ML04 — Neural Networks

Romain Gautron

CIAT

March 5, 2019

1 The perceptron

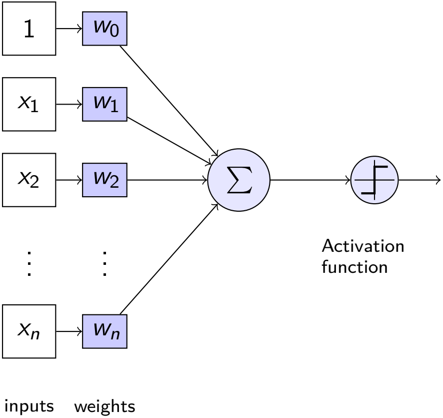
2 Multilayer perceptron

3 Deep Neural Networks

4 Transfer Learning

5 NN in practice

Rosenblatt perceptron

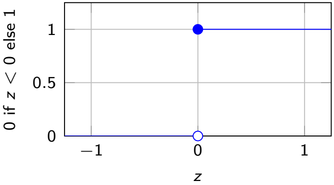


$$z = W^T X_i = \sum_{i=0}^{n} x_i w_i$$

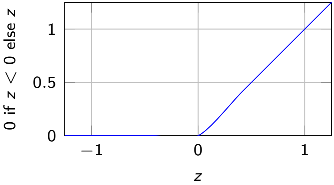
$$A(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ else} \end{cases}$$

$$Y = A(W^T X_i)$$

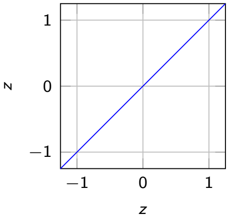
Some common activation functions



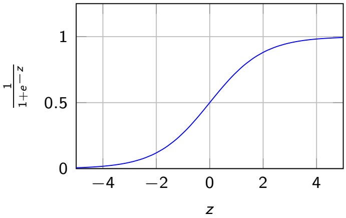
(a) Heaviside



(b) ReLU

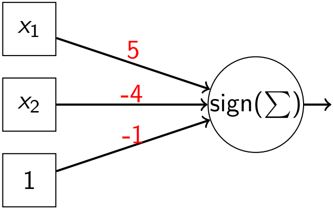


(c) Linear



(d) Logistic sigmoid

Perceptron example



x_1	x_2	$\hat{y}$
1	1	1
-1	1	0
0	0	0

$$z = W^T X_i = \sum_{i=0}^{n} x_i w_i$$

$$A(z) = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ else} \end{cases}$$

$$Y = A(W^T X_i)$$

1 The perceptron

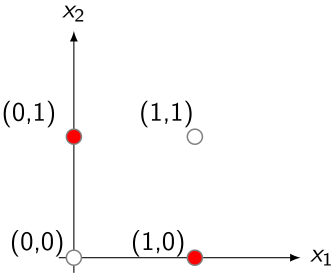
2 Multilayer perceptron

3 Deep Neural Networks

4 Transfer Learning

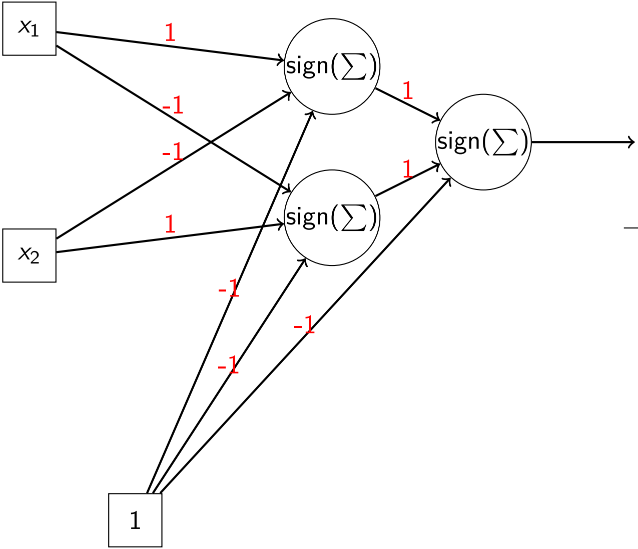
5 NN in practice

The XOR problem



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

One XOR solution (Heaviside activation)

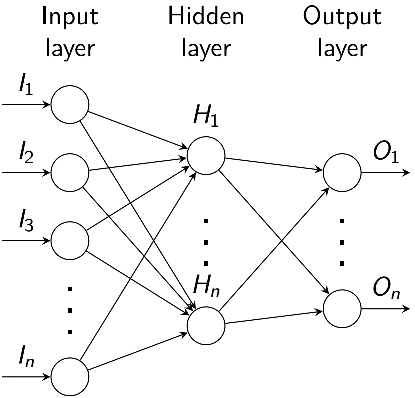


x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

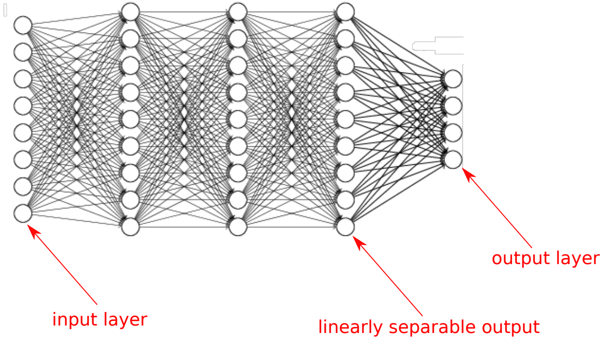
MLP

Universal approximation theorem (Haykin, 99)

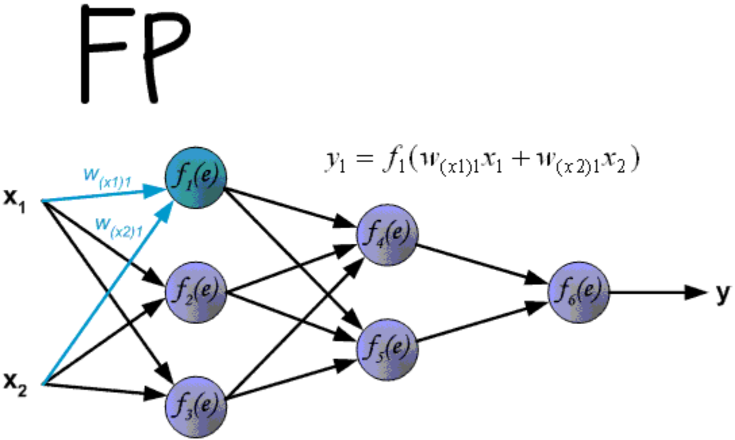
"A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"



Stacking



Back-propagation



Vocabulary

Batch
: Blocks of training set sliced in a given size (ex:64) feeding each FP + BP [parameter impacting gradient descent]

Epoch
: FP + BP of **all** training set (all batches)

Iteration
: Number of batches with FP + BP

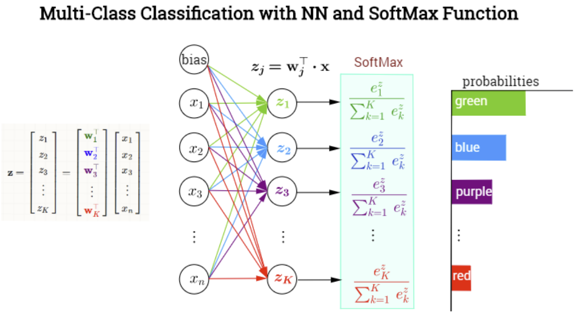
Classification VS Regression Output Layer

Classification (m classes)	**Regression**
$m == 2$: **1 neuron** with **sigmoid** activation $m > 2$: $\boldsymbol{m}$ **neurons** with **softmax** activation	**1 neuron** with **linear** activation

Softmax activation

Softmax

$$softmax(z_j) = \frac{\exp z_j}{\sum_{i=1}^{m} \exp z_i}, \mathbb{R} \rightarrow]0, 1]$$



→ Extension of the logistic regression to a multiclass (>2) problem

- ↪ Outputs interpreted as the probability of each class
- ↪ $\hat{y}$ is the maximum probability

Neural Network regularization (1)

A MLP has a very high complexity: prone to overfitting

$\hookrightarrow$ Avoiding co-adaptation in MLP hidden layers $\Rightarrow$ leds to overfitting

Dropout

At each step of the training, we randomly set the output of hidden layer neurons to 0 with a probability p

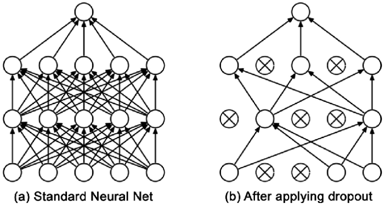


Figure: Dropout illustration, courtesy Wenhao Zhang

In general, improves well generalization performances

Neural Network regularization (2)

Same idea than with linear regression

$\ell 1$ and $\ell 2$ norms

$$\mathbf{L}_{\ell 2} = \mathbf{L} + \frac{\lambda}{2}||\mathbf{W}||_2^2$$

$$\mathbf{L}_{\ell 1} = \mathbf{L} + \frac{\lambda}{2}||\mathbf{W}||_1$$

Batch Normalization

- $\rightarrow$ whereas they can deal with, NN prefer scaled inputs
- $\rightarrow$ batch normalization: exponential moving average scaling the output of layer feeding another one
- $\rightarrow$ helps to converge
- $\rightarrow$ typically used after dense or convolution layers

How do I choose the architecture?

Choices to do

- $\rightarrow$ activation functions
- $\rightarrow$ weight initialization
- $\rightarrow$ number of hidden layers
- $\rightarrow$ number of neurons in hidden layers
- $\rightarrow$ regularization
- $\rightarrow$ ...

No known heuristics to help, rules of thumb:

- $\rightarrow$ as much input neurons as features in $\mathbf{X}$
- $\rightarrow$ start with one hidden layer of the mean between the number of input and output neurons
- $\rightarrow$ choose the output layer according to your problem

1 The perceptron

2 Multilayer perceptron

3 Deep Neural Networks

4 Transfer Learning

5 NN in practice

Convolutional NN: Translation invariance

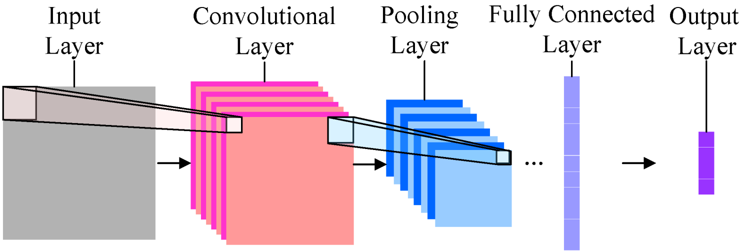


Figure: CNN architecture — from *NIRFaceNet: A Convolutional Neural Network for Near-Infrared Face Identification*

Convolution

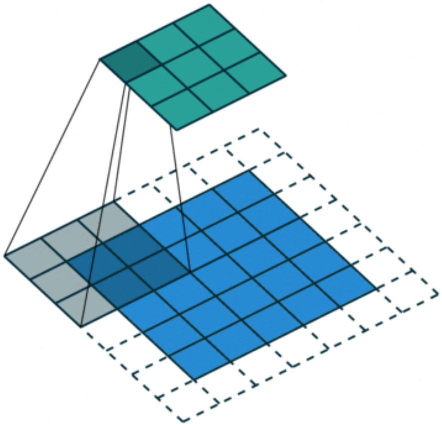


Figure: Convolution illustration — courtesy Vincent Dumoulin, Francesco Visin

Pooling

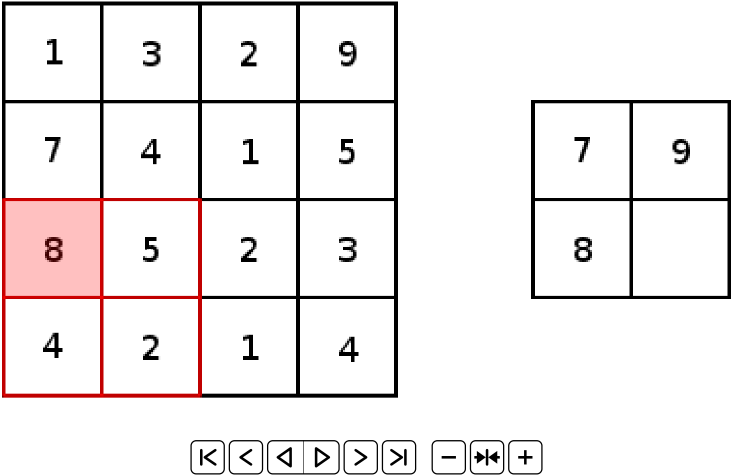


Figure: Pooling illustration — courtesy Justin Francis

Recurrent NN: Time invariance (1)

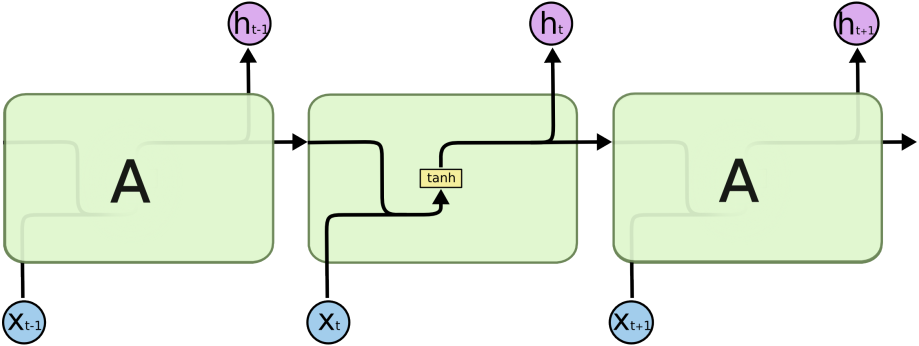


Figure: RNN — courtesy Christopher Olah

Recurrent NN: Time invariance (2)

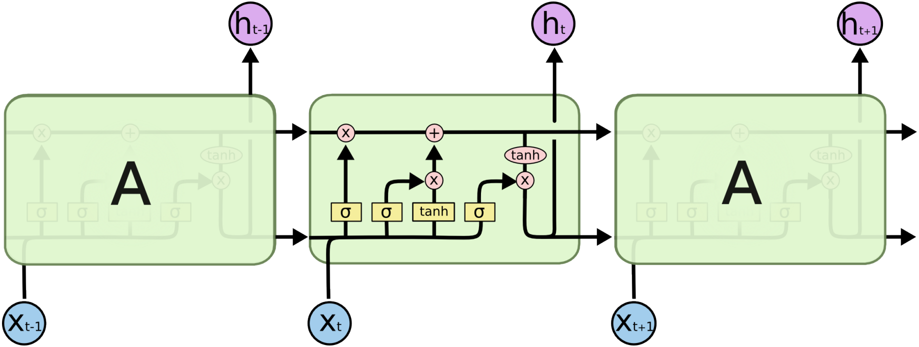


Figure: LSTM — courtesy Christopher Olah

1. The perceptron
2. Multilayer perceptron
3. Deep Neural Networks
4. **Transfer Learning**
5. NN in practice

Hierarchy in CNN

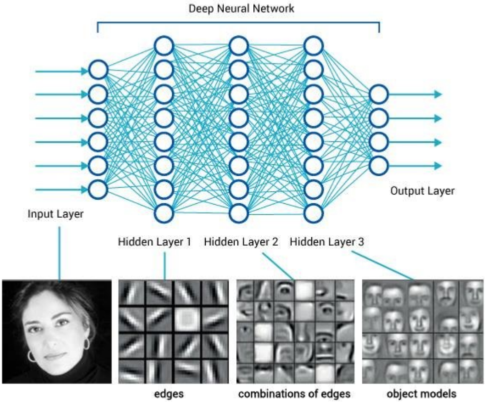


Figure: Hiearchy in DL — courtesy Yalie Nie

Principle

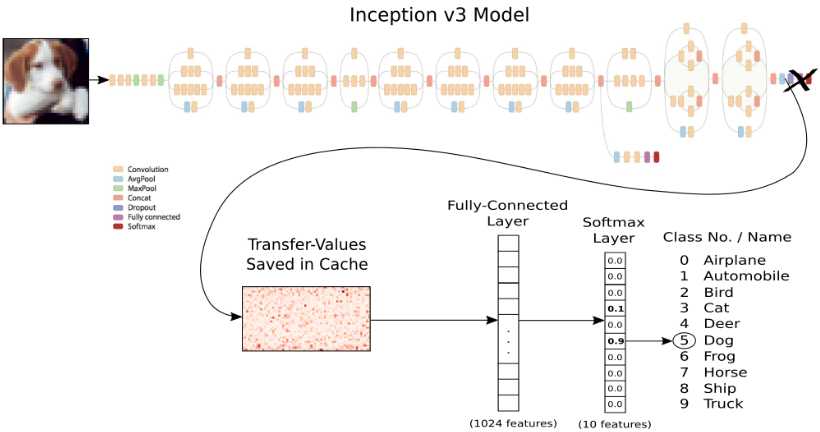


Figure: Transfer Learning principle — courtesy Hvass-Labs

Advantages of TL

→ low cost to train (possible on a laptop)

→ works with a few thousands training images

1 The perceptron

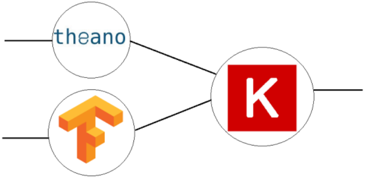
2 Multilayer perceptron

3 Deep Neural Networks

4 Transfer Learning

5 NN in practice

Keras



Use of GPU

What did we learn?

- $\rightarrow$ NN unit is based on Rosenblatt perceptron
- $\rightarrow$ the MLP can approximate any function
- $\rightarrow$ NN training is are iterative suit of forward pass and backpropagation
- $\rightarrow$ NN can perfom either classification or regression
- $\rightarrow$ Widely used DNN architectures are convolutional (space invariance) and recurrent NN (time invariance)
- $\rightarrow$ transfer learning is an handy way to use DNN with minimal efforts