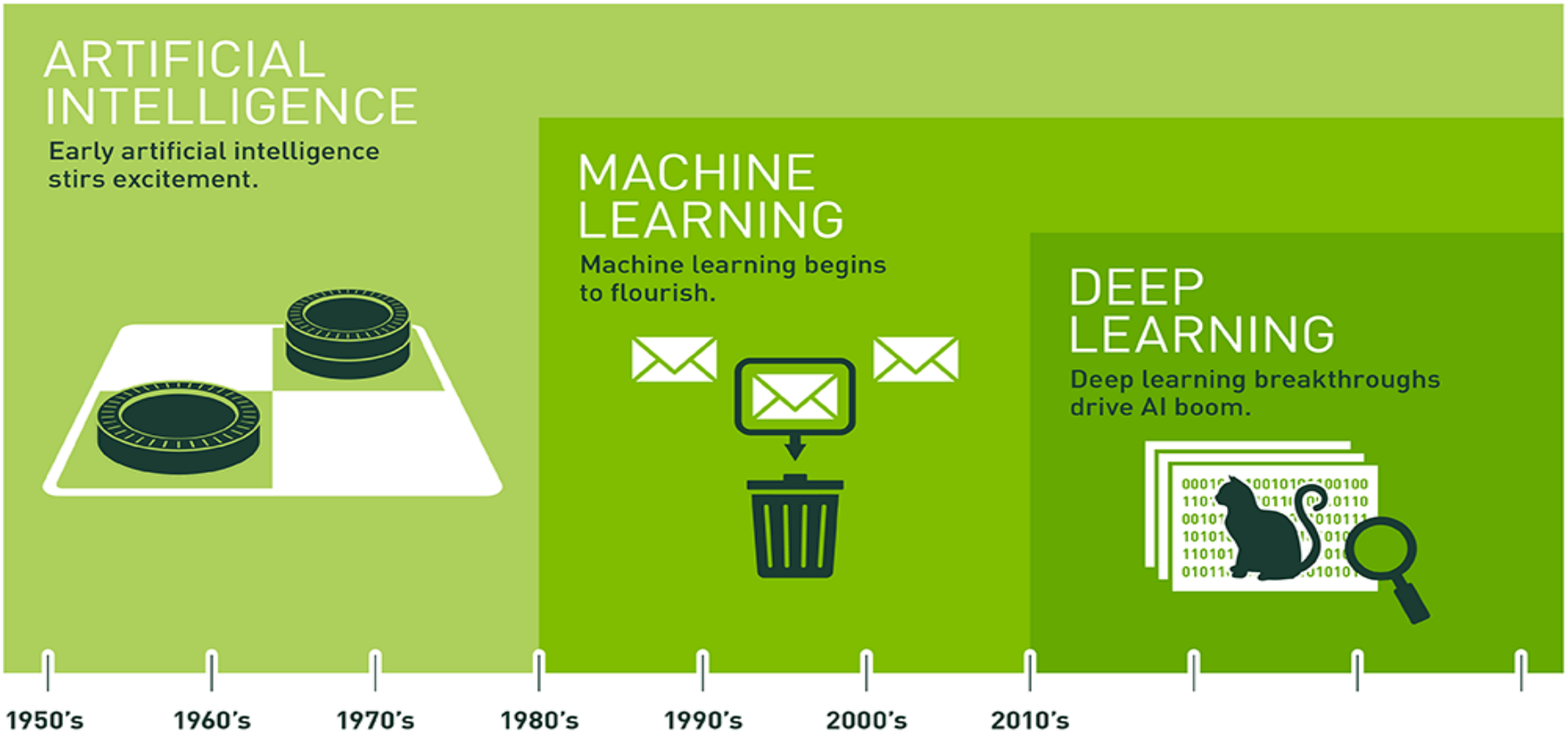
- Solution: to use machine learning to **discover** not only the mapping from representation to output but also the representation itself.
  - This approach is known as **representation learning.**

- **Deep learning** allows the computer to build complex representations out of simpler representations.

Learning Multiple Components



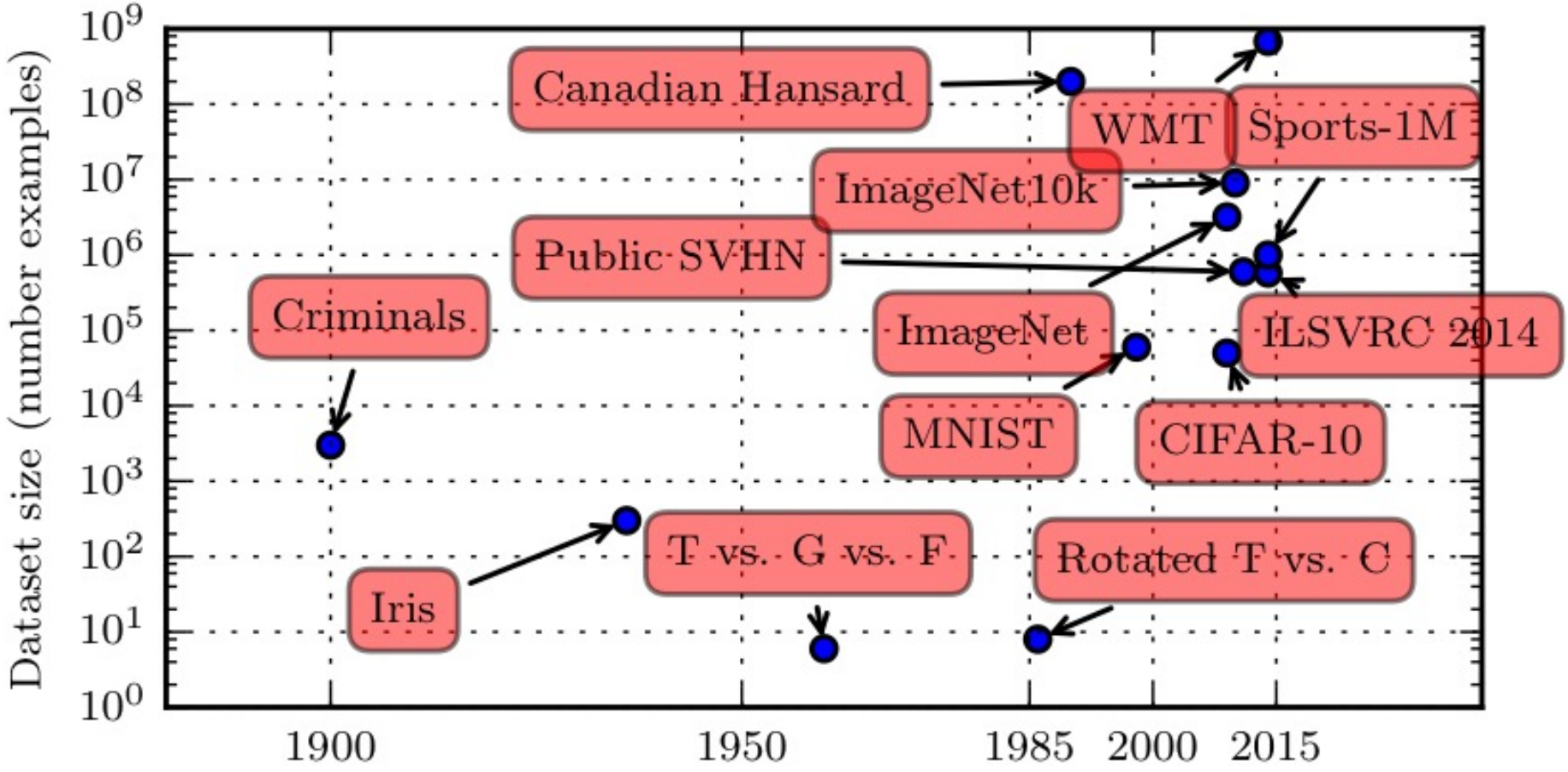
Deep Learning



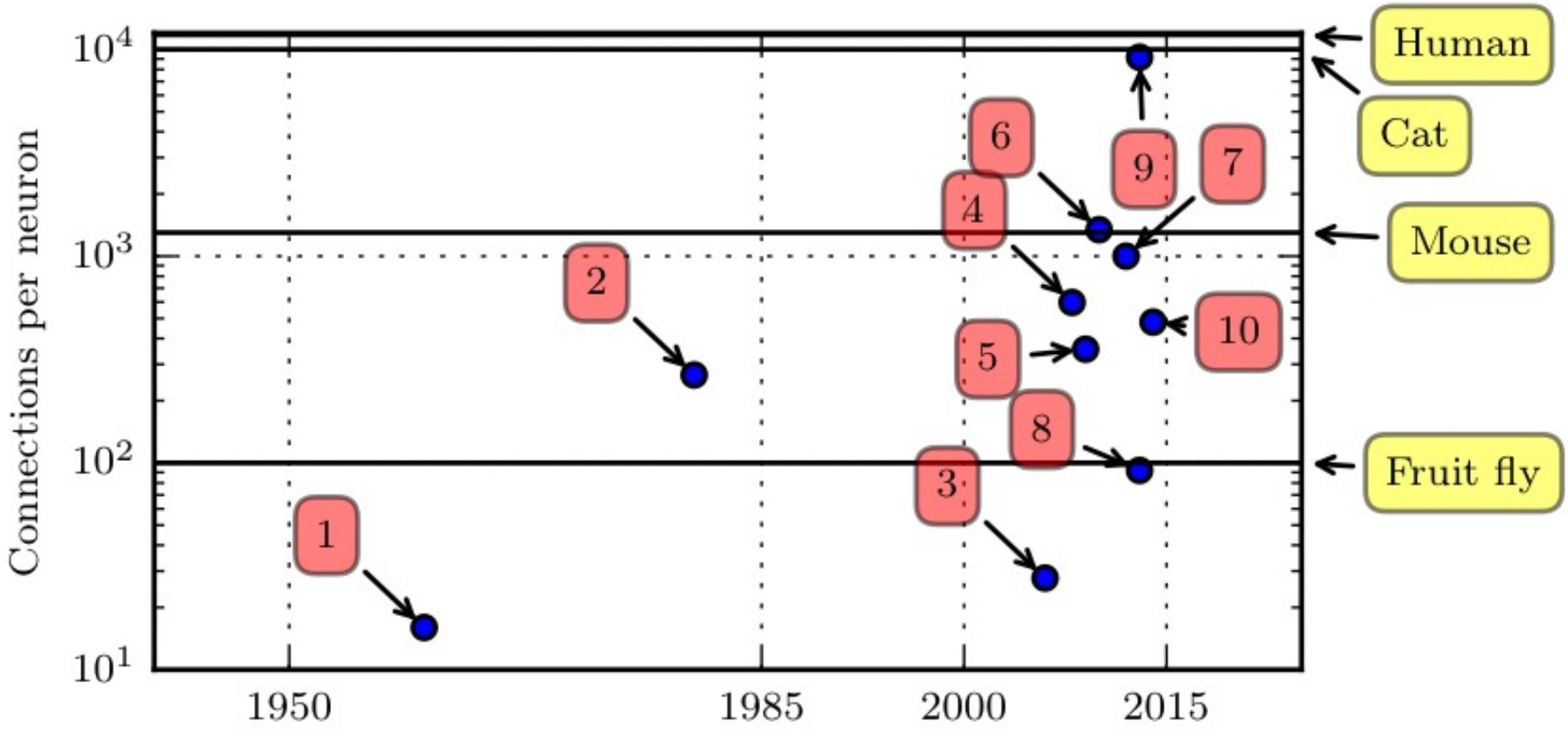
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

Historical Trends:
Growing Datasets

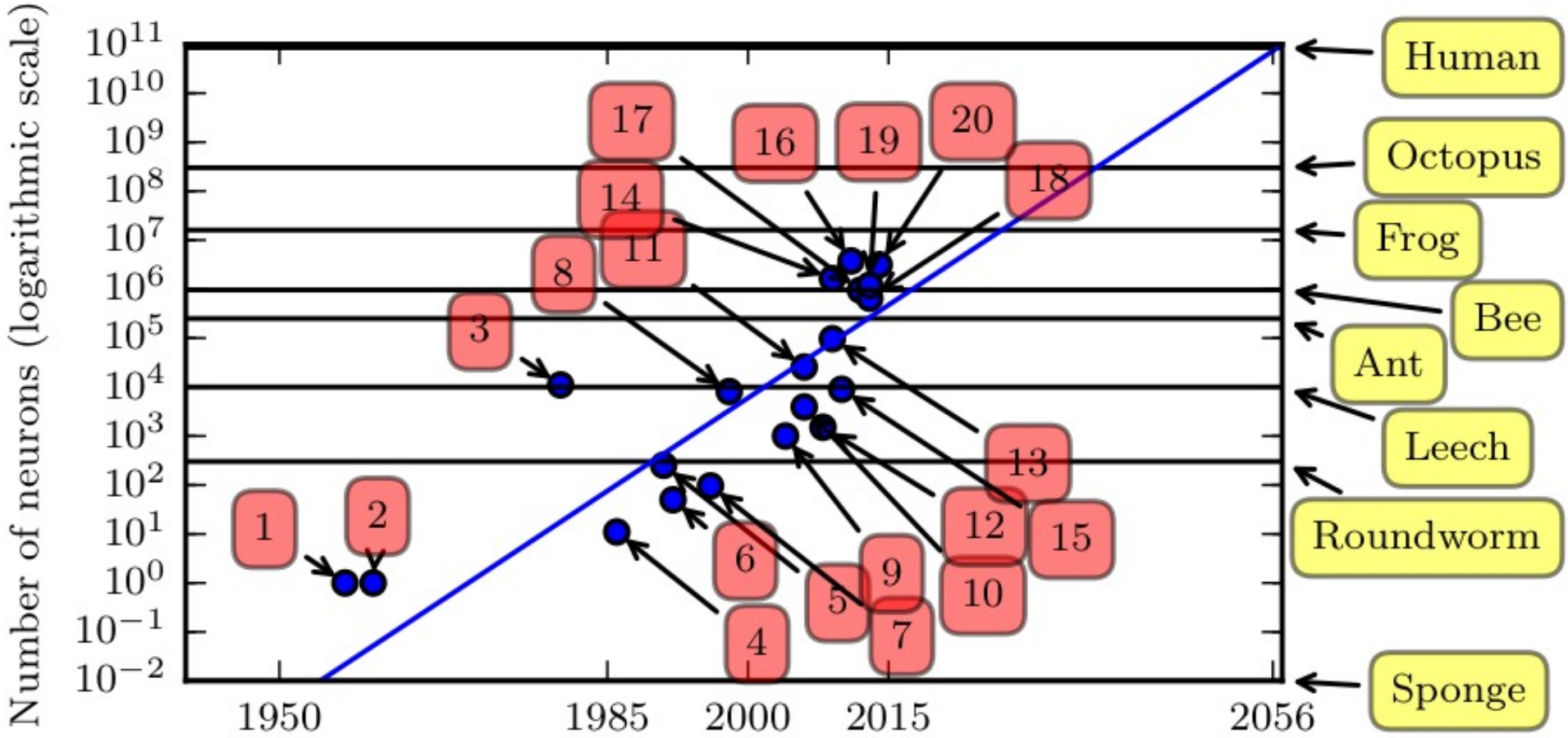


Historical Trends:
Growing Connections per Neuron



9: COTS HPC unsupervised CNN (Coates *et al.*, 2013)
10: GoogLeNet (Szegedy *et al.*, 2014)

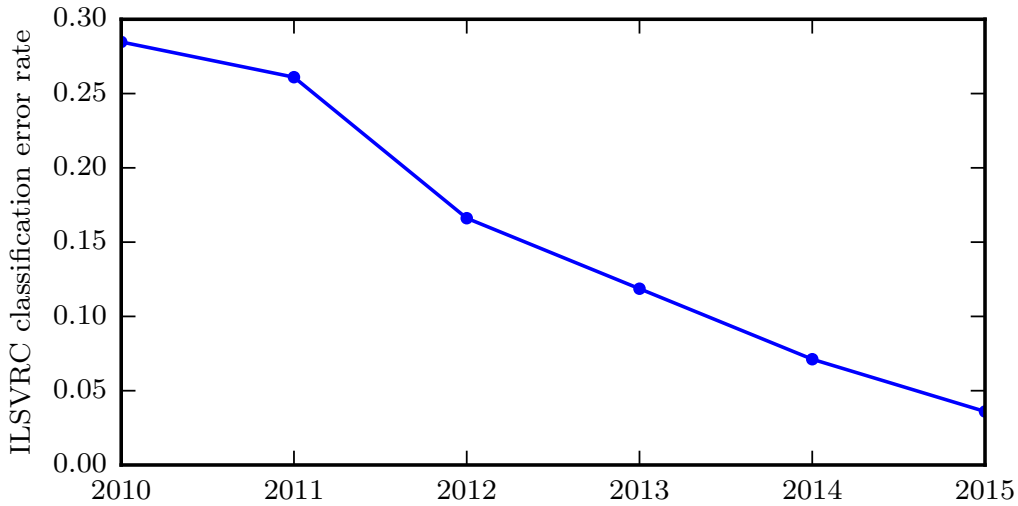
Historical Trends:
Growing Number of Neurons

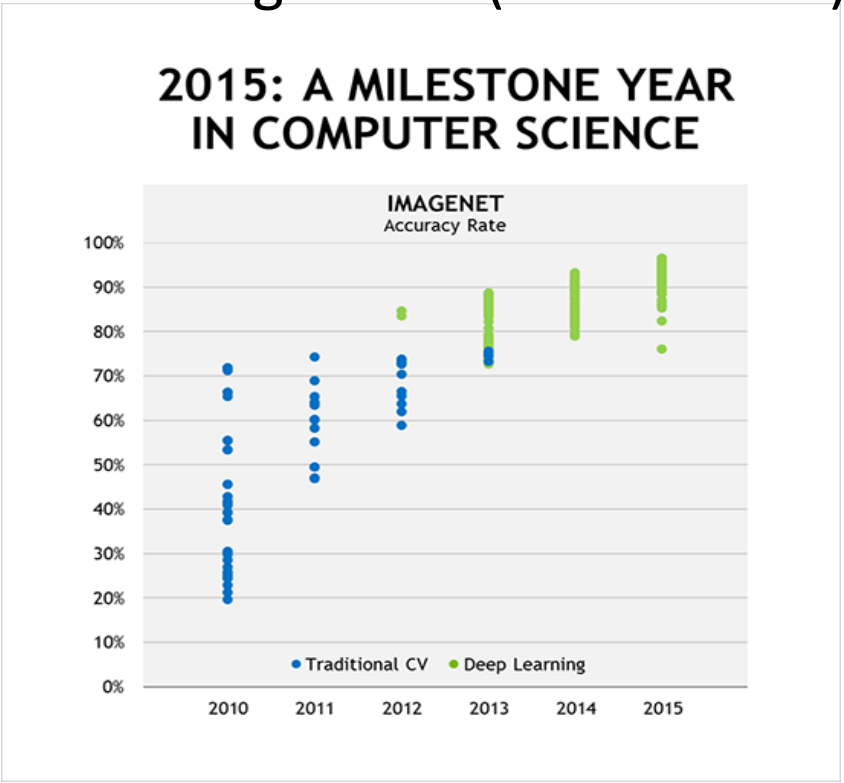


9: Echo state network (Jaeger and Haas, 2004)
20: GoogLeNet (Szegedy *et al.*, 2014)

Historical Trends: **Increasing Accuracy**

- DL has solved increasingly complicated applications with increasing accuracy:
  - Image Recognition and Object Recognition:
    - Large Scale Visual Recognition Challenge 2017 (ILSVRC2017)





Deep Learning x GPUs

- What is the relation between DL and GPUs?

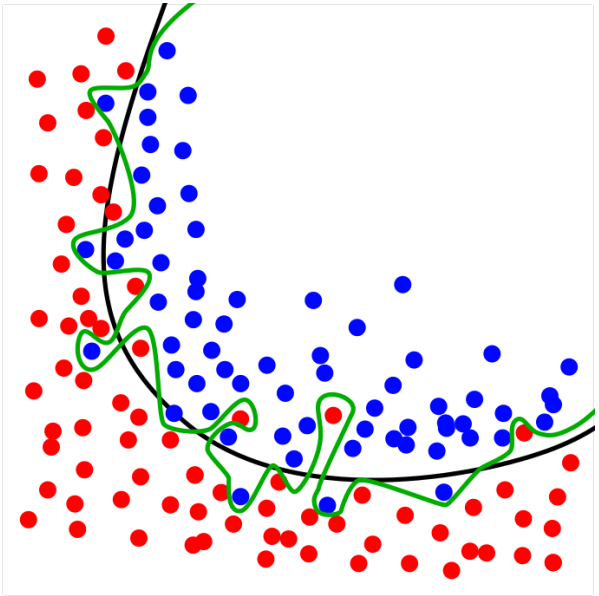
  Deep Learning & GPU:

  – Matrix operations

  – Parallel computing paradigm

Problems with DNN (Deep MLP)

- **Overfitting**:
  - The more layers you have, the more degrees of freedom you have.
  - DNNs model rare dependencies in the training data.
- **Diffusion of Gradient**: error attenuates as it propagates to early layers.
  - **Early layers never learn!**



https://en.wikipedia.org/wiki/Deep_learning

Main Deep Learning Architectures

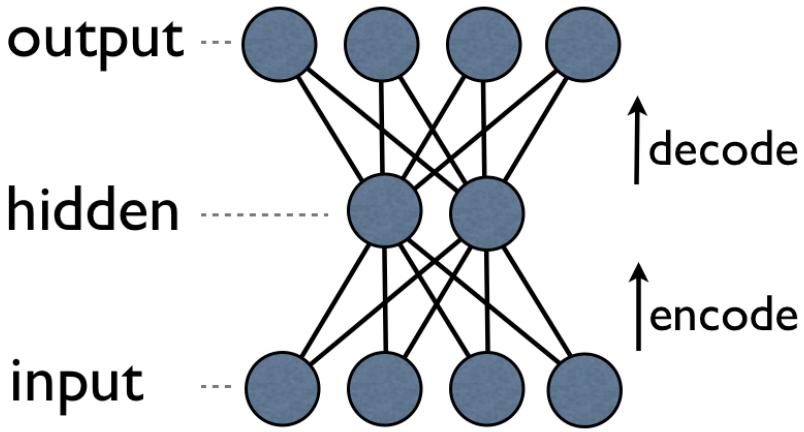
- Deep Belief Networks / Autoencoders
  - Greedy layer-wise pretraining, by Hinton *et al.*, 2006
- Deep Convolutional Neural Networks
  - La Net, by Le Cun et al, 1998.
- Deep Recurrent Networks
  - Long short-term Memory, by Sepp&Jurgen, 97.

Greedy layer-wise pretraining,
by Hinton *et al.*, 2006.

AUTOENCODERS

Autoencoders

- Autoencoders have just **one layer**.
- The aim of an autoencoder is to **learn a representation (encoding)** for a set of data, typically for the purpose of **dimensionality reduction**.



Autoencoders

- An auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to **reproduce the input**.
- By making this happen with (many) fewer units than the inputs, this forces the 'hidden layer' units to become **good feature detectors**.

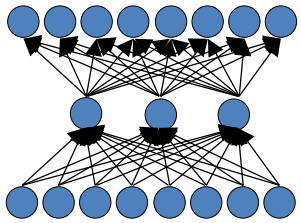
https://www.macs.hw.ac.uk/~dwcorne/Teaching/introdl.ppt

Representation learning (hidden layer)

1 0 0
0 1 1

Entrada		Saída hs		Saída
10000000	→	.89 .04 .08	→	10000000
01000000	→	.15 .99 .99	→	01000000
00100000	→	.01 .97 .27	→	00100000
00010000	→	.99 .97 .71	→	00010000
00001000	→	.03 .05 .02	→	00001000
00000100	→	.01 .11 .88	→	00000100
00000010	→	.80 .01 .98	→	00000010
00000001	→	.60 .94 .01	→	00000001

Intermediate representation: "discover" binary code !!!!

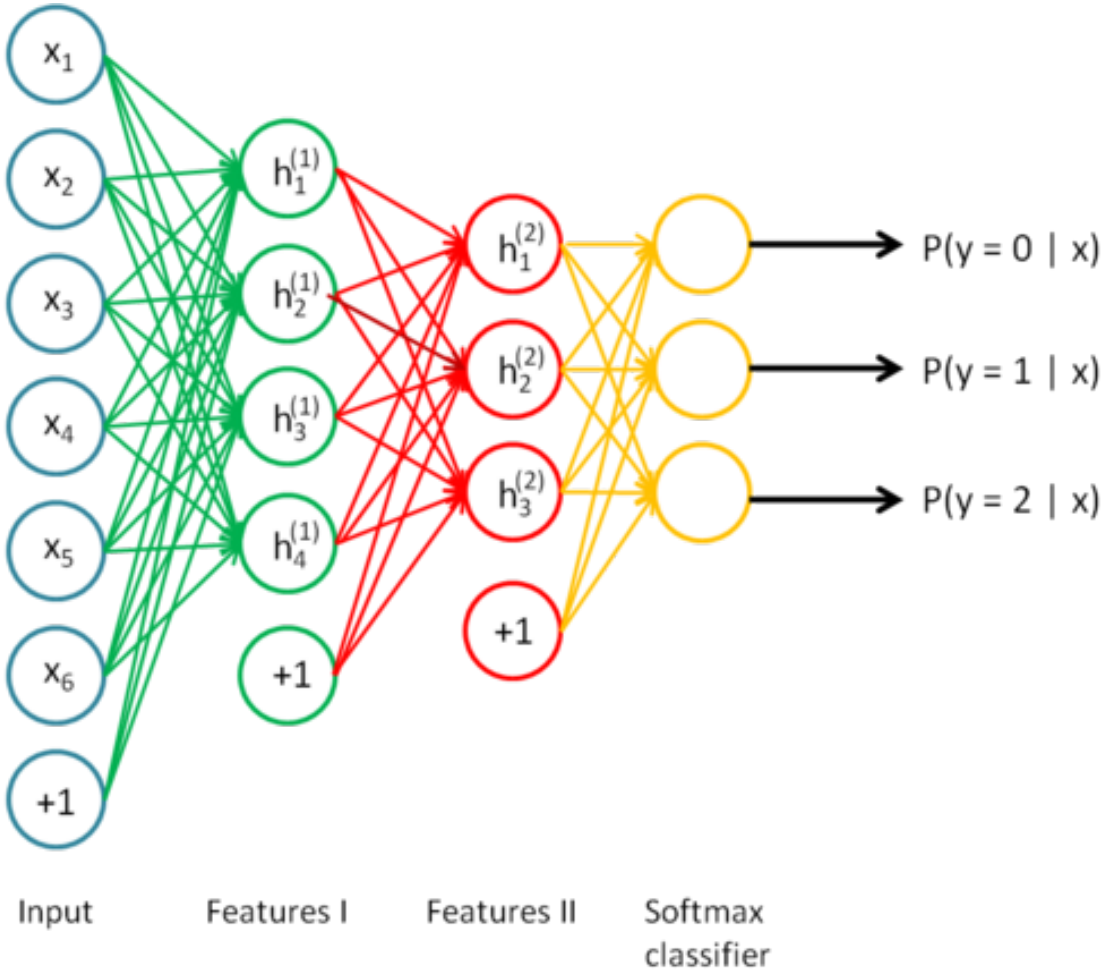


Greedy Layer-Wise Training

Geoffrey E. Hinton and Simon Osindero andYee-Whye Teh. A fast learning algorithm for deep belief nets, 2006

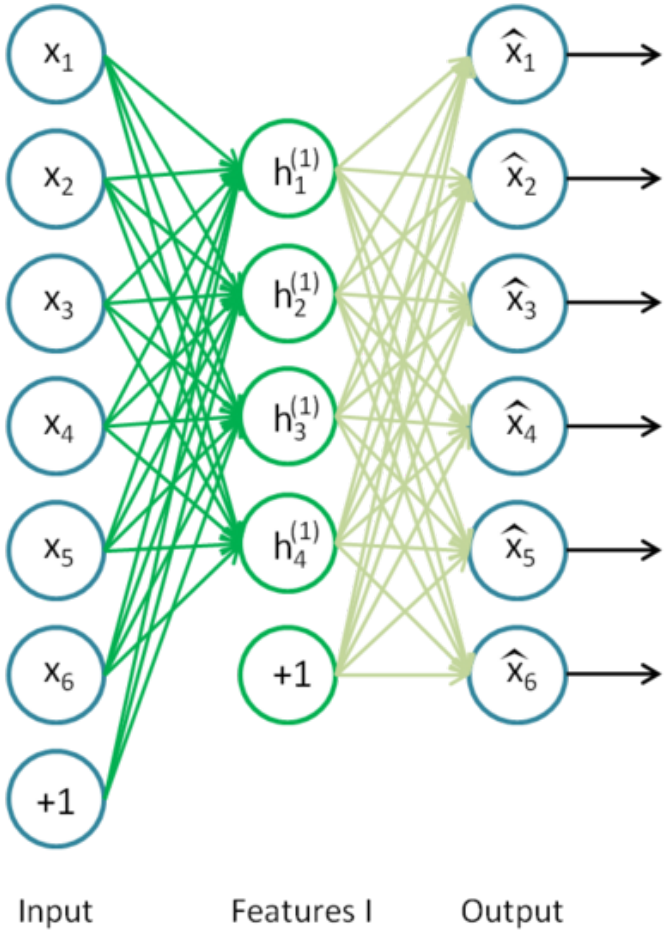
1. Train first layer using your data without the labels.
2. Then freeze the first layer parameters and start training the second layer using the output of the first layer as the input to the second layer.
3. Repeat this for as many layers as desired:
   - This builds our set of robust features
4. Use the outputs of the final layer as inputs to a supervised layer/model and train the last supervised layer(s) (leave early weights frozen)
5. Unfreeze all weights and fine tune the full network by training with a supervised approach, given the pre-processed weight settings.

How to train this network?



Advanced Machine Learning and Neural Networks, Tony Martinez
http://axon.cs.byu.edu/~martinez/classes/678/
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

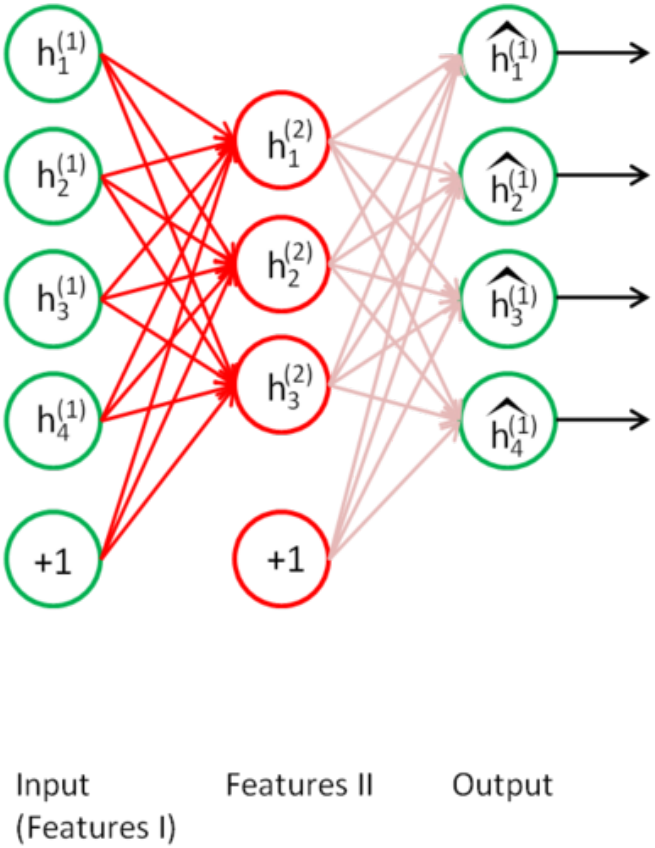
How to train this network?



Autoencoder in the first layer

Advanced Machine Learning and Neural Networks, Tony Martinez
http://axon.cs.byu.edu/~martinez/classes/678/
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

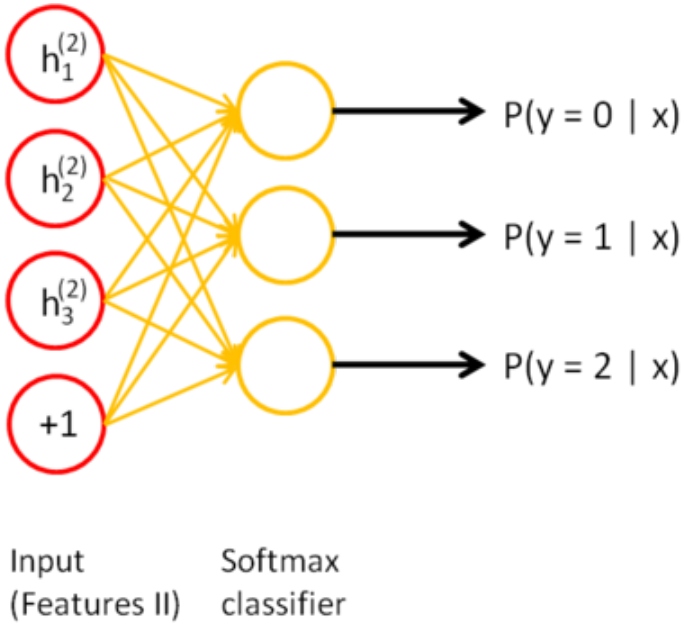
How to train this network?



Autoencoder in the second layer

Advanced Machine Learning and Neural Networks, Tony Martinez
http://axon.cs.byu.edu/~martinez/classes/678/
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

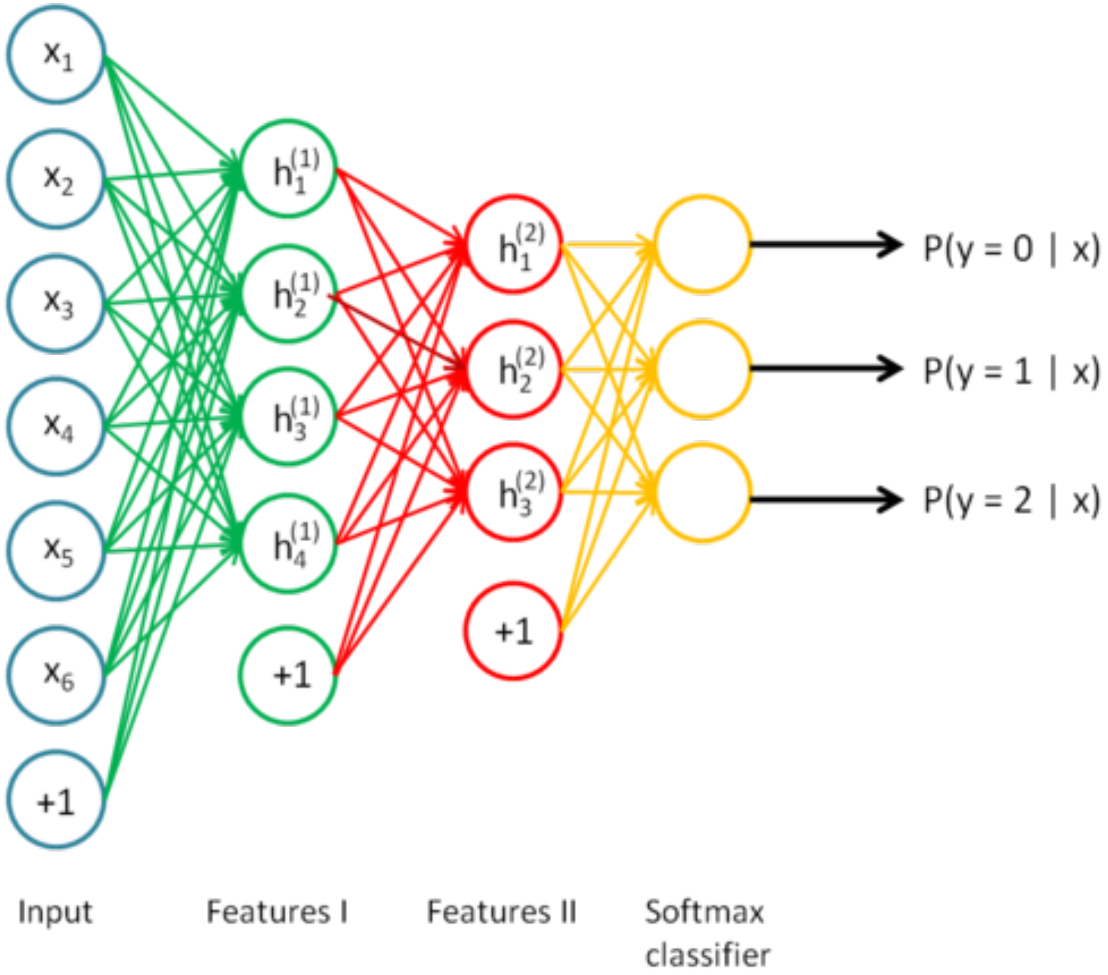
How to train this network?



Supervised learning in the last layer

Advanced Machine Learning and Neural Networks, Tony Martinez
http://axon.cs.byu.edu/~martinez/classes/678/
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

How to train this network?



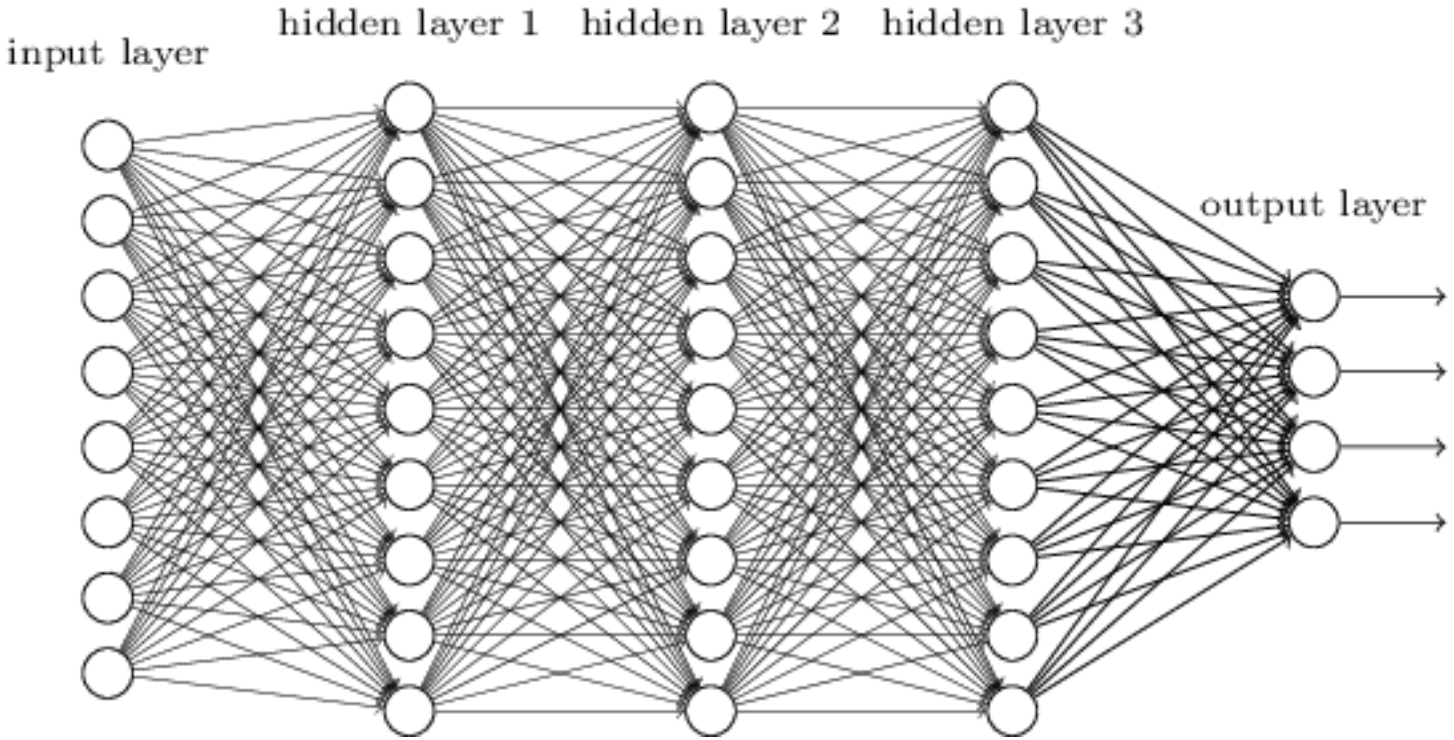
Fine tune the full NN with supervised learning

Advanced Machine Learning and Neural Networks, Tony Martinez
http://axon.cs.byu.edu/~martinez/classes/678/
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

CONVOLUTIONAL NEURAL NETWORKS

CNN

- We know it is good to learn a small model.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?

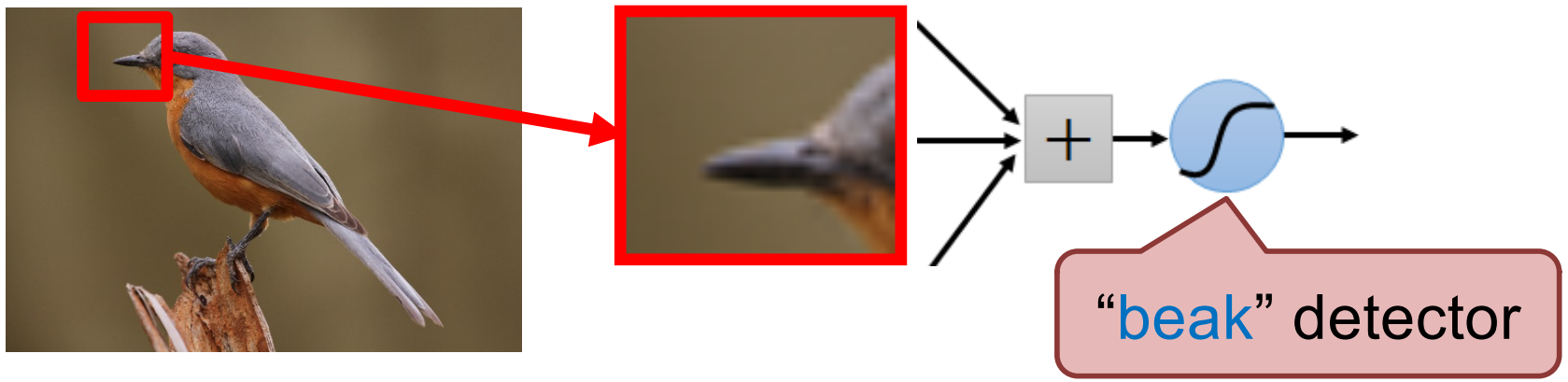


https://cs.uwaterloo.ca/~mli/cs898-2017.html

Consider learning an image:

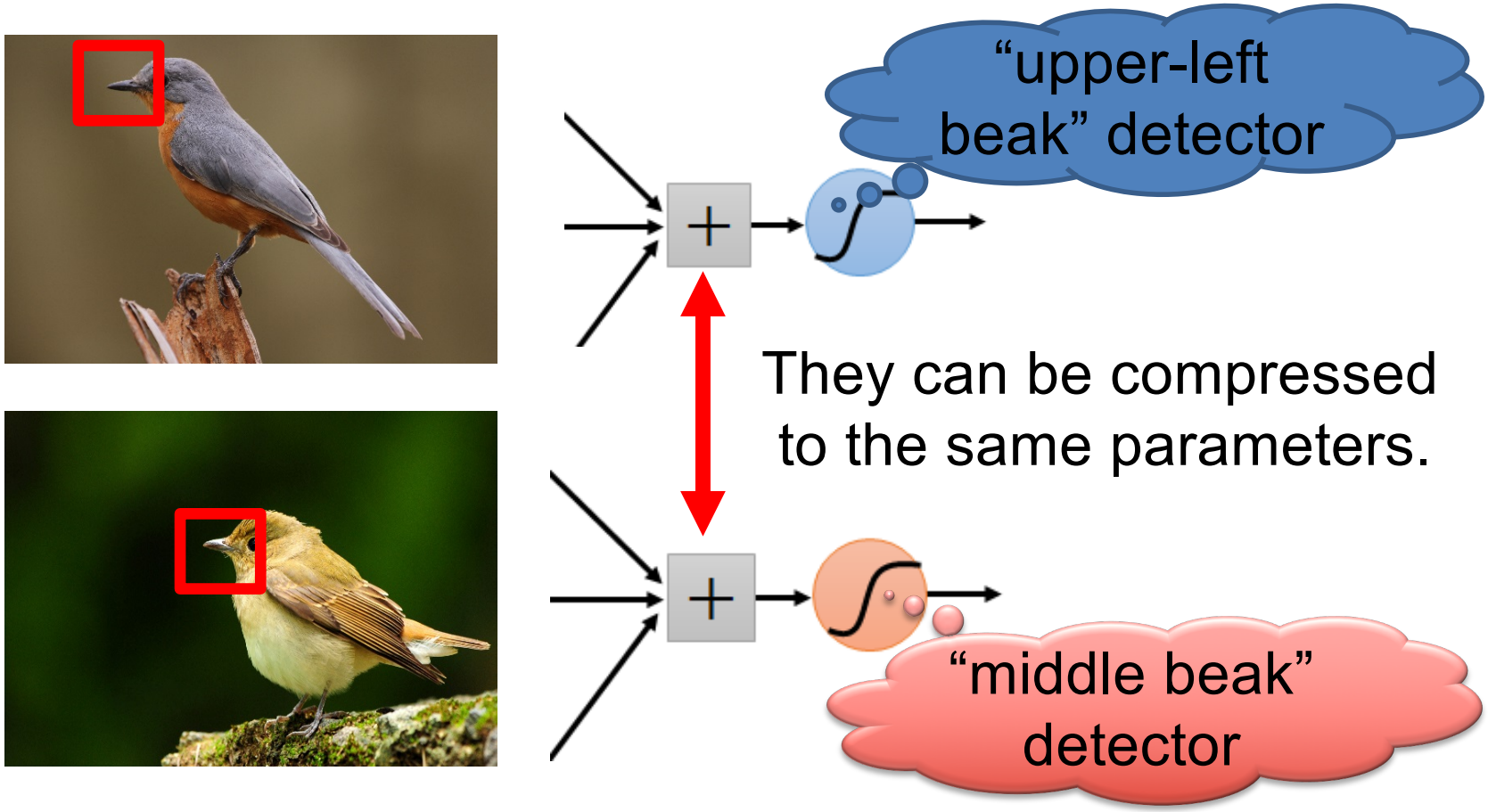
- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters



https://cs.uwaterloo.ca/~mli/cs898-2017.html

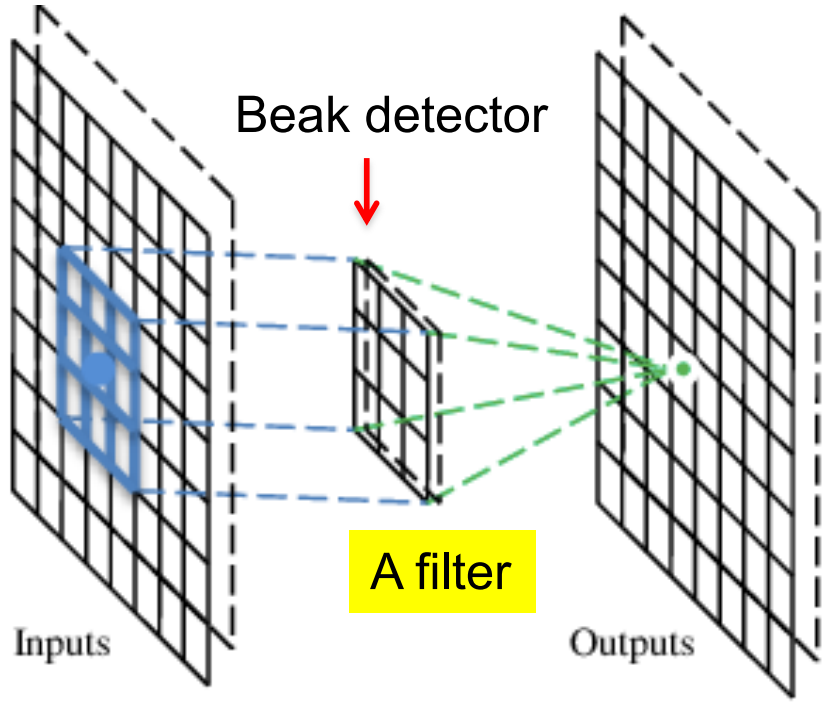
Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors
and each detector must “move around”.



https://cs.uwaterloo.ca/~mli/cs898-2017.html

A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



https://cs.uwaterloo.ca/~mli/cs898-2017.html

Convolution

These are the network parameters to be learned.

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

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

https://cs.uwaterloo.ca/~mli/cs898-2017.html

Convolutio

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

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

Dot product

3 -1

6 x 6 image

https://cs.uwaterloo.ca/~mli/cs898-2017.html

Convolutio

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

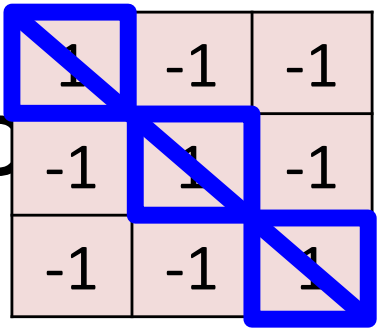
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

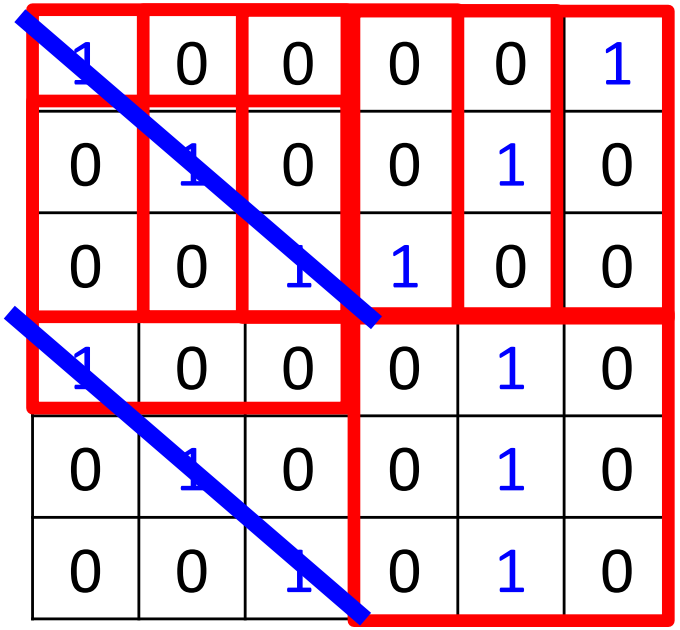
https://cs.uwaterloo.ca/~mli/cs898-2017.html

Convolutio

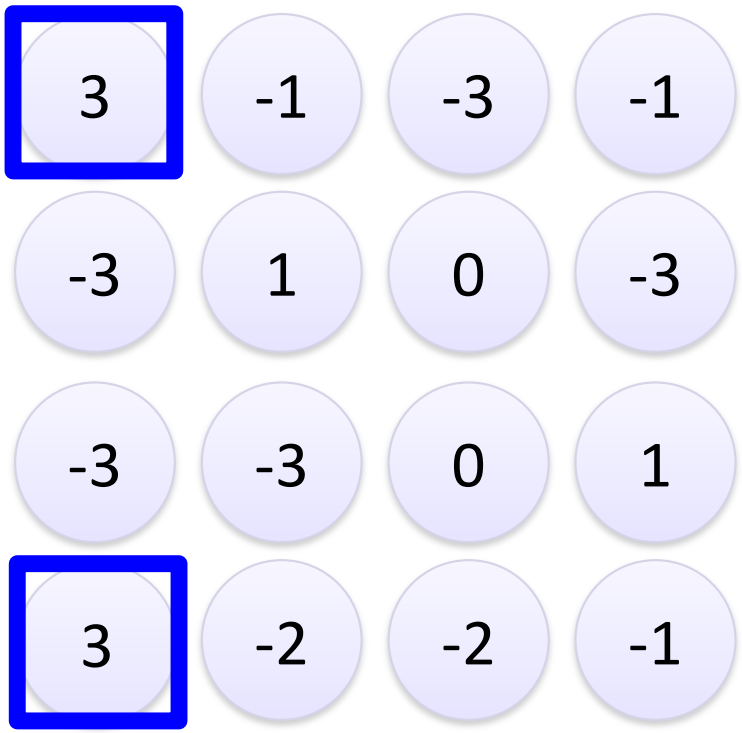


Filter 1

stride=1



6 x 6 image



https://cs.uwaterloo.ca/~mli/cs898-2017.html

Convolutio

-1	1	-1
-1	1	-1
-1	1	-1

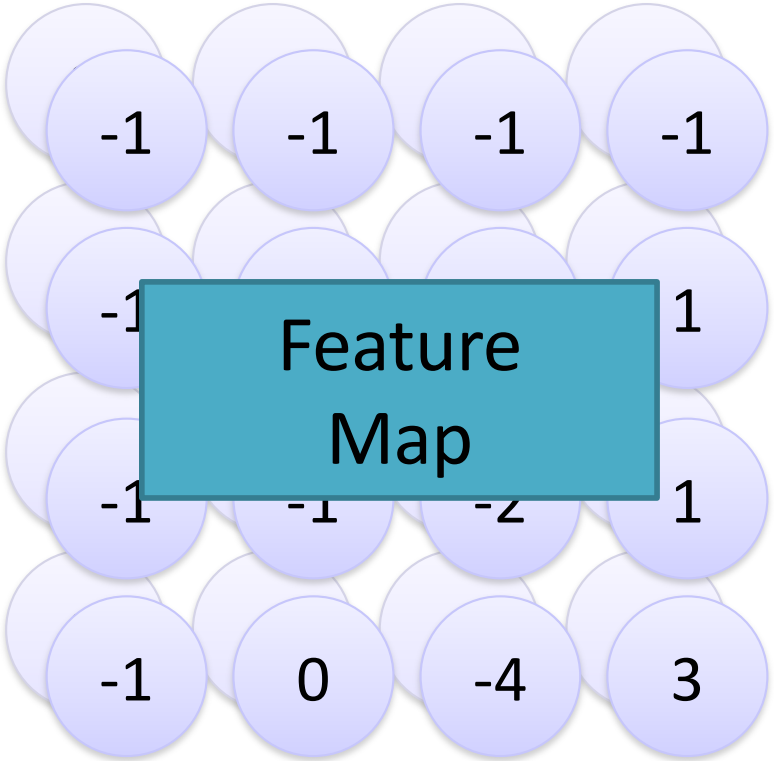
Filter 2

stride=1

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

6 x 6 image

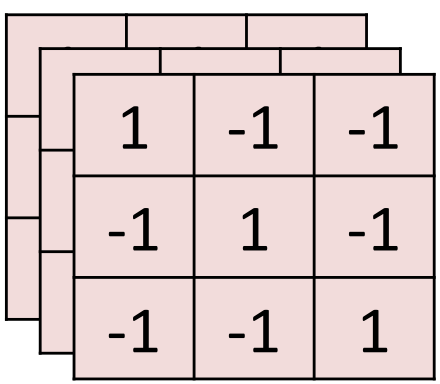
Repeat this for each filter



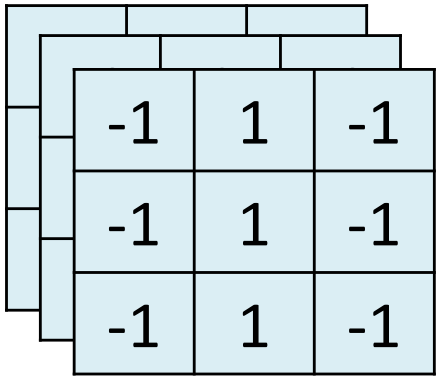
Two 4 x 4 images
Forming 2 x 4 x 4 matrix

https://cs.uwaterloo.ca/~mli/cs898-2017.html

Color image: RGB 3 channels

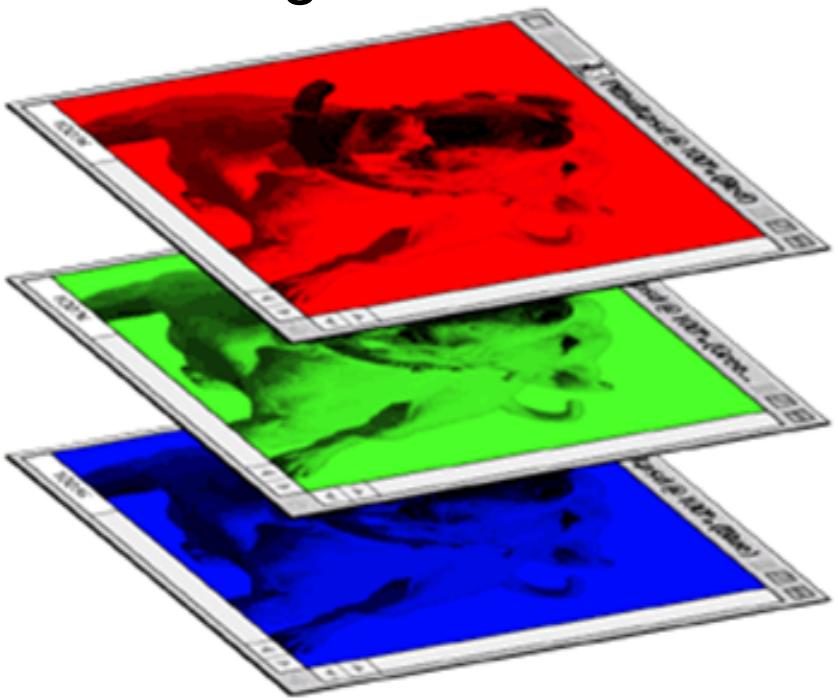


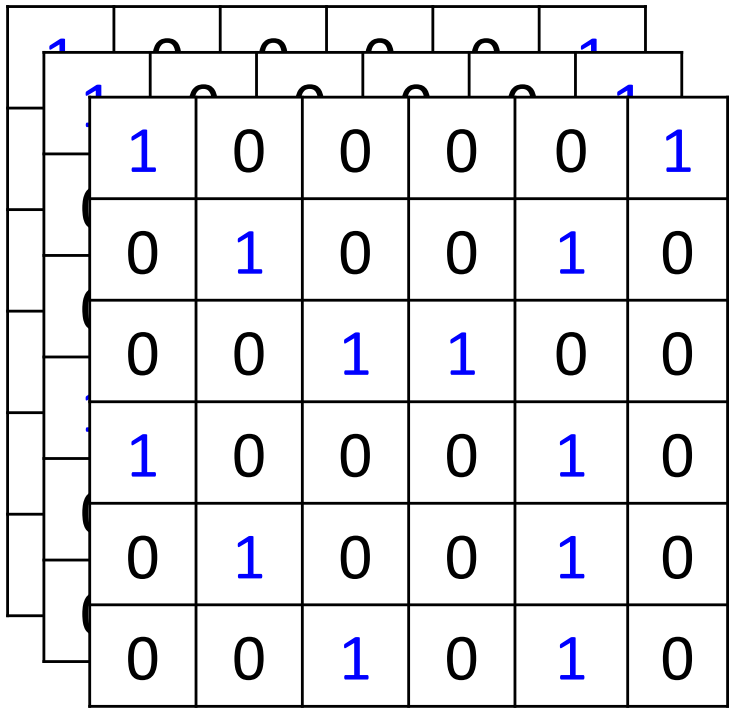
Filter 1



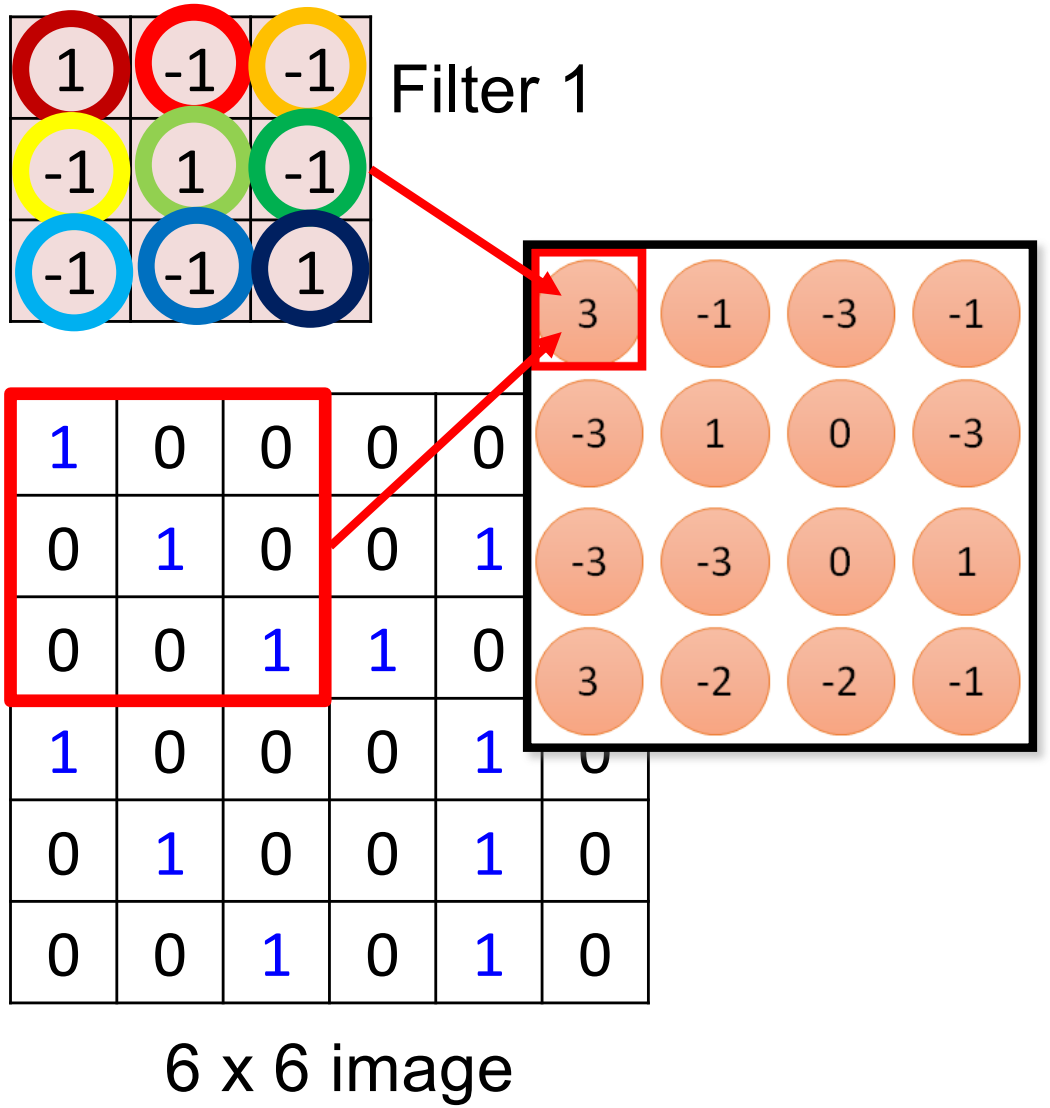
Filter 2

Color image

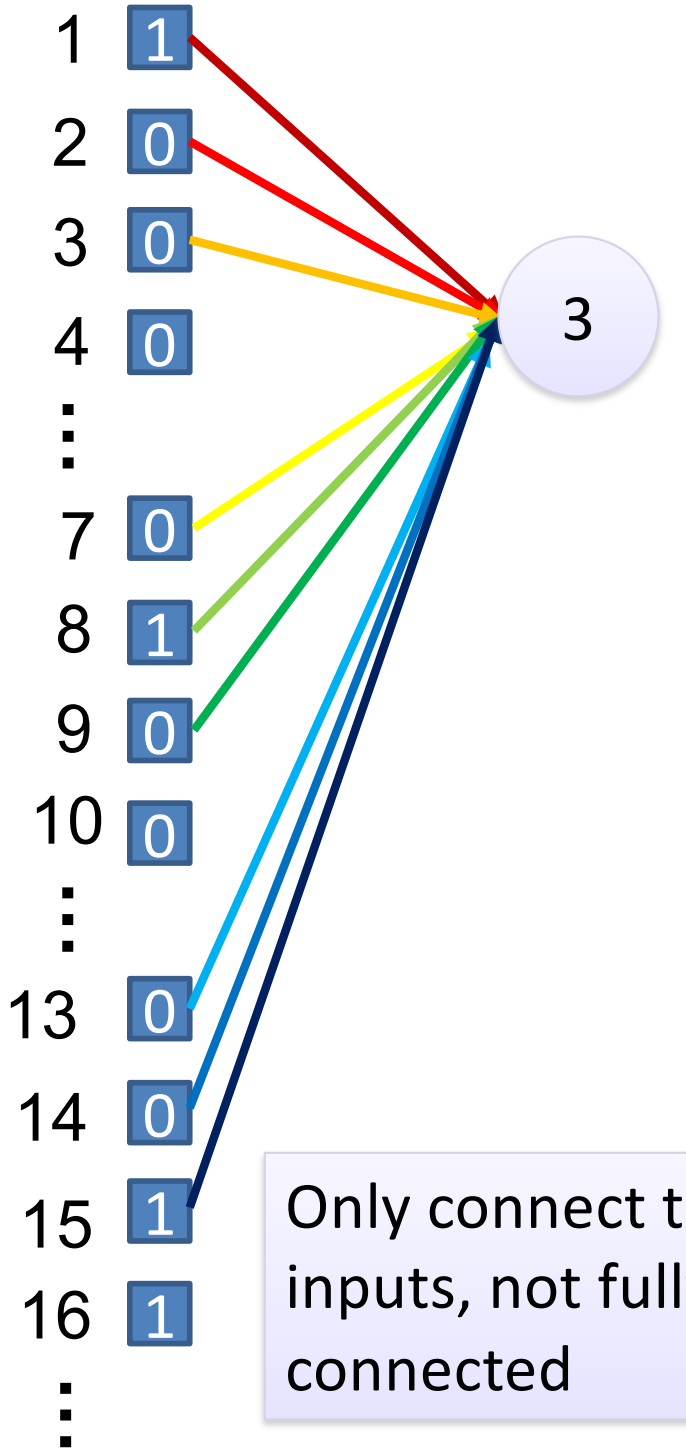




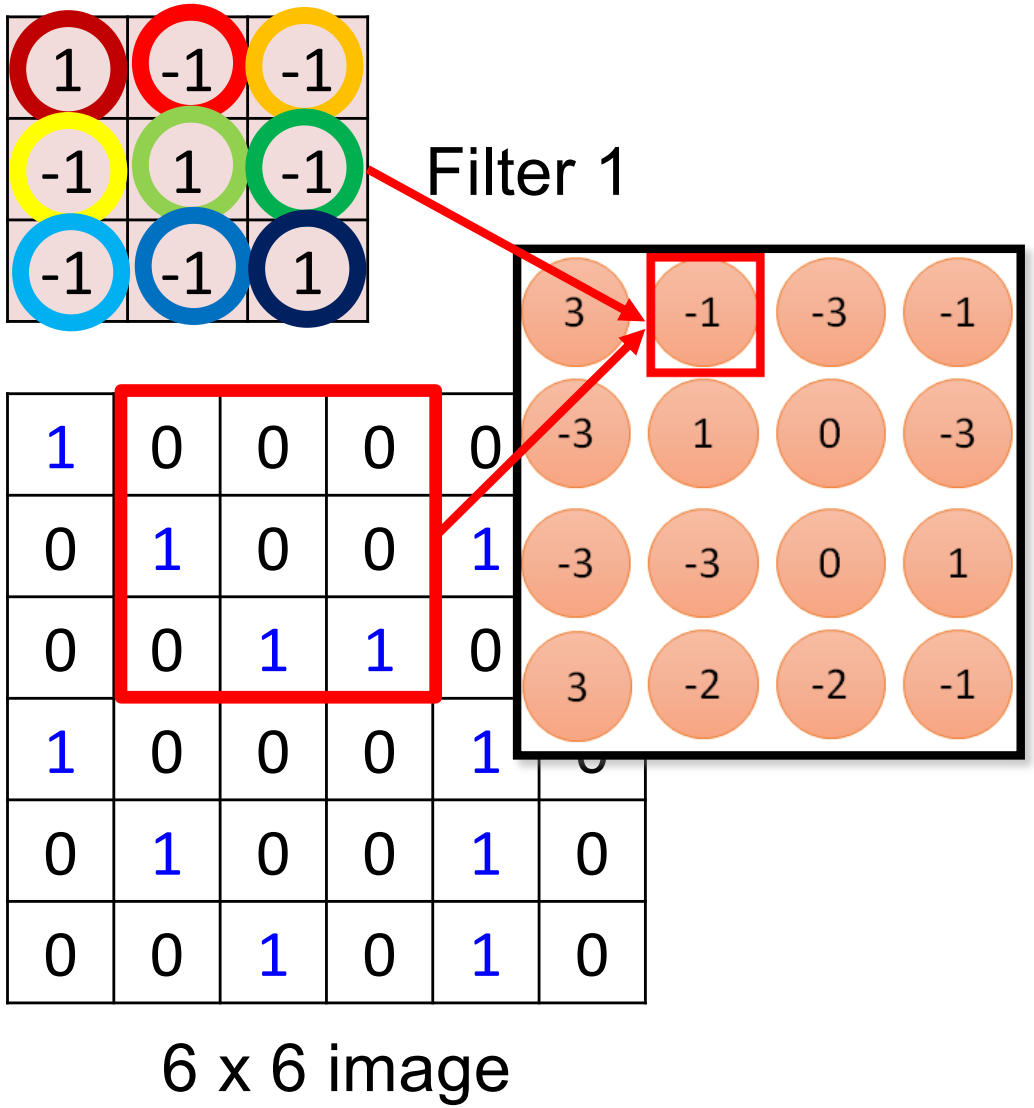
https://cs.uwaterloo.ca/~mli/cs898-2017.html



fewer parameters!

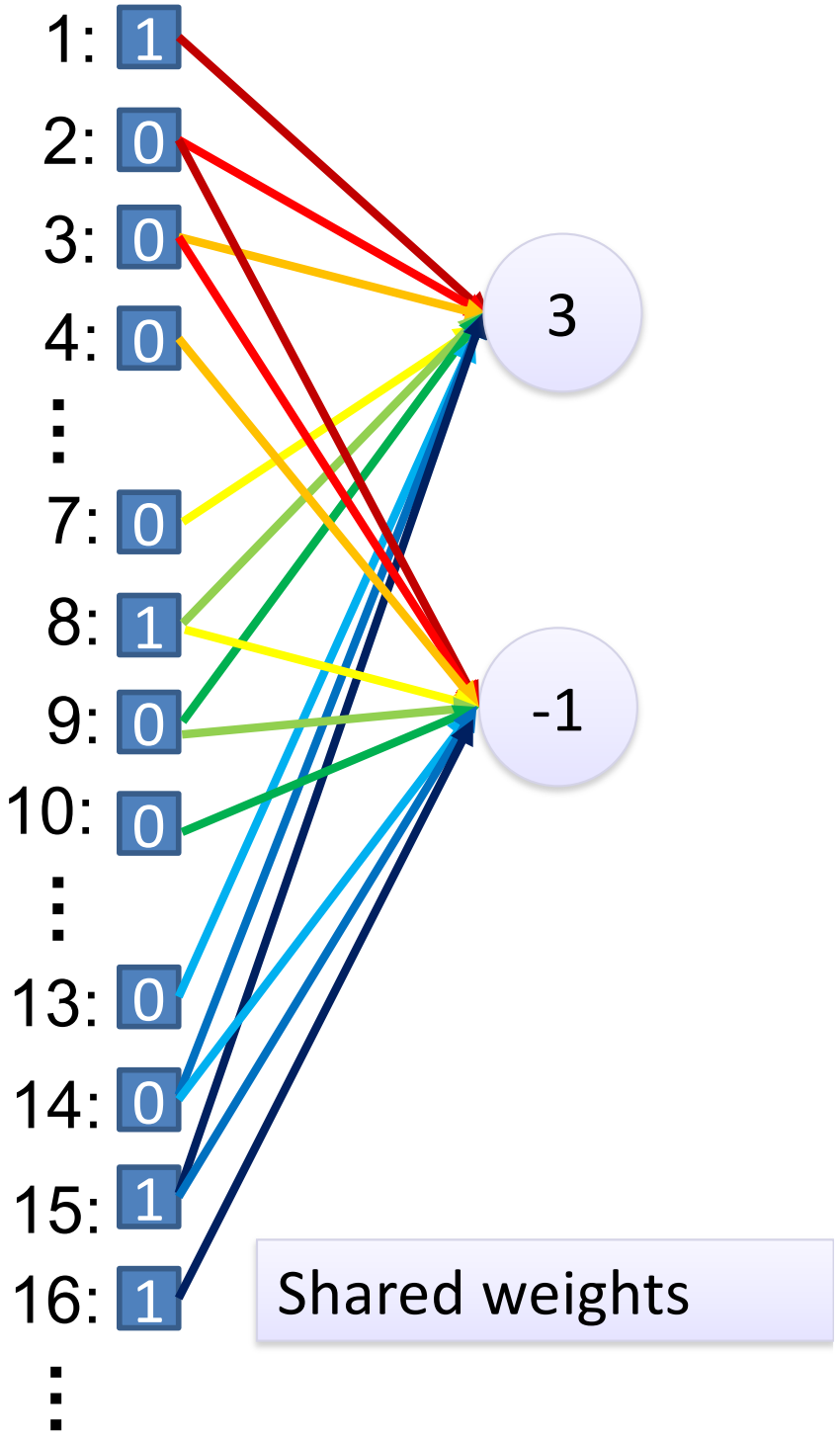


https://cs.uwaterloo.ca/~mli/cs898-2017.html



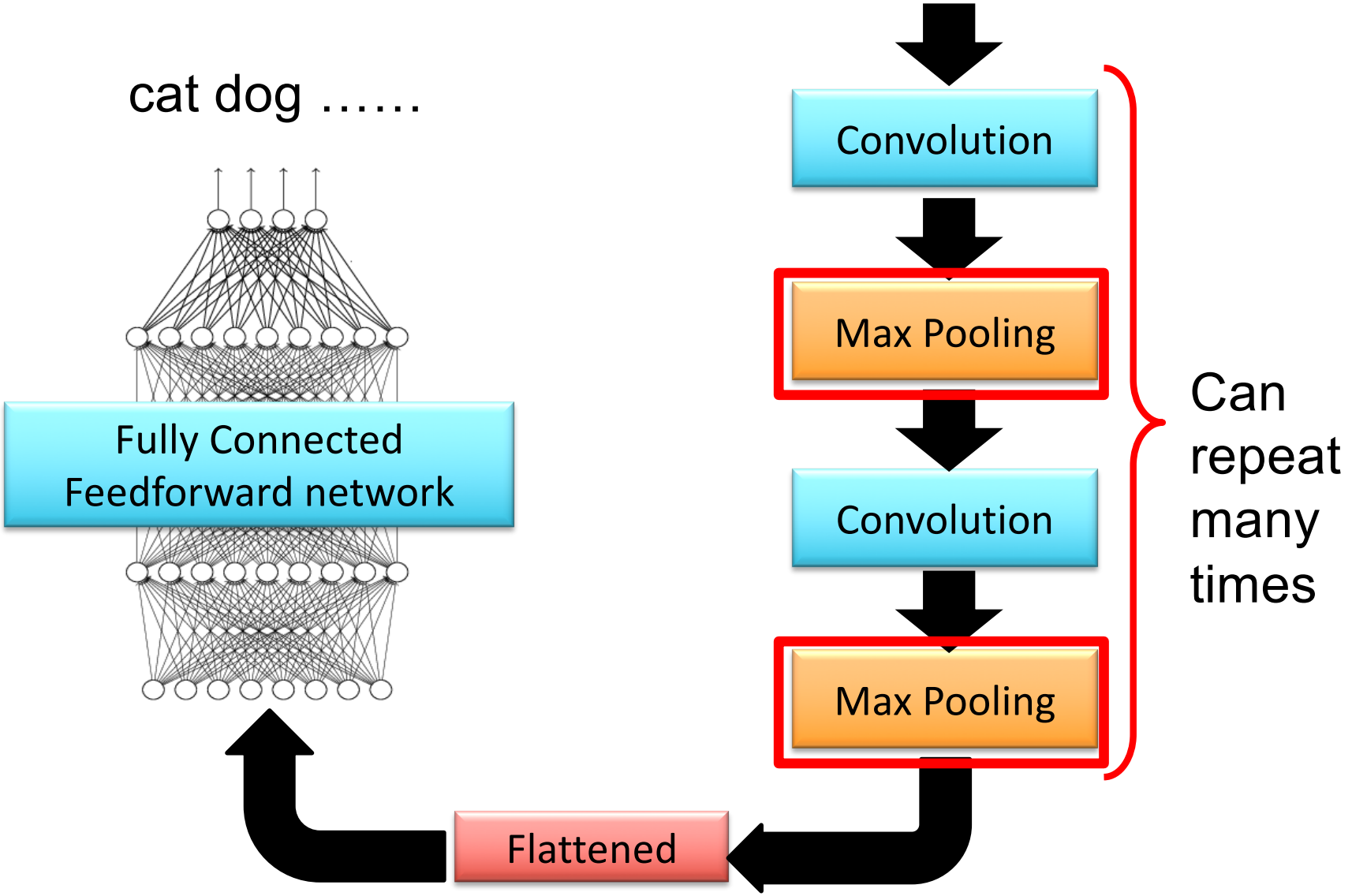
Fewer parameters

Even fewer parameters



https://cs.uwaterloo.ca/~mli/cs898-2017.html

The whole



https://cs.uwaterloo.ca/~mli/cs898-2017.html

Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

3	-1
-3	1

-3	-1
0	-3

-3	-3
3	-2

0	1
-2	-1

-1	-1
-1	-1

-1	-1
-2	1

-1	-1
-1	0

-2	1
-4	3

https://cs.uwaterloo.ca/~mli/cs898-2017.html

Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

fewer parameters to characterize the image

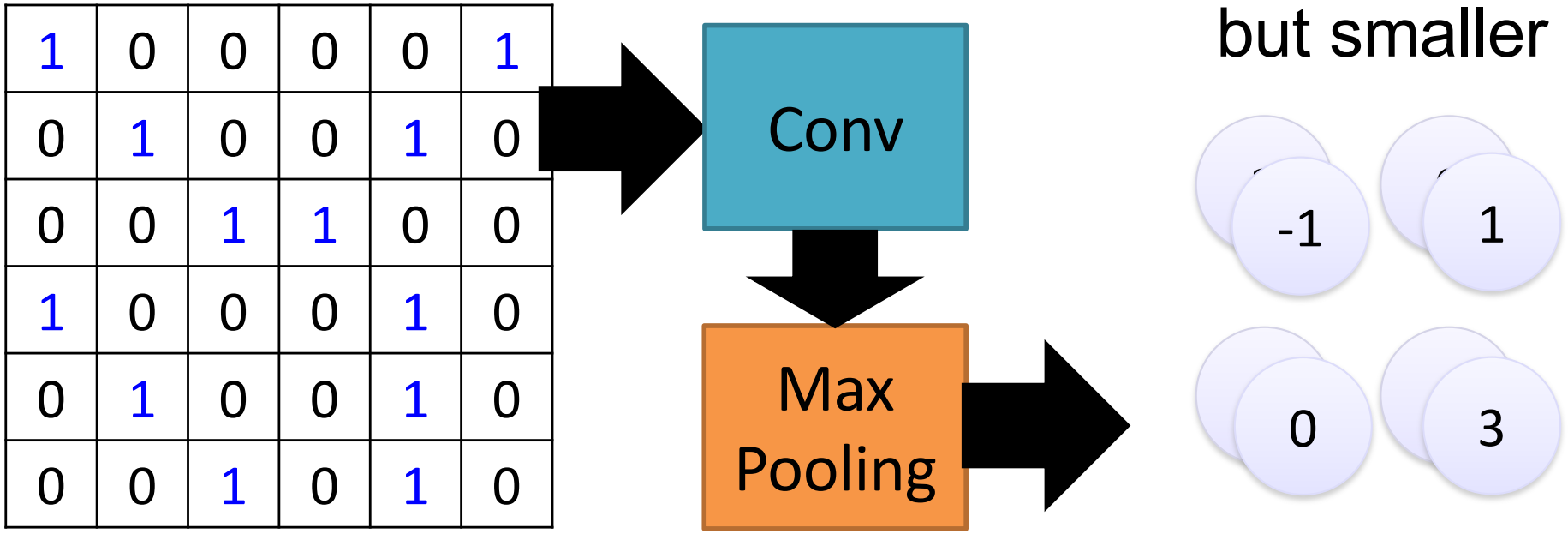
https://cs.uwaterloo.ca/~mli/cs898-2017.html

A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

https://cs.uwaterloo.ca/~mli/cs898-2017.html

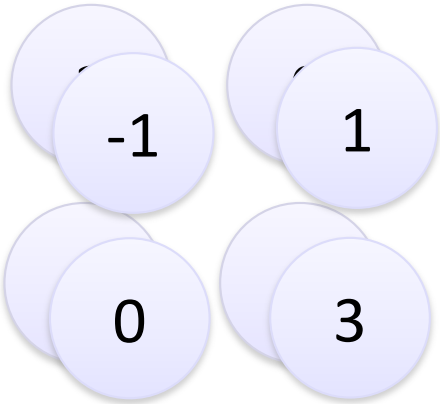
Max Pooling

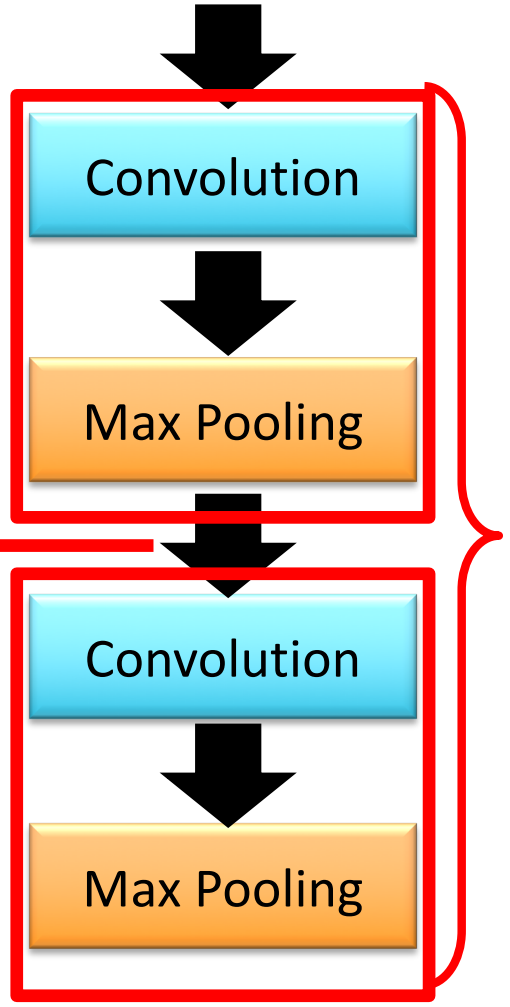


Each filter is a channel

https://cs.uwaterloo.ca/~mli/cs898-2017.html

The whole





Can repeat many times

Smaller than the original image

The number of channels is the number of filters

https://cs.uwaterloo.ca/~mli/cs898-2017.html

The whole

cat dog ……

Convolution

Max Pooling

A new image

Convolution

Max Pooling

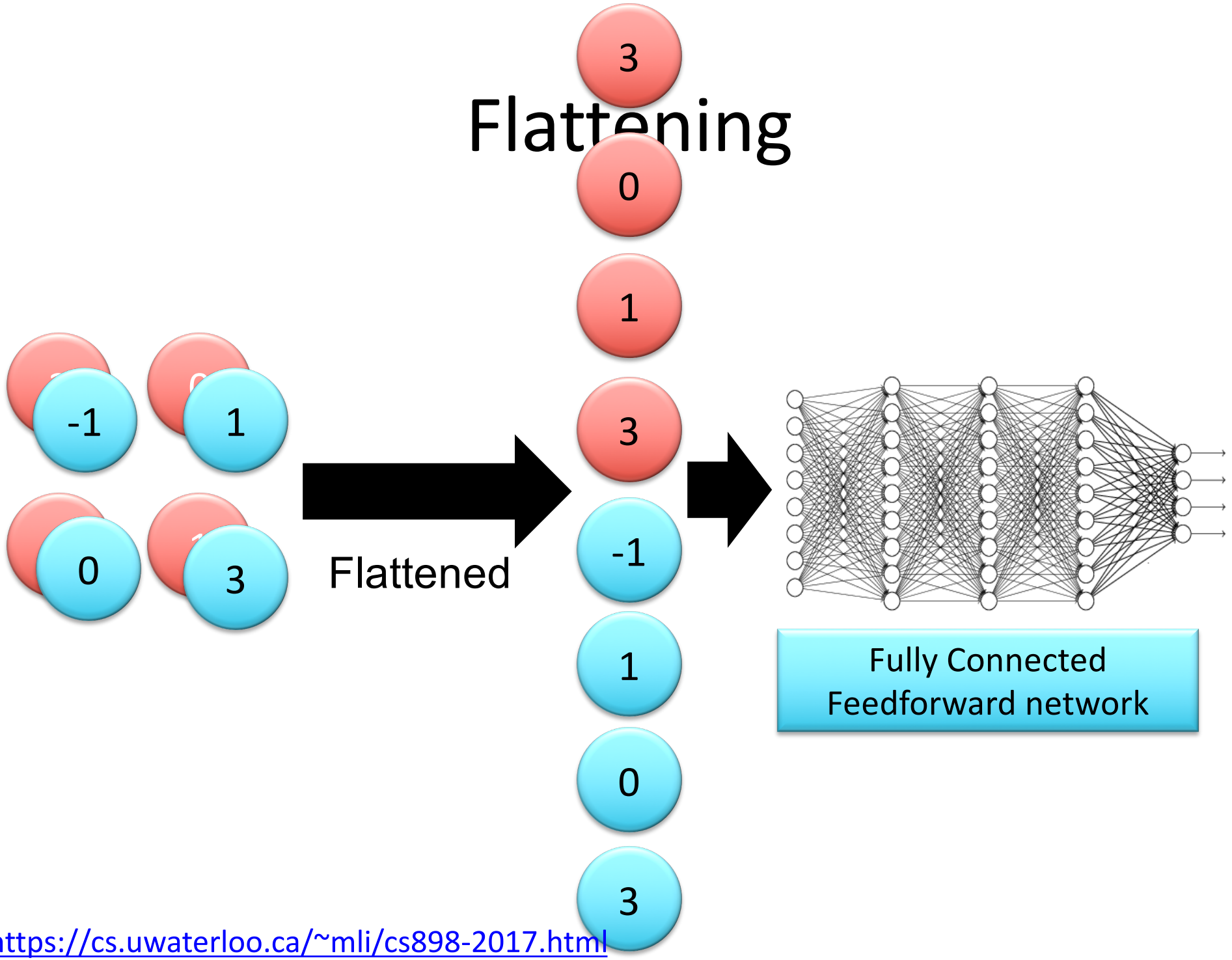
A new image

Flattened

Fully Connected Feedforward network

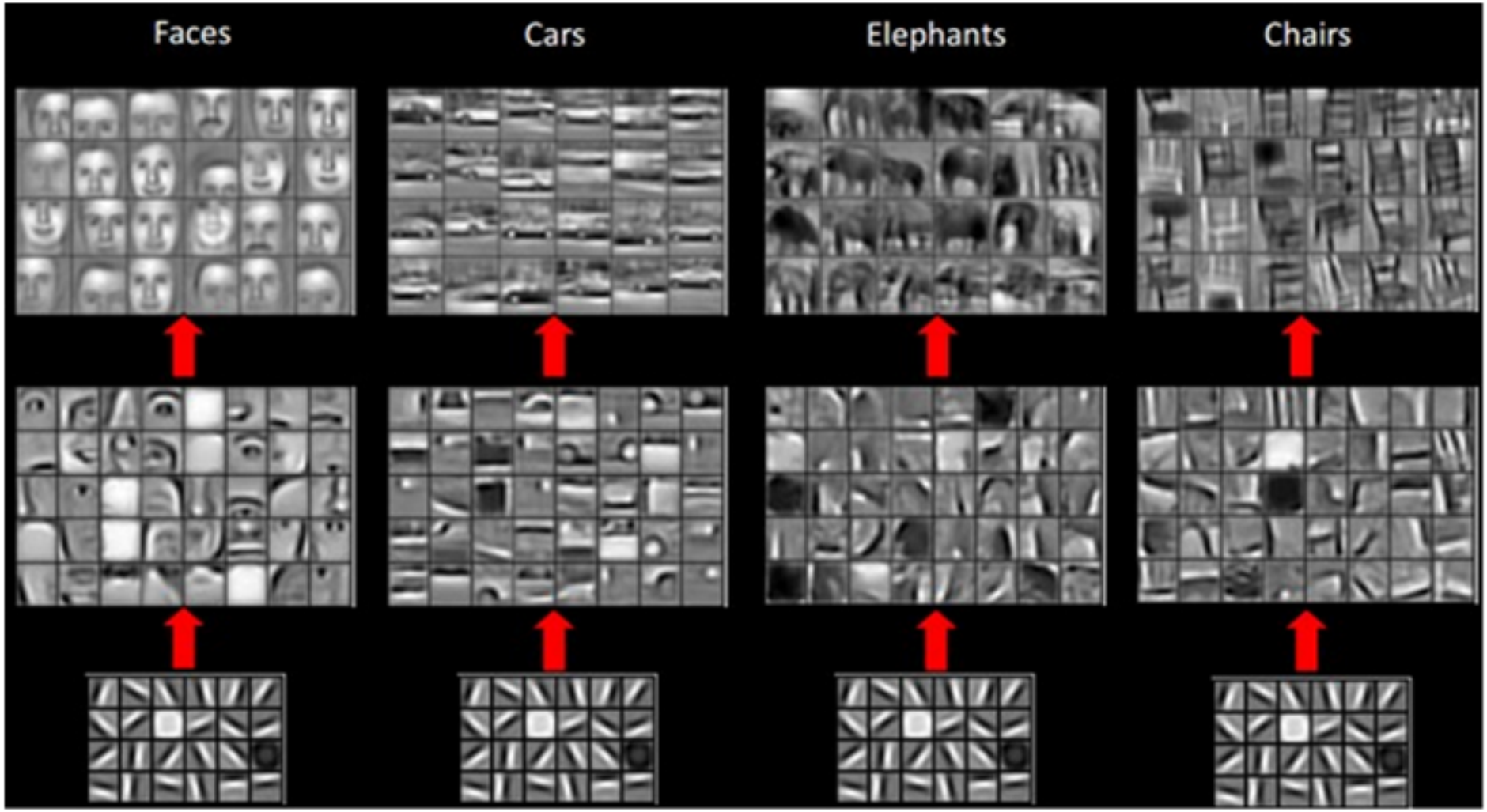
https://cs.uwaterloo.ca/~mli/cs898-2017.html

Flattening



https://cs.uwaterloo.ca/~mli/cs898-2017.html

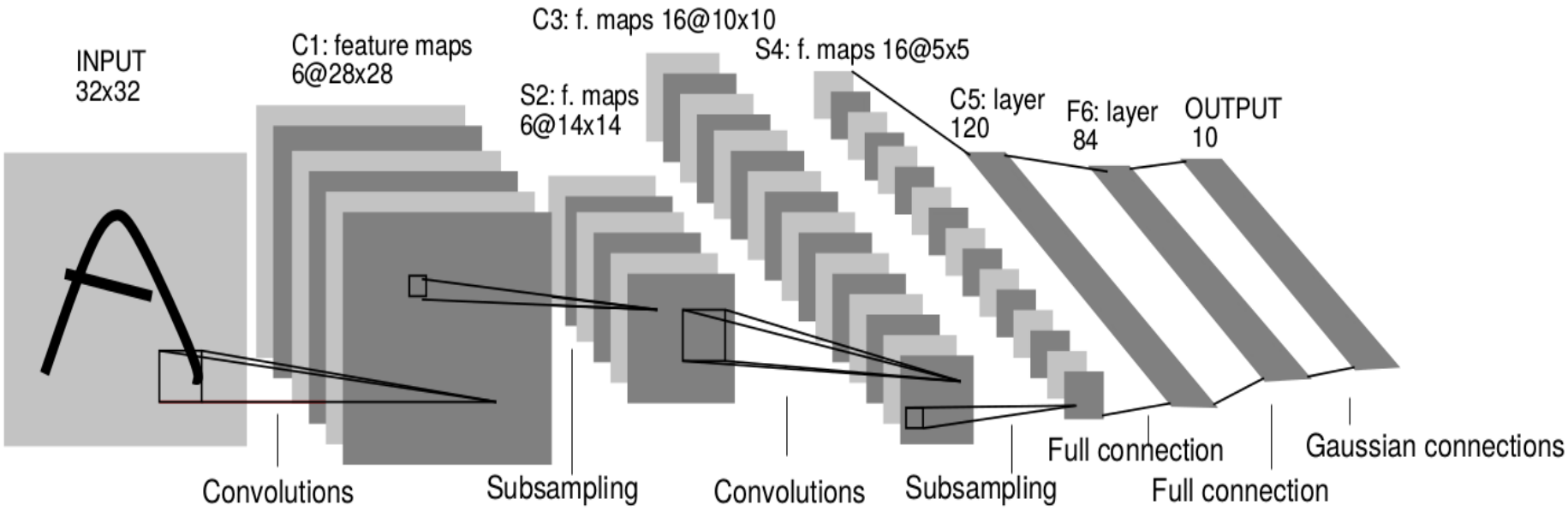
CNNs



http://stats.stackexchange.com/questions/146413/why-convolutional-neural-networks-belong-to-deep-learning

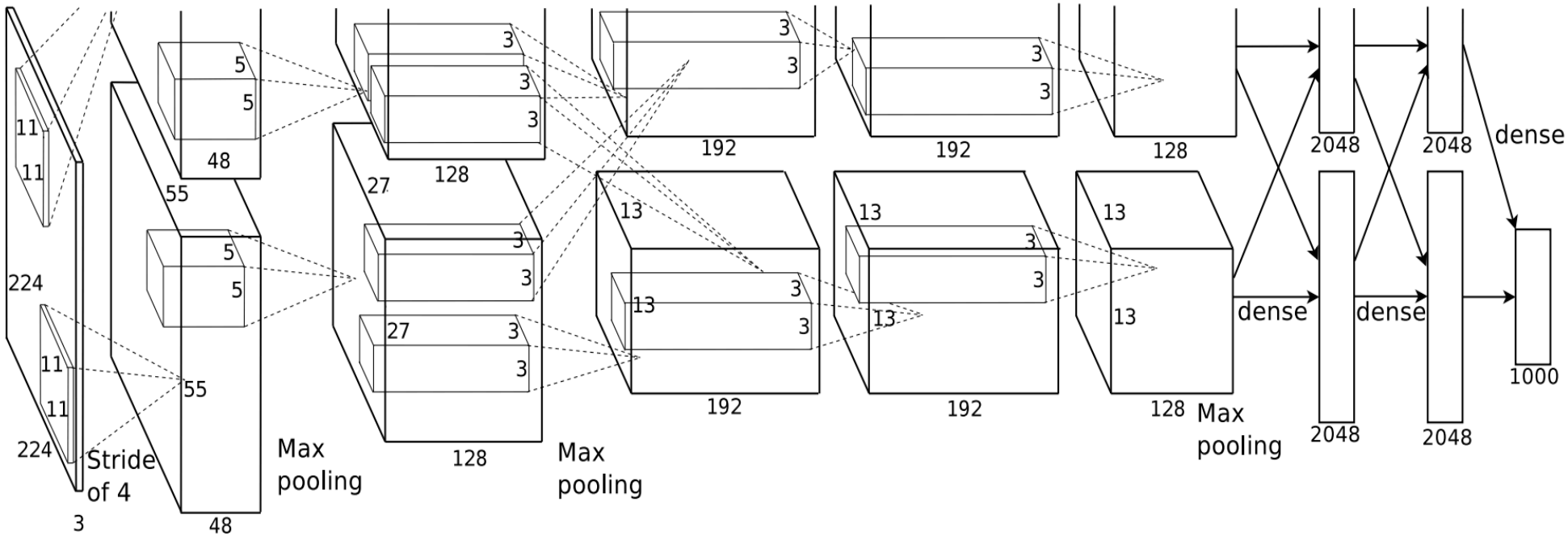
LeNet-5

LeCun, Bottou, Bengio & Haffner, 1998

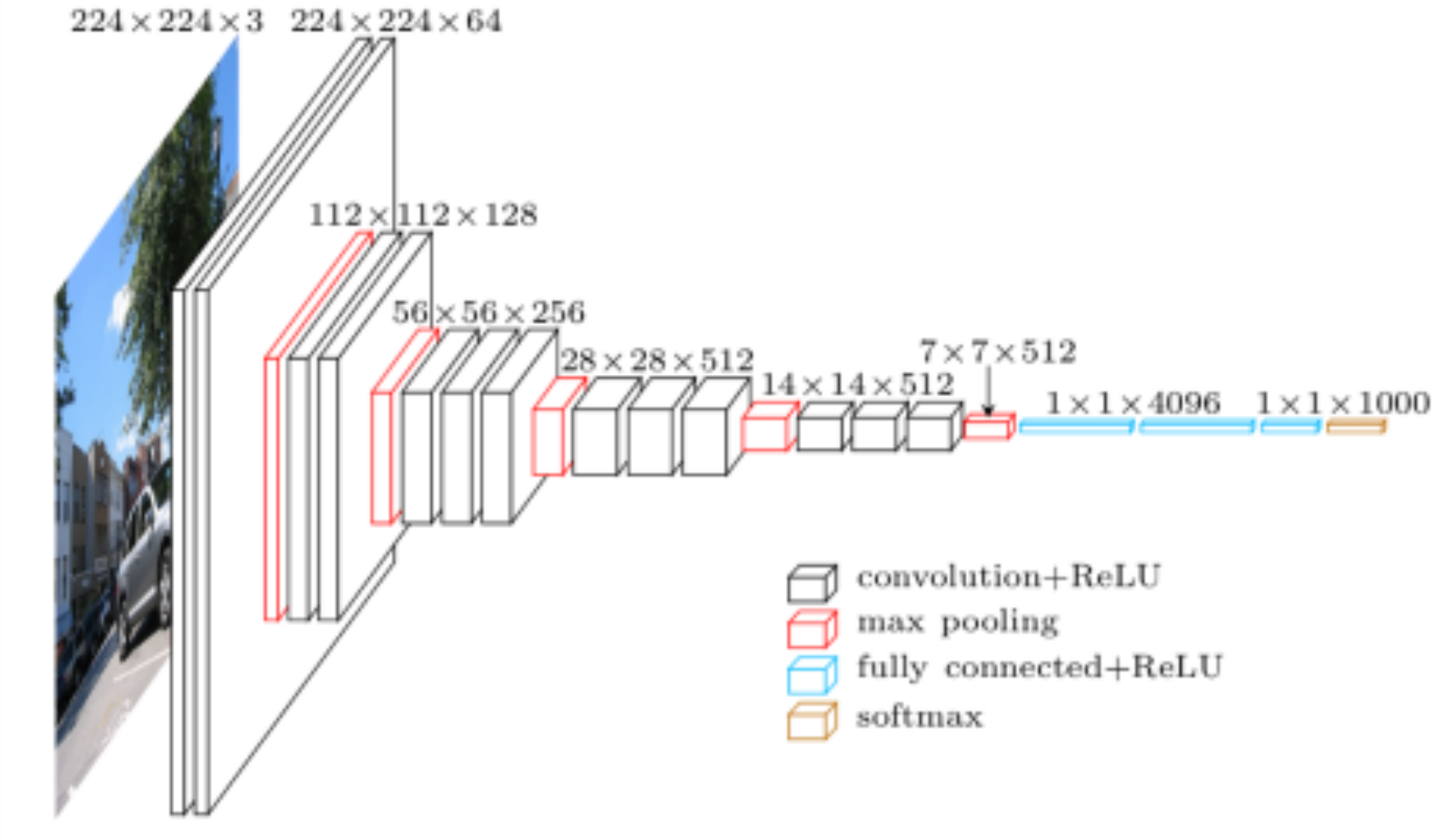


Alexnet

Krizhevsky, Sutskever & Hinton, 2012



VGG
Simonyan & Zisserman, 2014



https://arxiv.org/abs/1409.1556

DeepFace

Taigman, Yang, Ranzato &Wolf, 2014

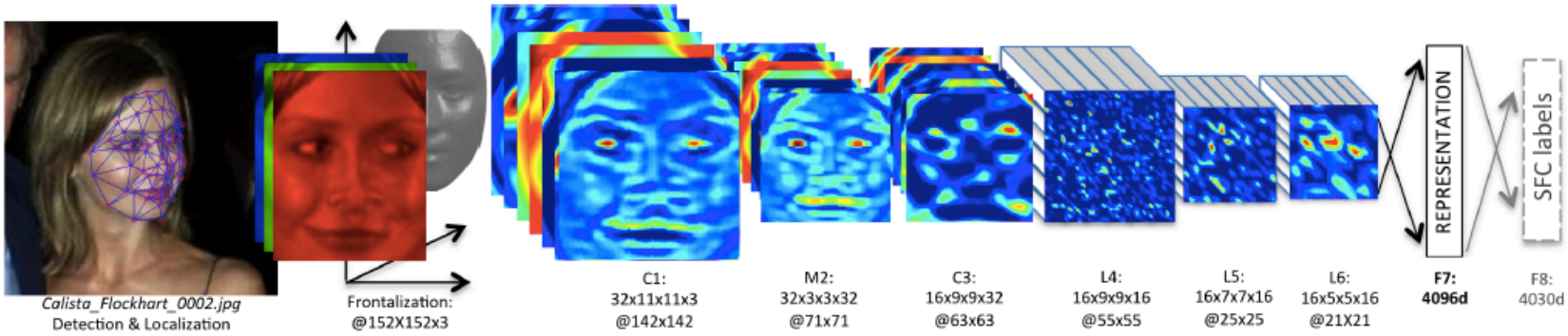
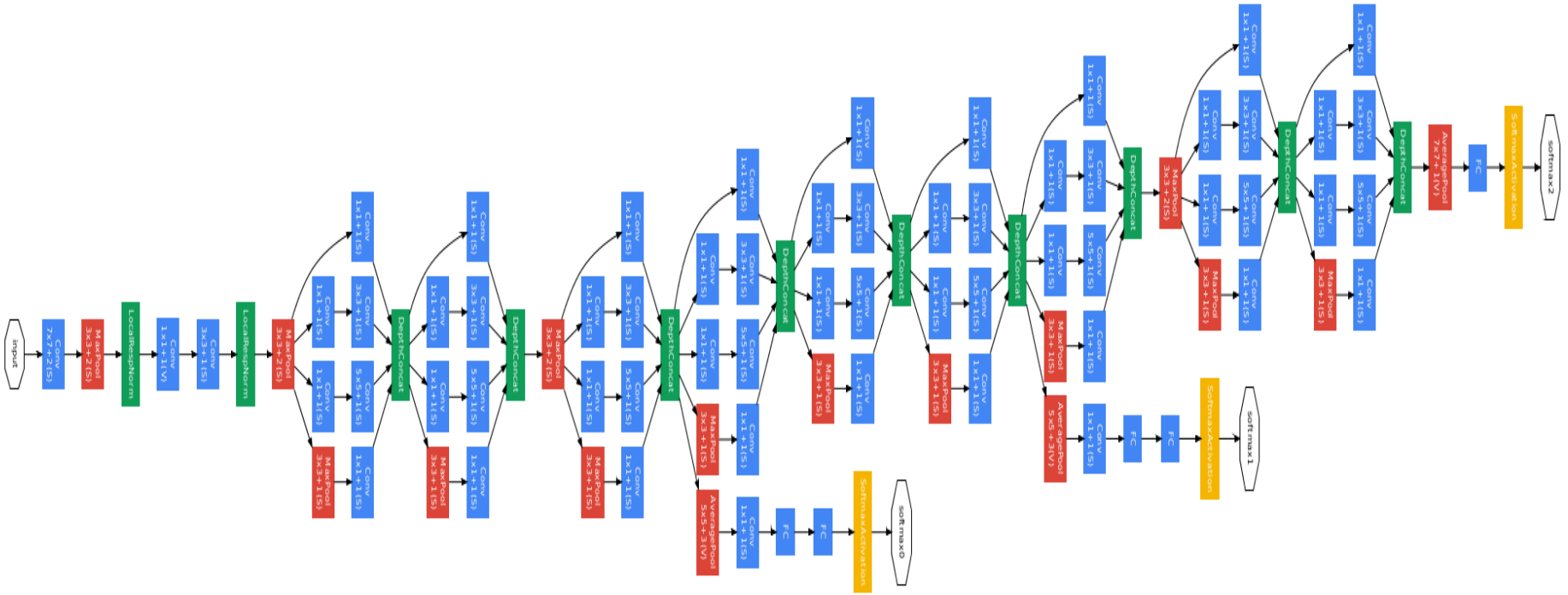


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

DeepFace: Closing the Gap to Human-Level Performance in Face Verification
https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/

GoogleLeNet
Szegedy et al, 2015.



http://research.google.com/pubs/pub43022.html

Conclusion

- CNN: Special purpose net – Just for images or problems with strong **grid-like local spatial/temporal correlation**
- Once trained on one problem could use same net (often tuned) for a new similar problem – general creator of vision features
- Autoenconder could be used to find initial parameters
- **Lots of hand crafting and tuning** to find the right recipe of receptive fields, layer interconnections, etc.
  - Lots more Hyperparameters than standard nets, and even than other deep networks, since the structures of CNNs are more handcrafted
  - CNNs getting wider and deeper with speed-up techniques (e.g. GPU, ReLU, etc.) and lots of current research, excitement, and success

Fully Supervised Deep Learning

- Much recent success in doing fully supervised deep learning with extensions which diminish the effect of early learning difficulties (unstable gradient, etc.)
- **Patience** (now that we know it may be worth it), **faster computers**, and use of **GPUs**
- More efficient activation functions (e.g. ReLUs) in terms of both computation and avoiding $f'(net)$ saturation

Open problems

- A More Scientific Approach is Needed, not Just Building Better Systems…
  - Geoff Hinton, Yoshua Bengio & Yann LeCun, NIPS 2015