3. MULTI-LAYER PERCEPTRONS

Stephan Robert-Nicoud
HEIG-VD/HES-SO

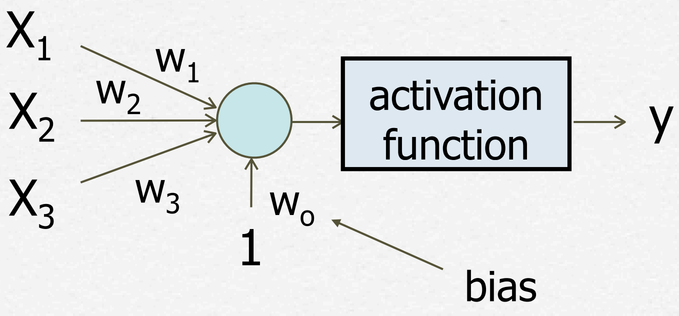
Credit: Andres Perez-Uribe

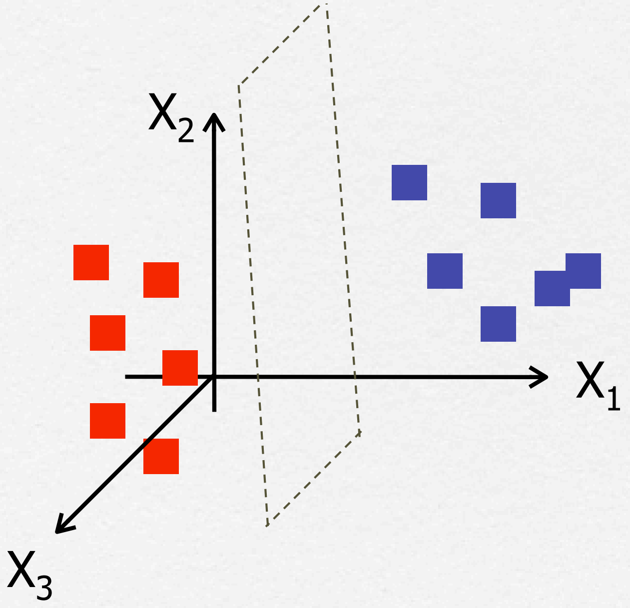
Objectives

- ☐ Understand how the basic model of an artificial neural network works
- ☐ Understand the capabilities of a Multi-layer Perceptron
- ☐ Understand how to train a Multi-layer Perceptron
- ☐ Understand the working of the Backpropagation algorithm

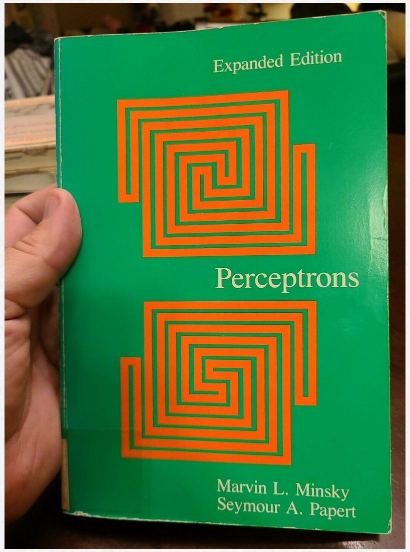
The Perceptron

A Perceptron implements a linear separation in the input space

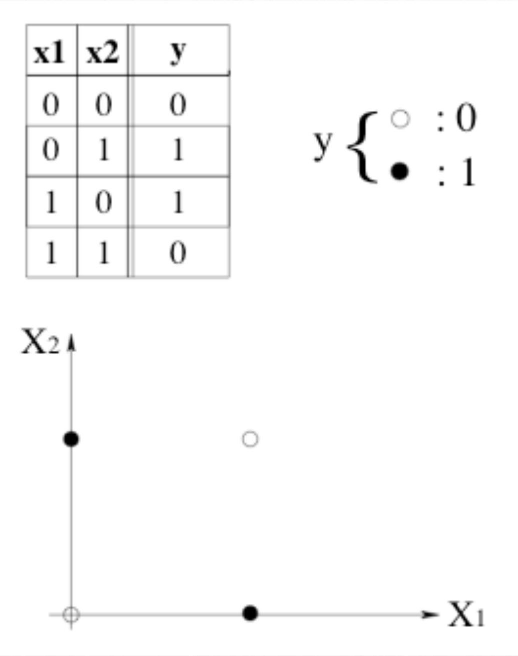




The XOR problem (1)



“Perceptrons” by
Minsky & Pappert, 1969



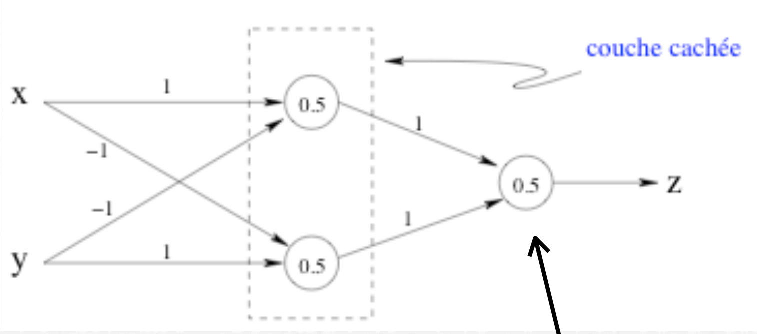
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Can we draw a line that
separates the two classes ?

The XOR problem (2)

The XOR problem is a nonlinearly separable: a single line cannot separate white and black dots

However, two lines can do the trick!

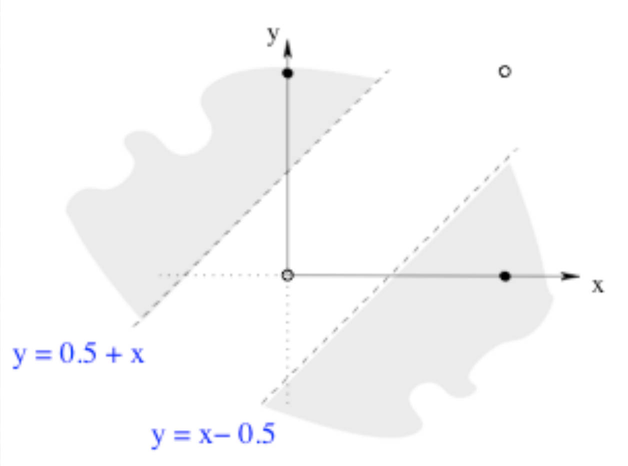


inputs

output

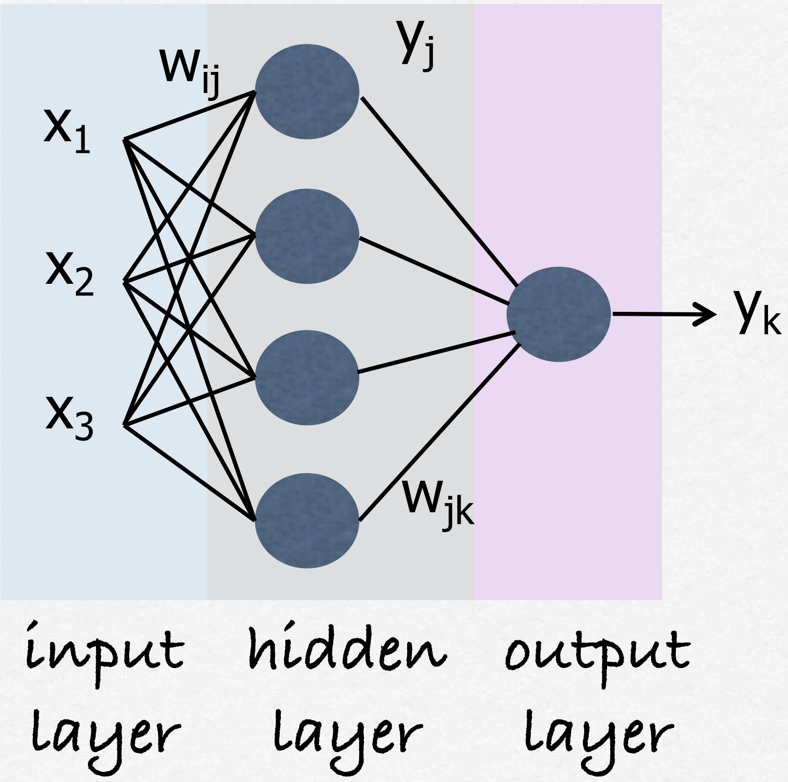
hidden layer

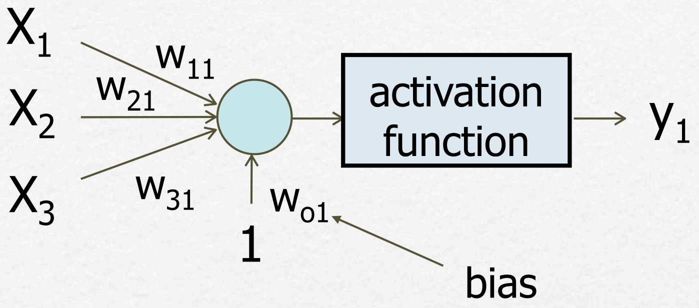
Perceptron



IF (x - y ≥ 0.5) OR (y - x ≥ 0.5)
THEN z = 1, ELSE z = 0

Multi-Layer Perceptron (MLP) [1]





$y_1 = f(\Sigma w_{i1}x_i) = f(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + w_{01}),$

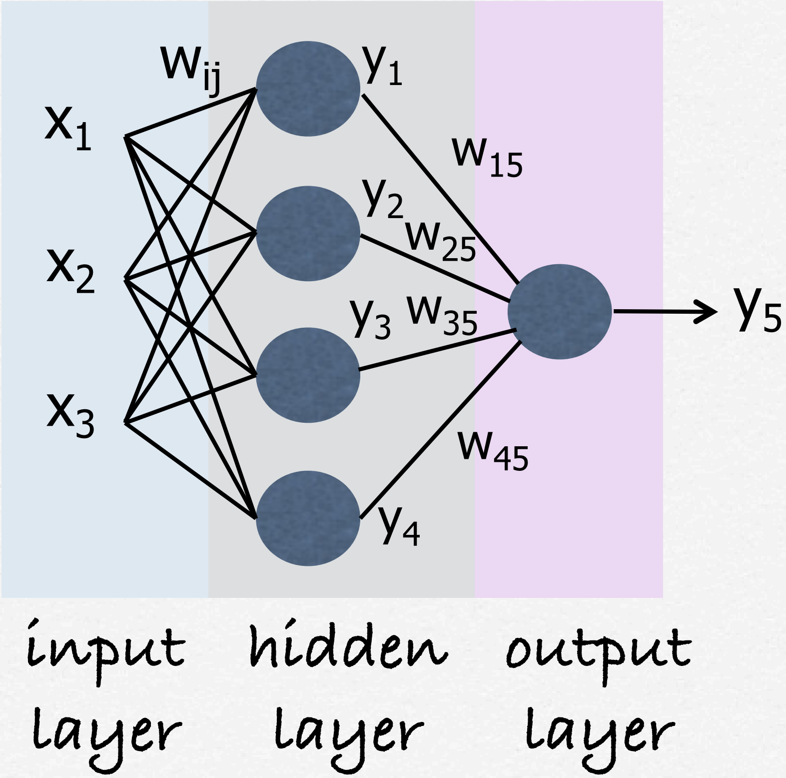
$f(x)$ = sigmoid(x) /* if we use a sigmoid */

$y_j = f(\Sigma w_{ij}x_i) = f(w_{1j}x_1 + w_{2j}x_2 + w_{3j}x_3 + w_{0j}),$

$j = 1 .. 4$

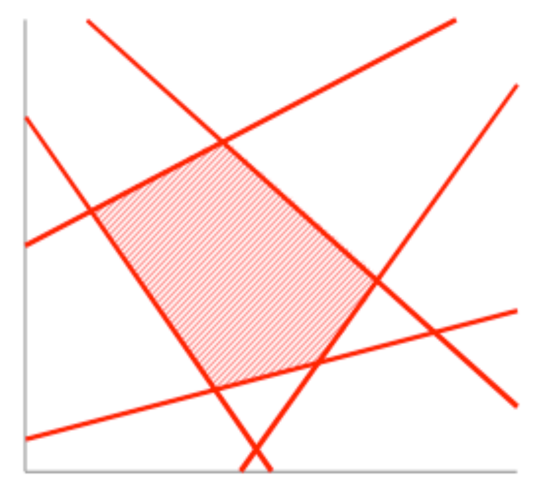
Artificial Neural Network (vanilla) == feed-forward neural net == fully connected layers

Multi-Layer Perceptron (MLP) [2]



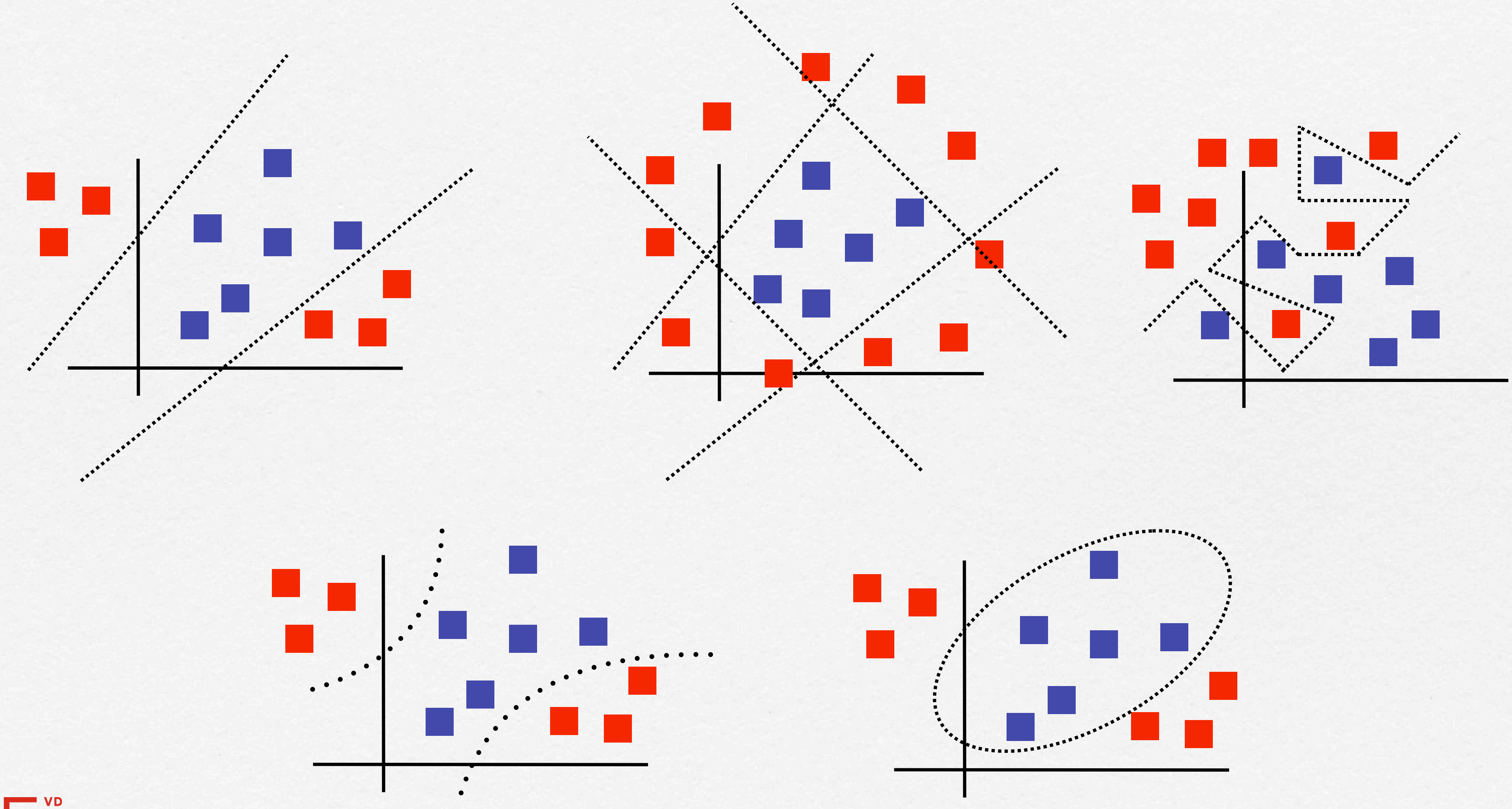
$$y_k = f(\Sigma w_{jk} y_j)$$

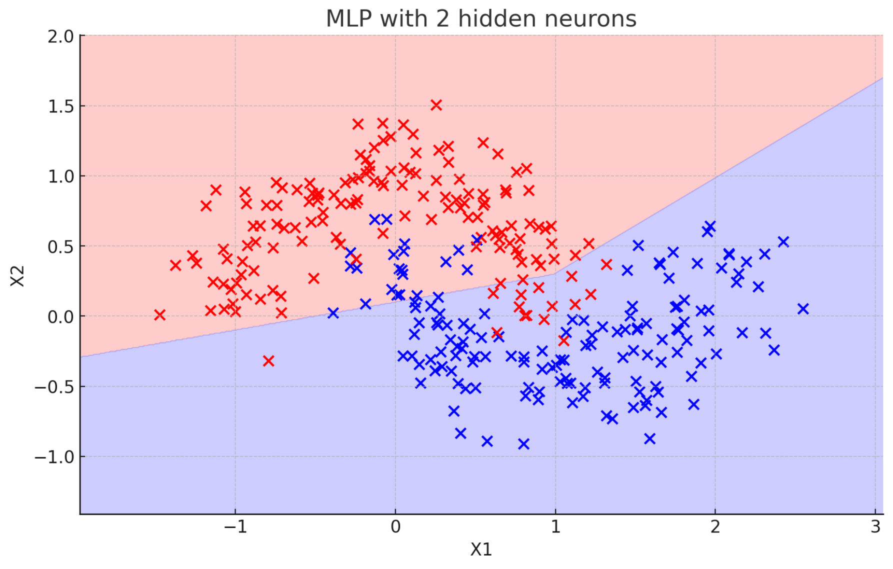
$$y_5 = f(w_{15}y_1 + w_{25}y_2 + w_{35}y_3 + w_{45}y_4 + w_{05}),$$
where $f(x) = \text{sigm}(x)$

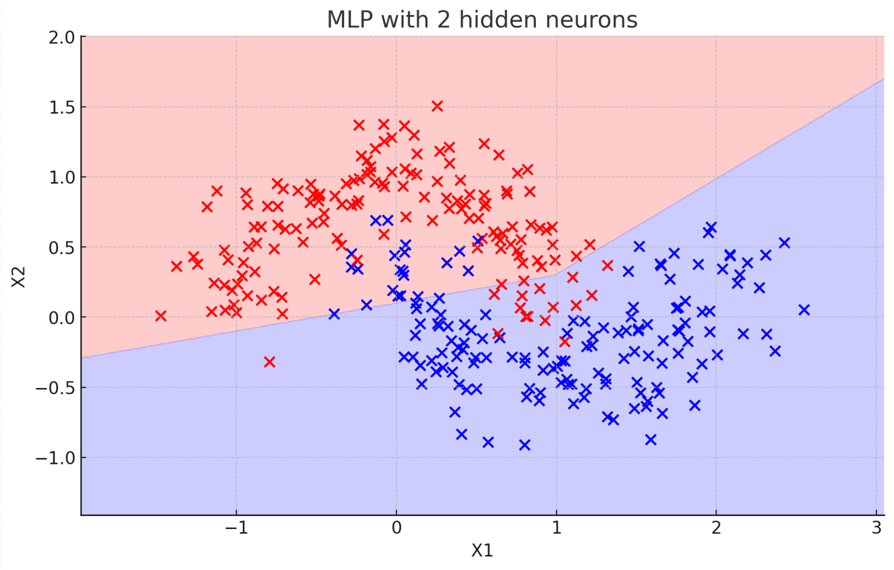


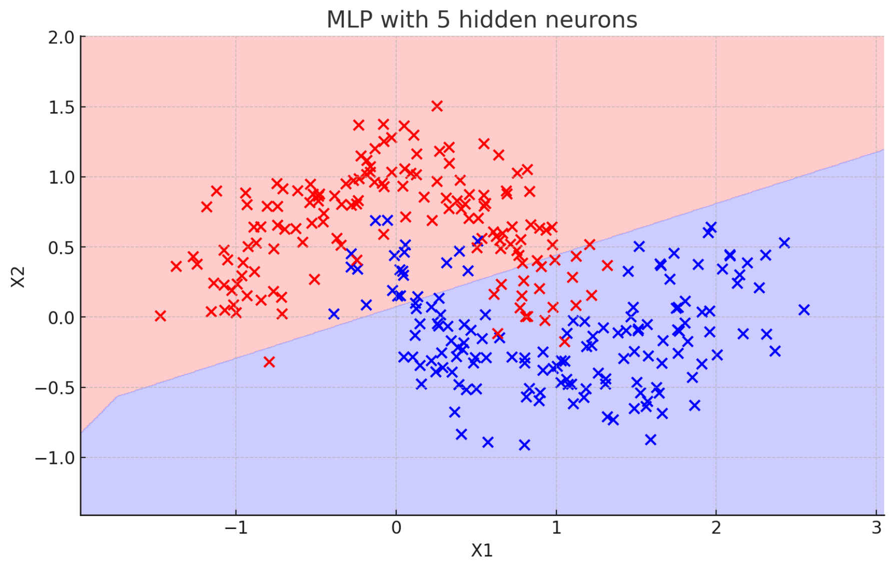
convex polygon region

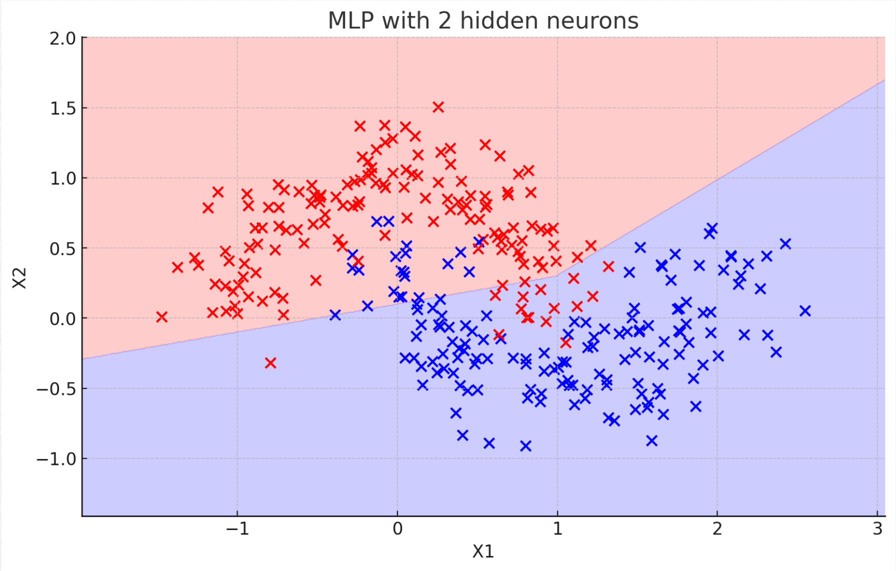
MLP nonlinear separation

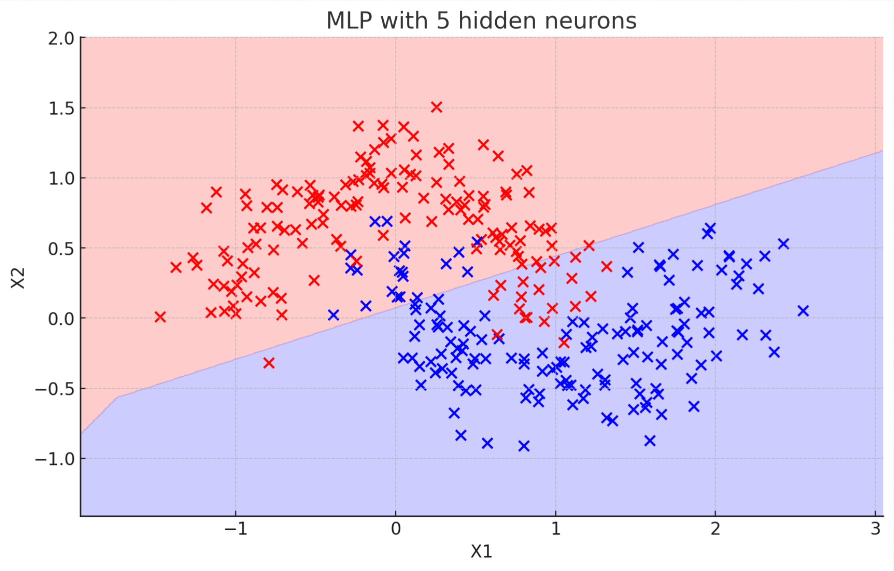


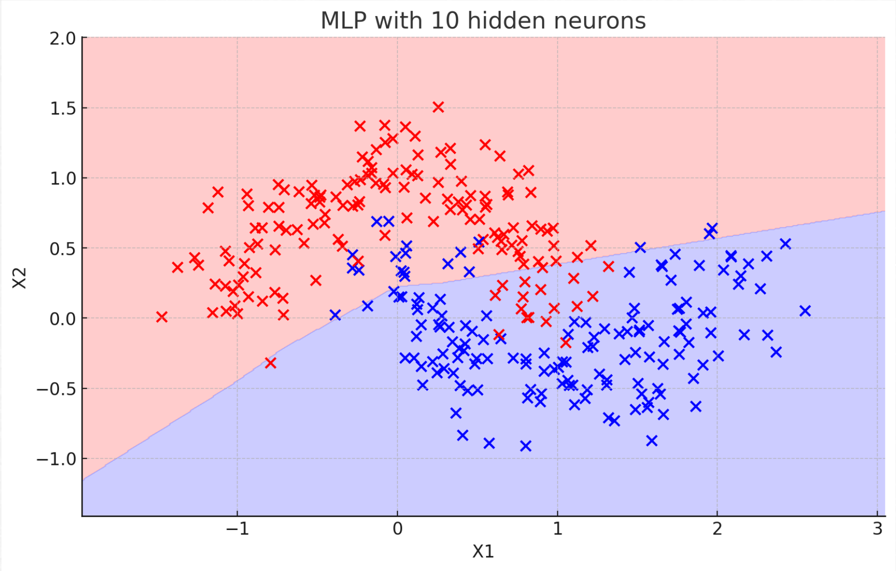

MLP with 2 hidden neurons
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
X2
−1
0
1
2
3
X1

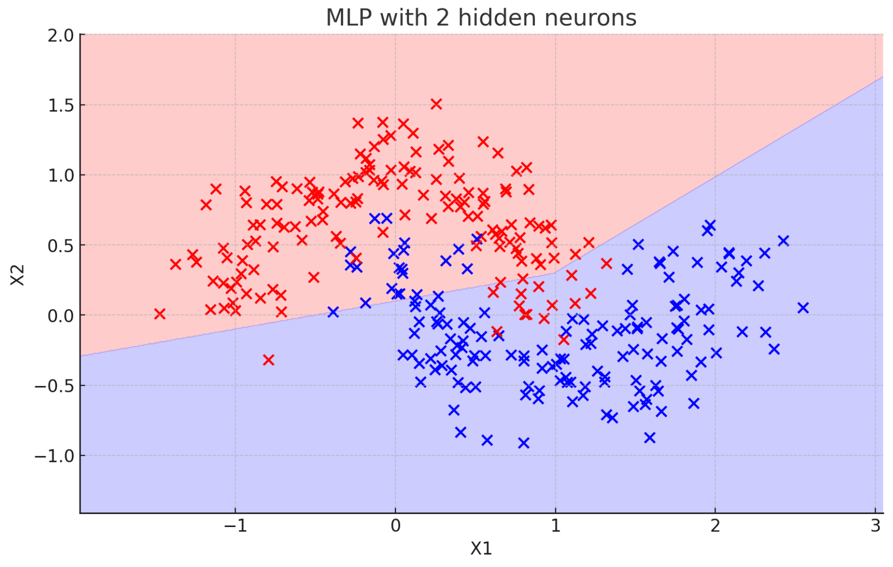

MLP with 2 hidden neurons
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
X2
−1
0
1
2
3
X1

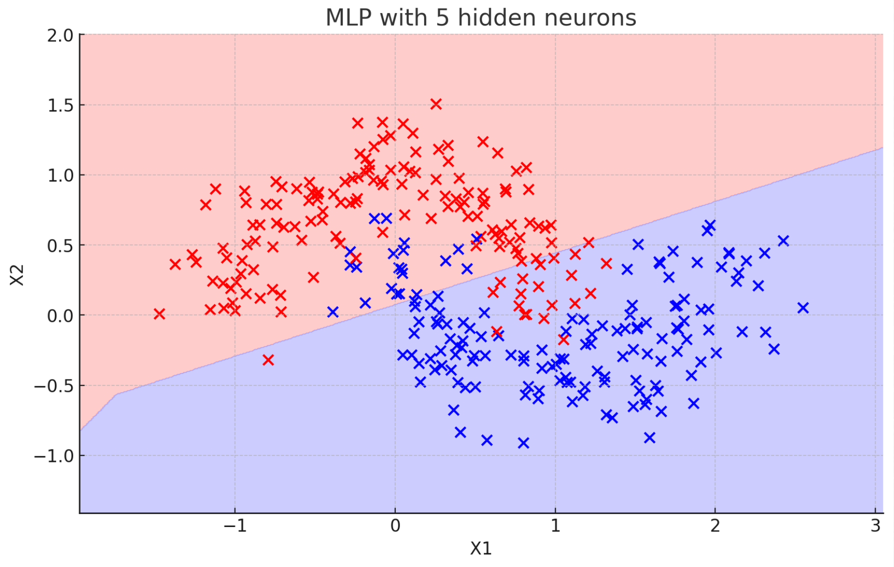

MLP with 5 hidden neurons
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
X2
−1
0
1
2
3
X1

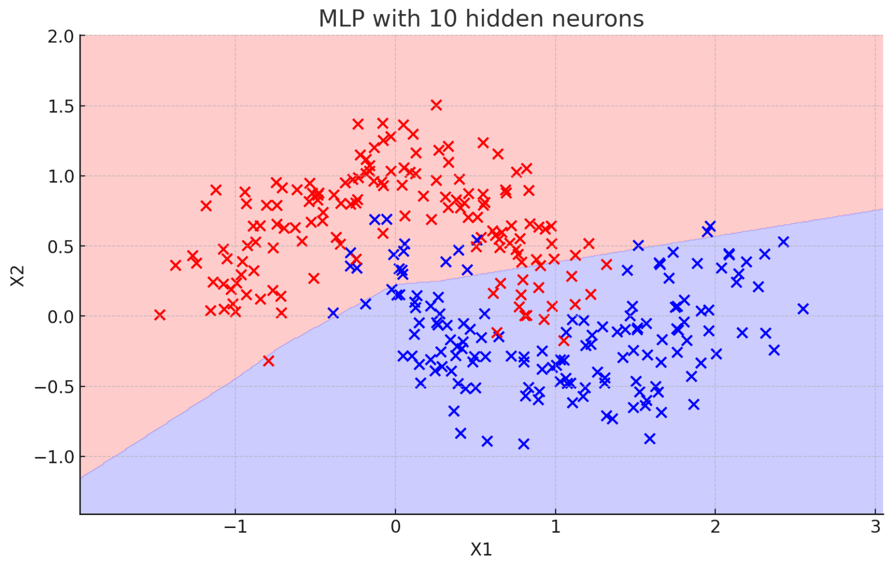

MLP with 2 hidden neurons

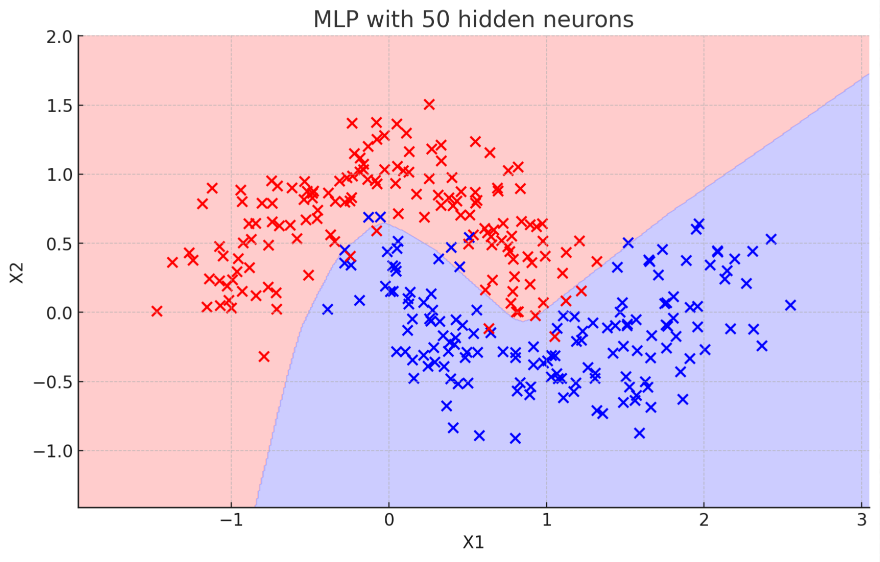

MLP with 5 hidden neurons


MLP with 10 hidden neurons

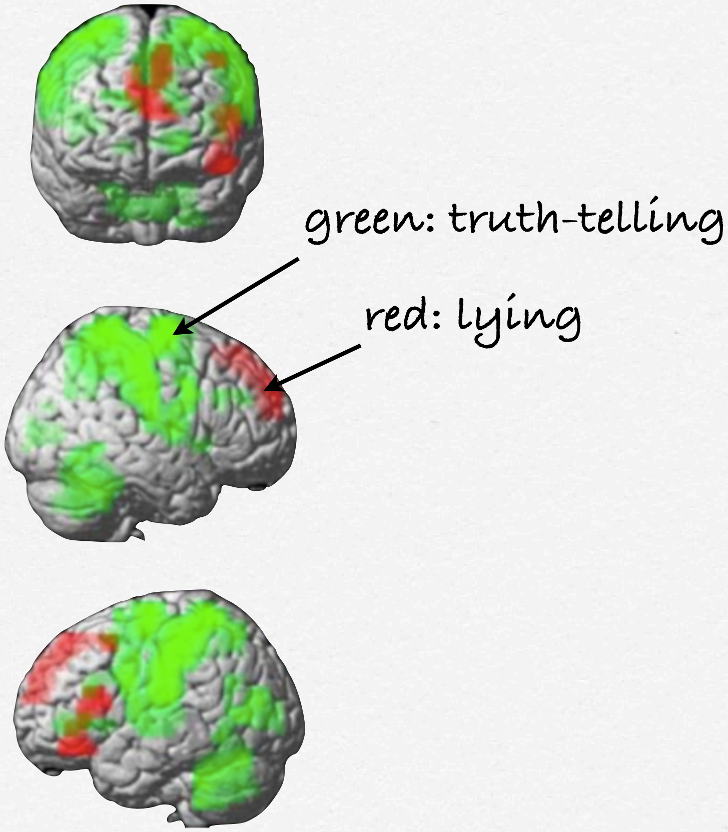

MLP with 2 hidden neurons


MLP with 5 hidden neurons


MLP with 10 hidden neurons


MLP with 50 hidden neurons

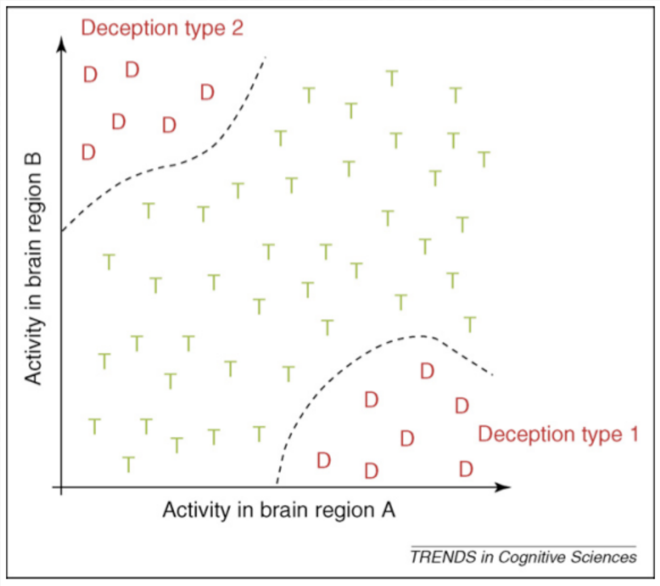
Truth-telling vs lying



green: truth-telling

red: lying

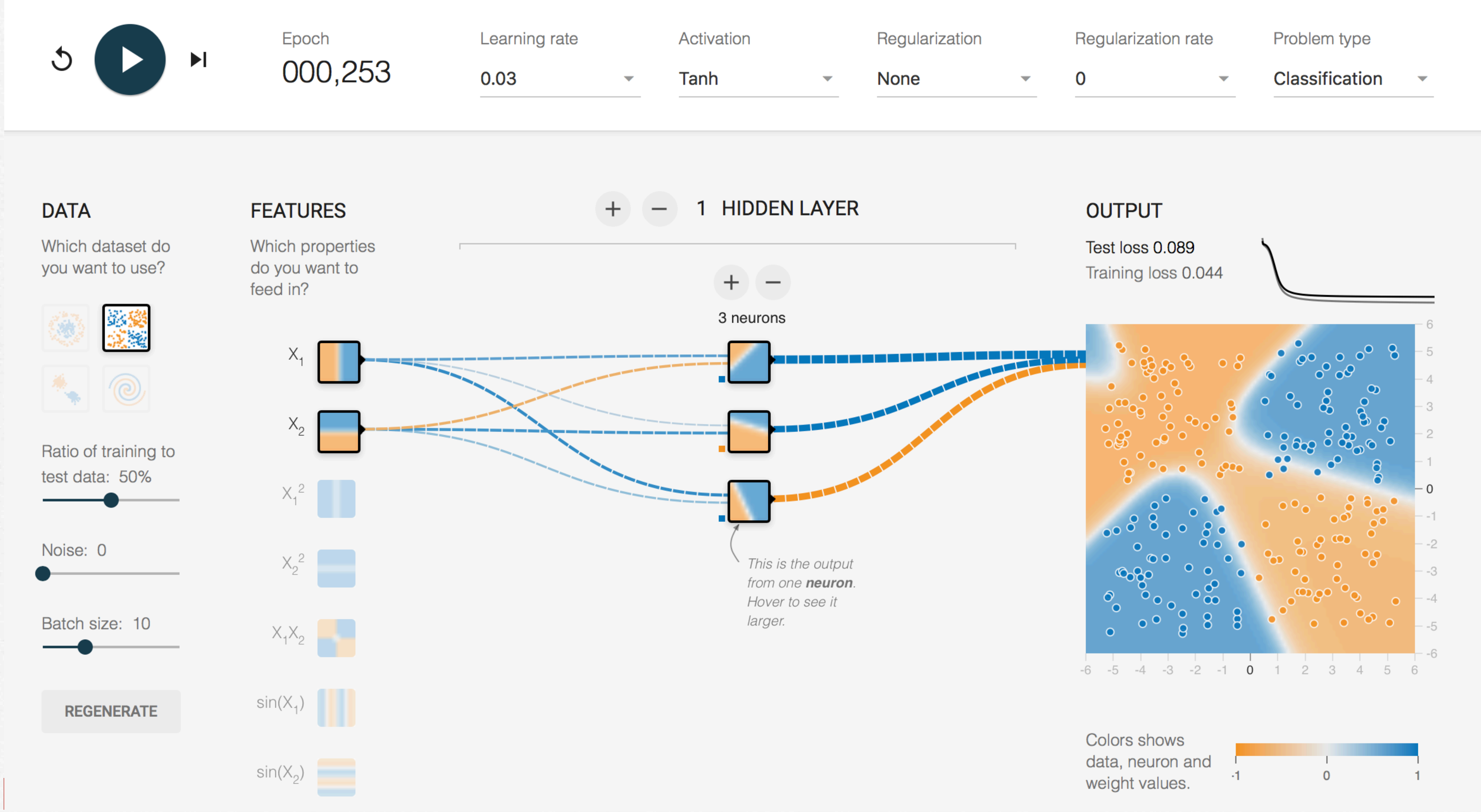
NeuroImage 28 (2005) 663 – 668



TRENDS in Cognitive Sciences

Trends in Cog. Sci, vol 12, No 4, March 2008

http://playground.tensorflow.org



Epoch 000,253

Learning rate 0.03

Activation Tanh

Regularization None

Regularization rate 0

Problem type Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

FEATURES

Which properties do you want to feed in?

1 HIDDEN LAYER

3 neurons

*This is the output from one **neuron**. Hover to see it larger.*

OUTPUT

Test loss 0.089

Training loss 0.044

Colors shows data, neuron and weight values.

Backpropagation algorithm

(Paul Werbos, 1974; Rumelhart & McClelland, 1986)

Error function to be minimized:

$$E(w^{(1)},w^{(2)}) = 1/2\sum_{\mu}\sum_{i}\left\| t_i^{out}(\mu) - x_i^{out}(\mu)\right\|^2$$

training pattern

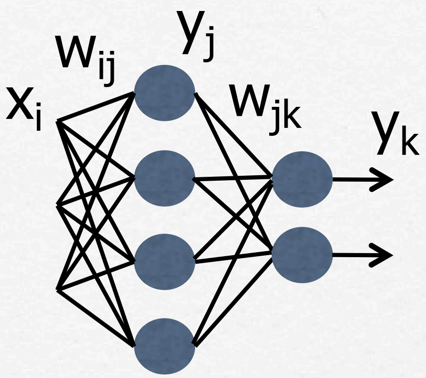
output neuron

Stochastic gradient descent:

(k) indicates the layer

$$\Delta w_{ij}^{(k)} = -\eta \partial E / \partial w_{ij}^{(k)}$$

Backpropagation



1. Randomly initialize weights
2. Compute the outputs y_k for a given input vector X:

   $y_k=f(\Sigma_j w_{jk} y_j)$, where $y_j=f(\Sigma_i w_{ij} x_i)$ and $f(s)=1/(1+e^{-s})$
3. For each output neuron, compute:

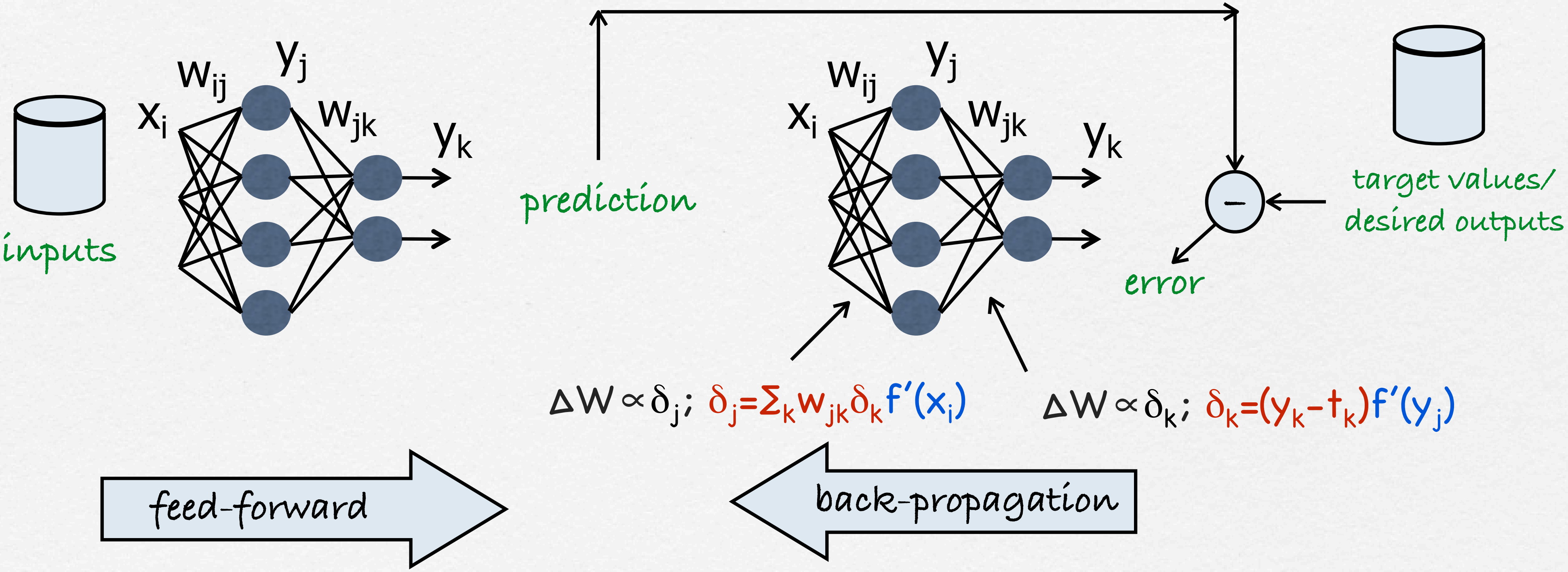
   $\delta_k=(y_k-t_k)f'(y_j)$, where $f'(y_j)=y_k(1-y_k)$ is the derivative of the sigmoid function, and t_k is the desired output of output neuron k
4. For each neuron of the hidden layer, compute:

   $\delta_j=\Sigma_k w_{jk} f'(x_i)\delta_k$, where $f'(x_i) = y_j(1-y_j)$
5. Update the networks' weights as follows:

   $w_{jk}(t+1) = w_{jk}(t) - \eta\delta_k y_j$; $w_{ij}(t+1) = w_{ij}(t) - \eta\delta_j x_i$,

   where η is the learning rate, $0<\eta<1$
6. Repeat 2 to 5 for a given number of steps or until the error is smaller than a given threshold

Backpropagation of the error

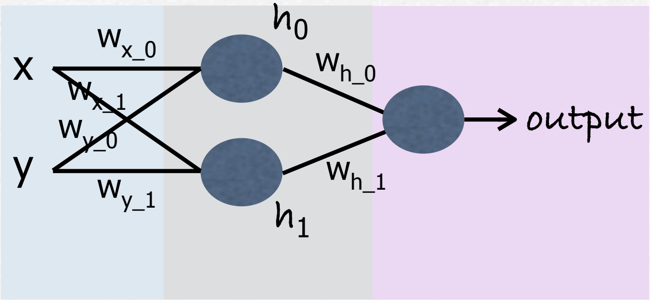


$\Delta W \propto \delta_j$; $\delta_j = \Sigma_k w_{jk} \delta_k f'(x_i)$

$\Delta W \propto \delta_k$; $\delta_k = (y_k - t_k) f'(y_j)$

δ_j is estimated from δ_k

Feed-forward computation



```
def sigmoid(neta):
    output = 1 / (1 + np.exp(-neta))
    d_output = output * (1 - output)
    return (output, d_output)

def perceptron(input_values, weights, bias, activation_function):
    neta = np.dot(input_values, weights) + bias
    return activation_function(neta)

h_0, h_0_d = perceptron(input_values, [w_x_0, w_y_0], b_0, activation_function)
h_1, h_1_d = perceptron(input_values, [w_x_1, w_y_1], b_1, activation_function)
h = np.array([h_0, h_1]).T
output, output_d = perceptron(h, [w_h_0, w_h_1], b_h, activation_function)
```

Weight adaptation by Backpropagation

learning rate

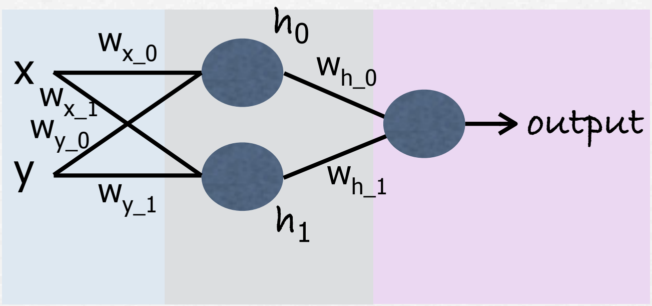
```
def compute_delta_w(input_values, targets, alpha, activation_function, weights, bias):
  …
  #output layer
  error = output - targets
  d_w_h_0 = -alpha * (error * output_d) * h_0
  d_w_h_1 = -alpha * (error * output_d) * h_1
  d_b_h   = -alpha * (error * output_d)
```

δ_k

δ_j

δ_k

…



$\delta_k = (\text{output-target}) f'(h_j)$

$\Delta W_h = -\alpha \delta_k h_0$

$\delta_j = \Sigma_k w_{jk} f'(x_i) \delta_k$

$\Delta W_{in} = -\eta \delta_j x_i$

Backpropagation using Machine learning libraries

```
#Dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense

# Neural network definition
model = Sequential()
model.add(Dense(5, input_dim=13, activation='sigmoid'))
model.add(Dense(3, activation='sigmoid'))

# Model compilation
model.compile(loss='mse', optimizer='sgd', metrics=['accuracy'])

# Model training
model.fit(X, y, epochs=100, batch_size=5)
```

hidden neurons

number of inputs

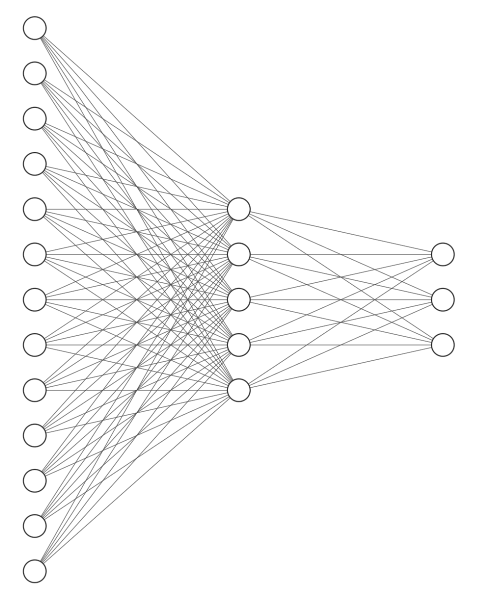
number of outputs

Stochastic Gradient Descent

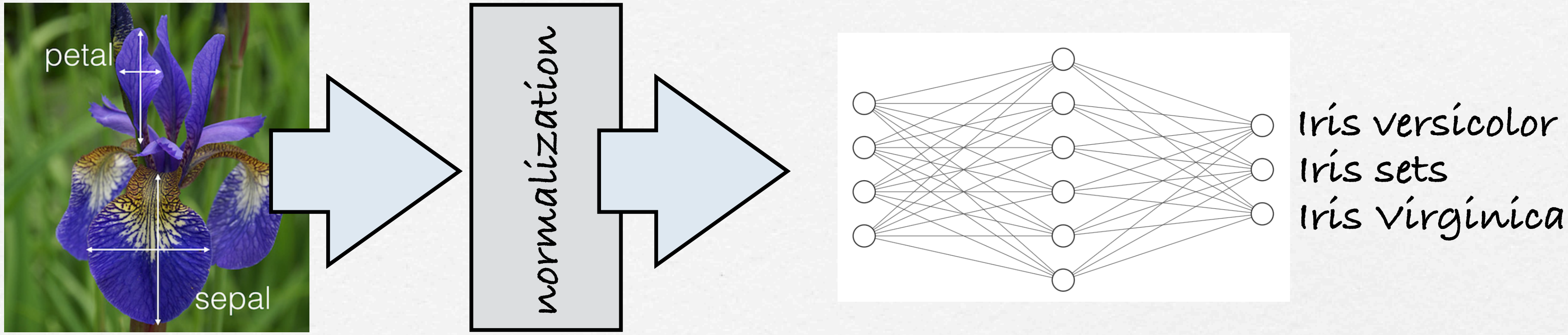
Mean Squared Error

{inputs; targets}

number of iterations

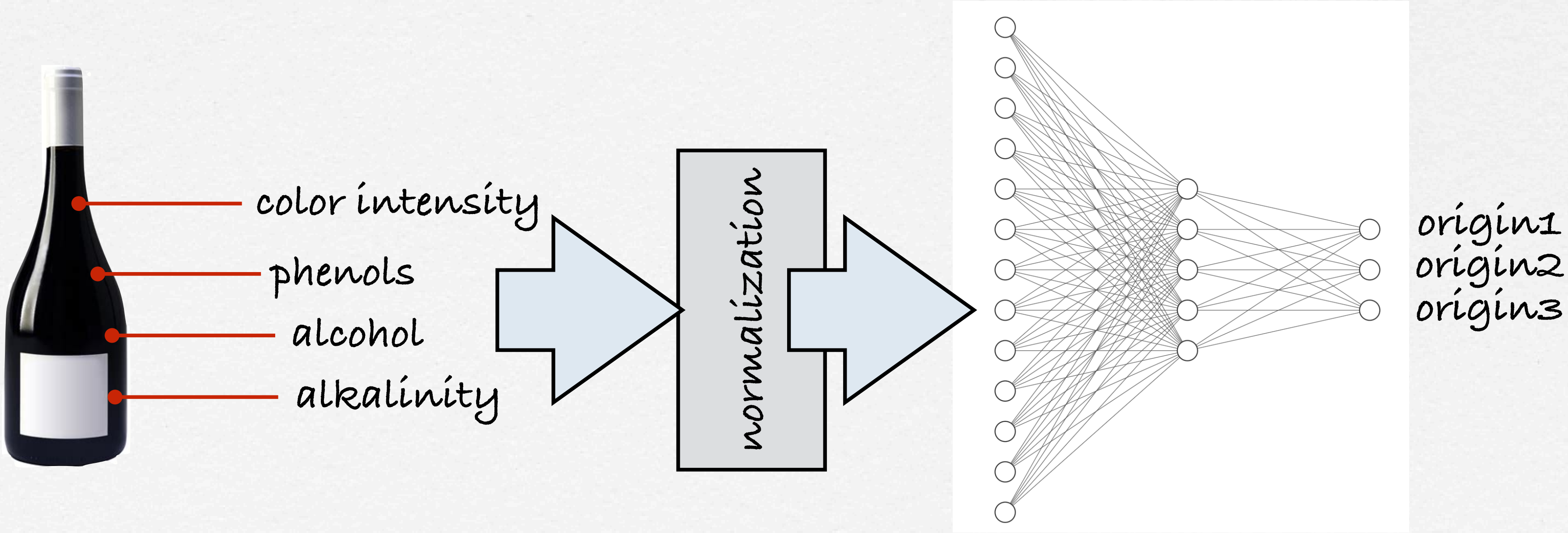


Example applications (1)



normalization

Iris versicolor
Iris sets
Iris Virginica

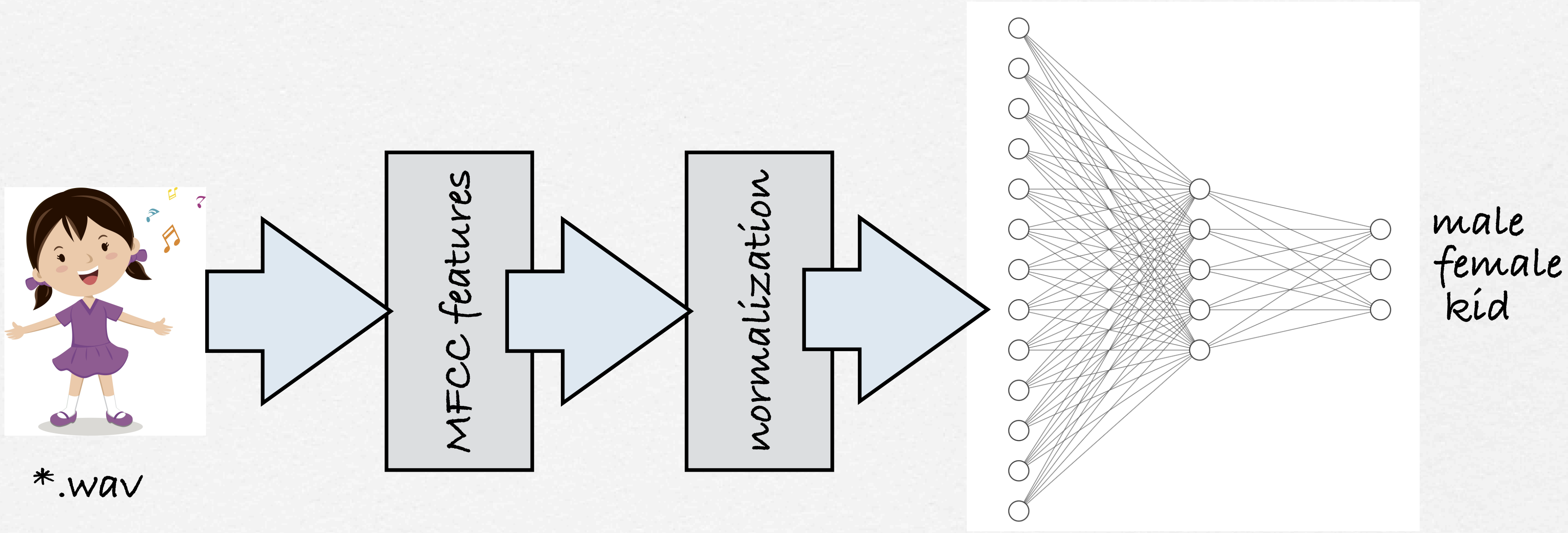


color intensity
phenols
alcohol
alkalinity

normalization

origin1
origin2
origin3

Example applications (2)



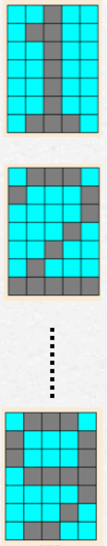
*.wav

MFCC features

normalization

male
female
kid

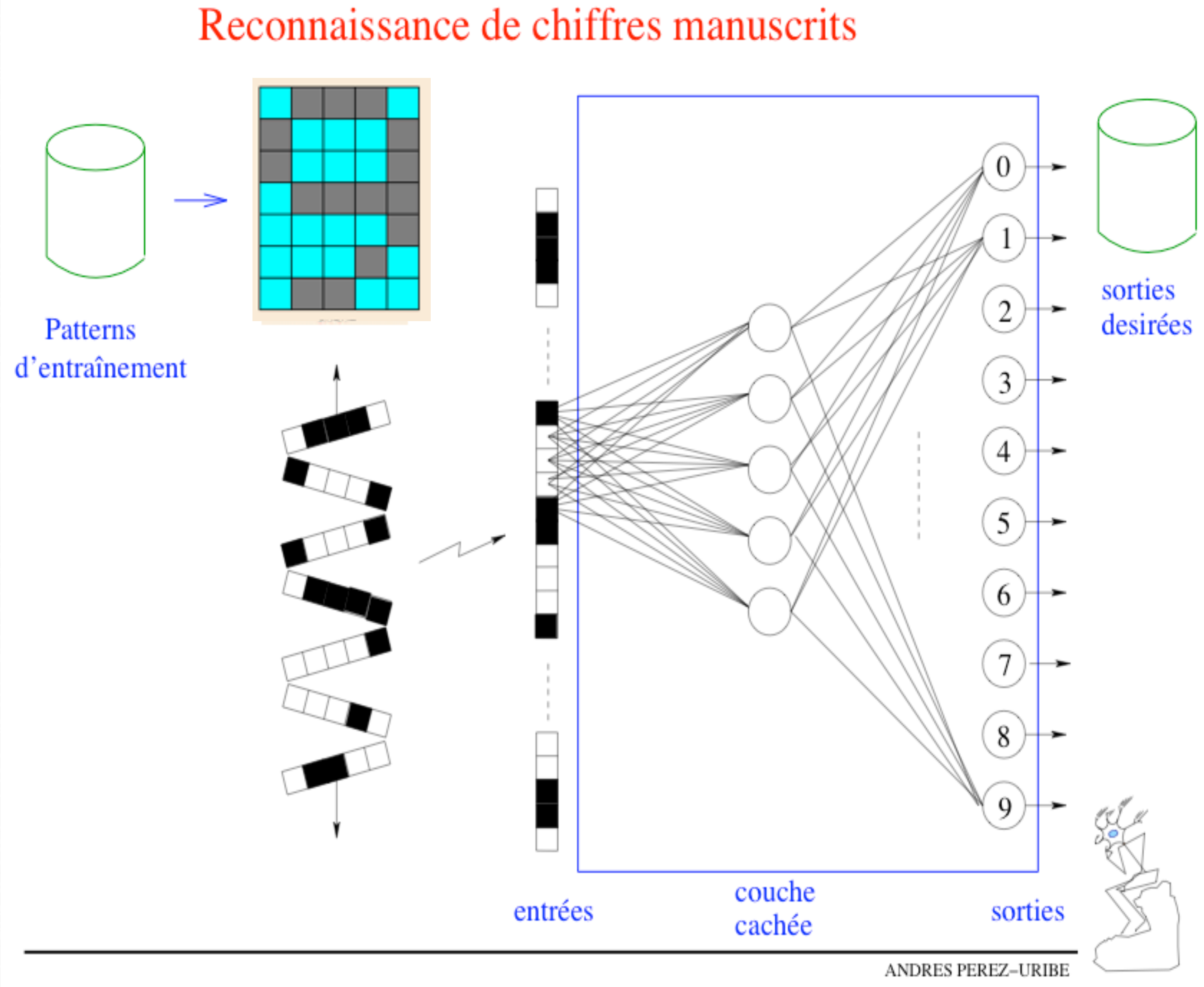
Digit recognition



= 1

= 2

= 9



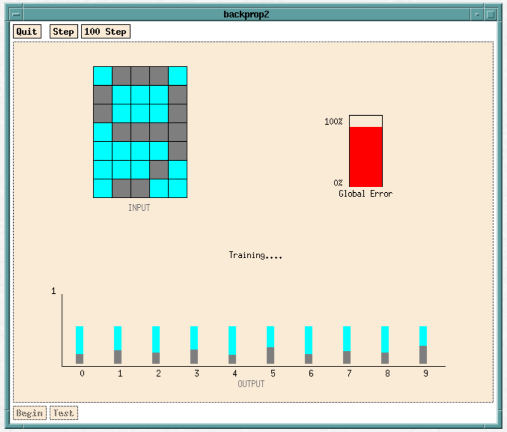
Reconnaissance de chiffres manuscrits

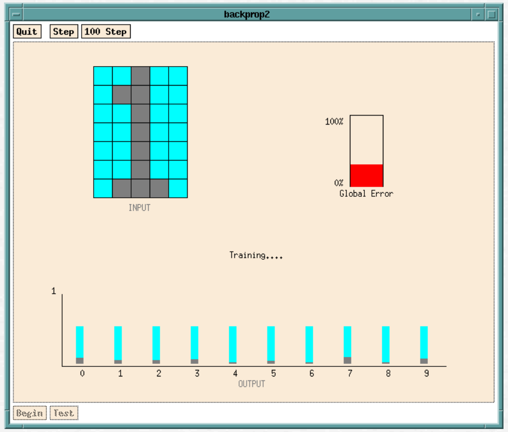
ANDRES PEREZ-URIBE

labels

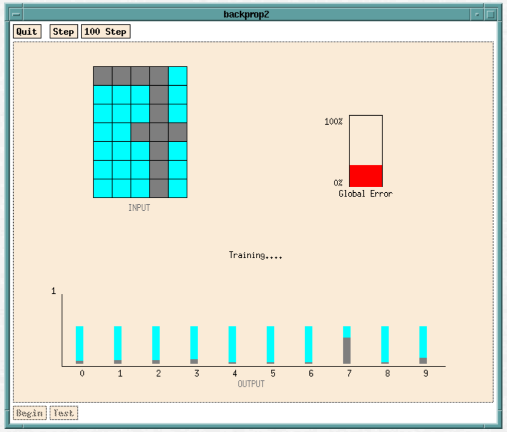
[0
0
0
0
0
0
0
0
0
1]

Learning process

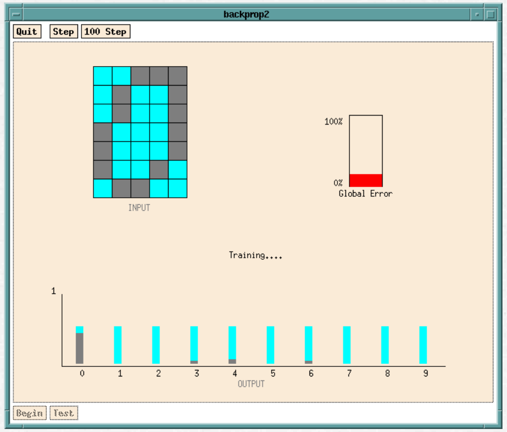




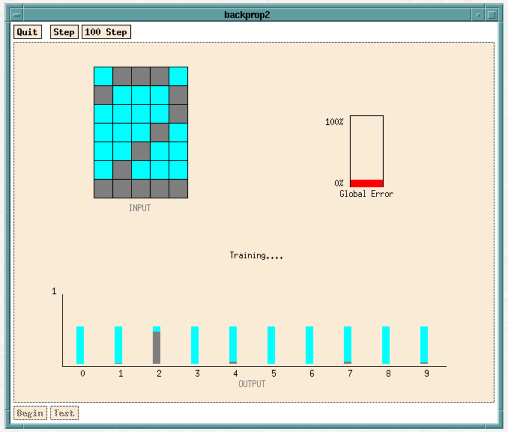
after 1000 steps



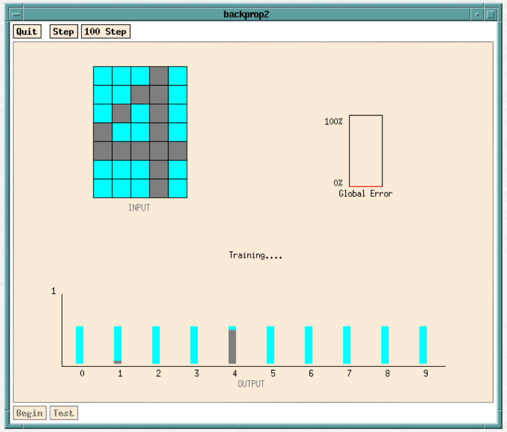
after 1021 steps



after 4000 steps

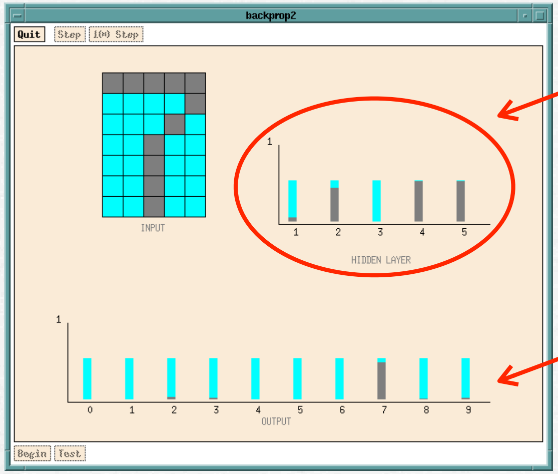


after 8000 steps



after 12000 steps

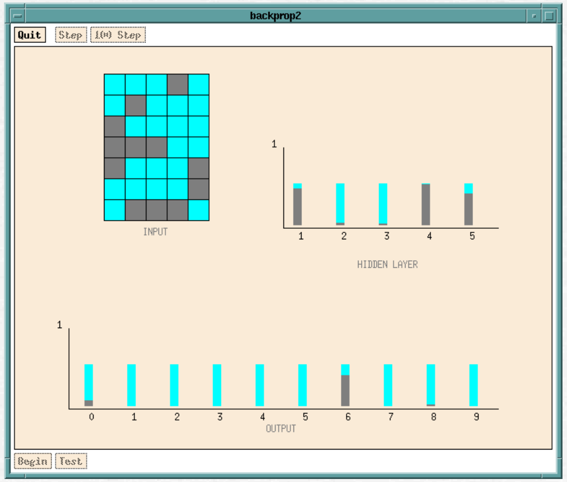
Generalization capability



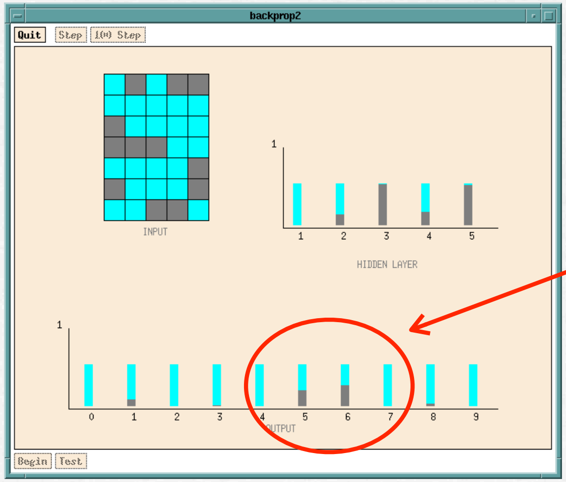
hidden unit activation ≈ feature detection

outputs' activation ≈ digit recognition

e.g., argmax(outputs)



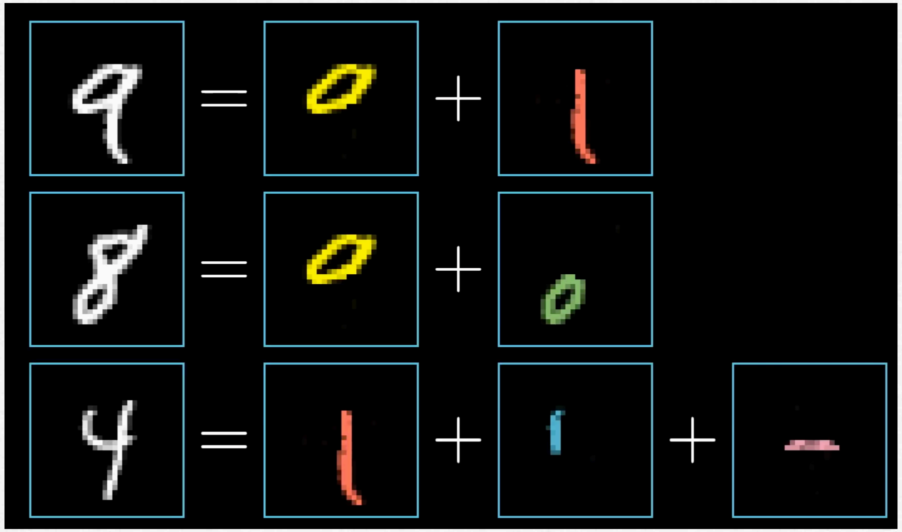
generalization == noise tolerance



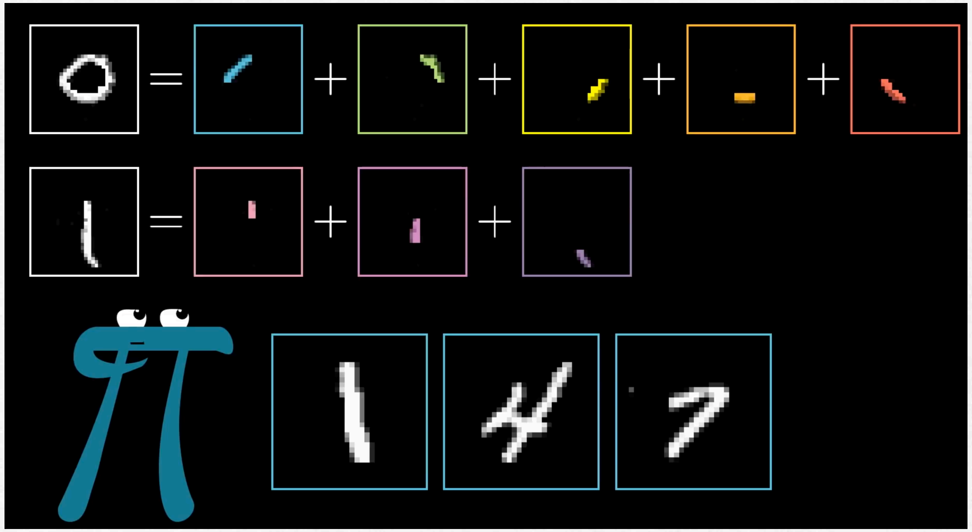
5 or 6 ?

there is no magic

Detected features in the hidden layers

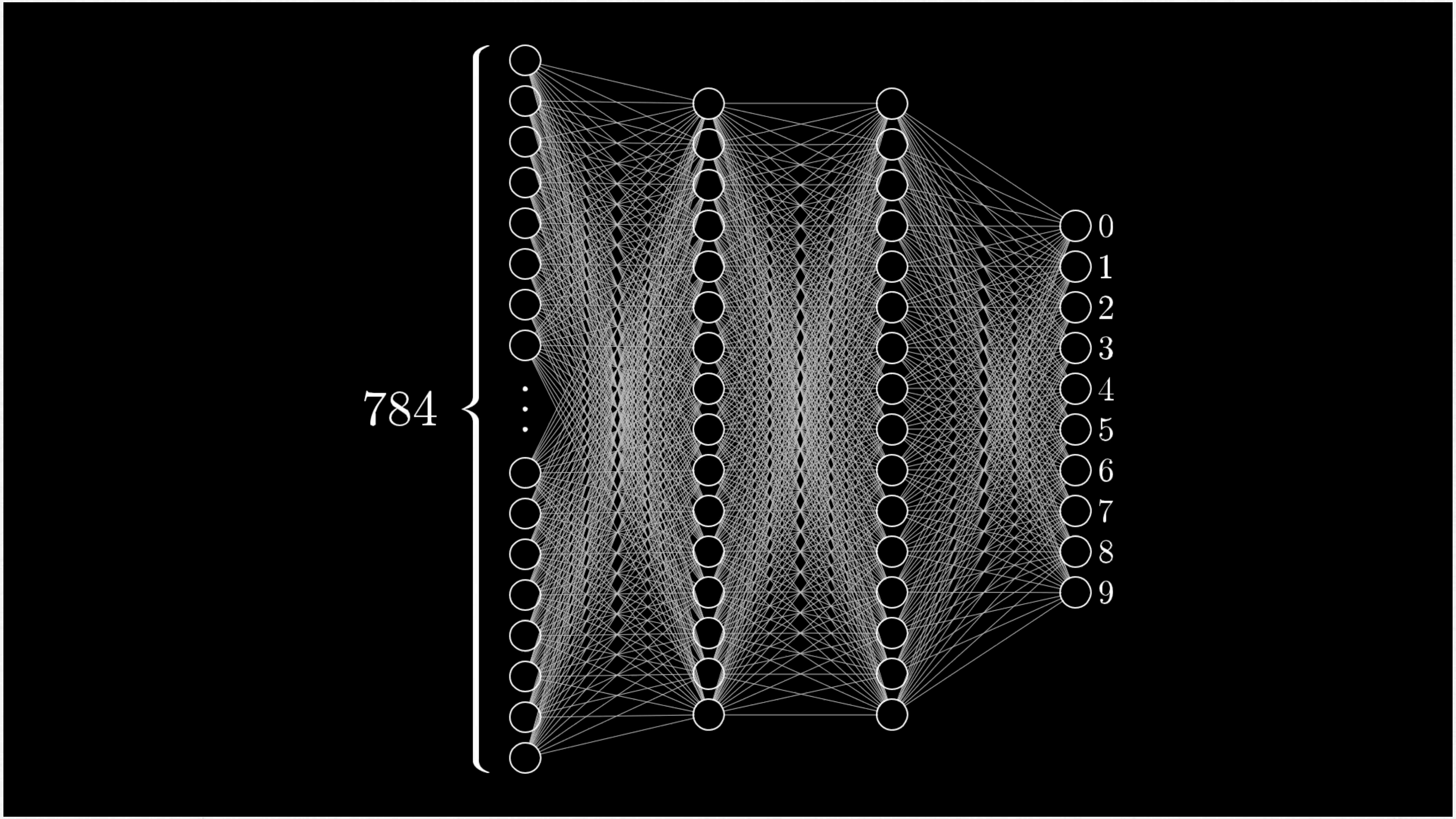


ideally, but current neural networks are not so smart



a loop might be the sum of several edges

from “What is a Neural Network”, video by 3blue1brown



https://www.youtube.com/results?search_query=3+brown+1+blue

Practical considerations

- How many neurons in the hidden layer ? or how many hidden layers ?
- What activation function shall we prefer ?
- How many learning iterations are needed ?
- What learning rate shall we use ?
- How to avoid overfitting ?

  - hyper-parameter tuning (model selection)
  - and more…