Neural Networks:

An Introduction

Reinier Maat

Outline

- Introduction
  - Relevance, Motivation and Concept
- How to train Neural Nets
  - Networks, Loss functions, Gradient Descent
- Visual Demo
- Packages
- Digit Recognition Example in TensorFlow

Credits

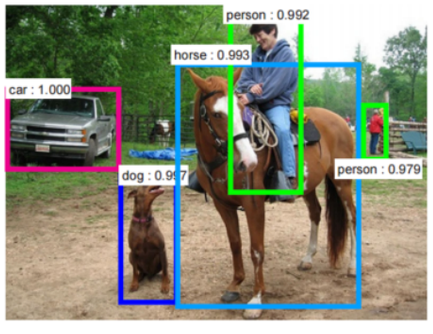
- This presentation was adapted from the ComputeFest session on Deep Learning by **Andrea Azzini** and **Giovanni Conserva**.

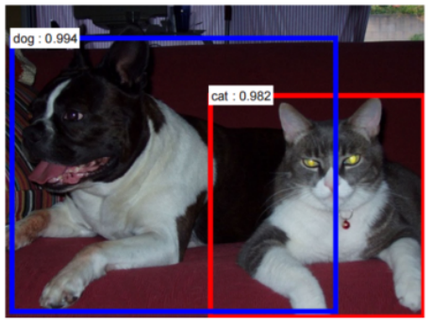
- Find their materials here: https://github.com/andreaazzini/DL_computefest_2017

Introduction

Intro: Relevance

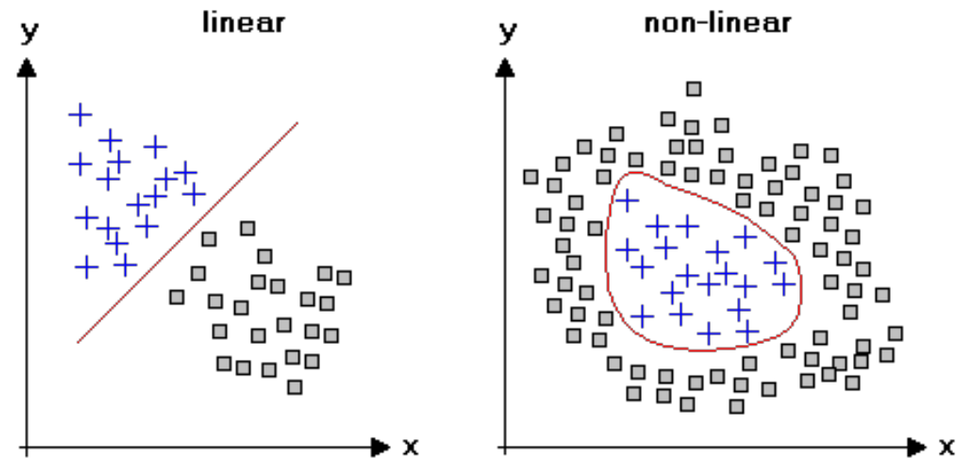
- Neural Nets immensely popular in the last 10 years
- Examples:
  - NLP / Speech recognition
    - Google Translate
  - Computer vision / Image classification
    - Self-driving cars
  - Time series prediction
    - Stock market



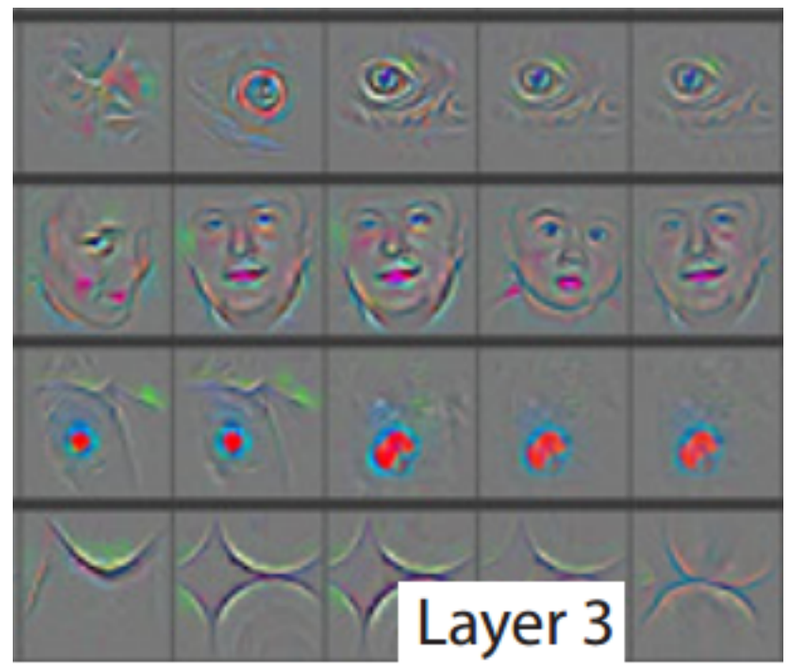


Intro: Motivation

- Why so *effective*?
  - Capture non-linearity
  - Nonparametric
  - Automated feature extraction from raw data

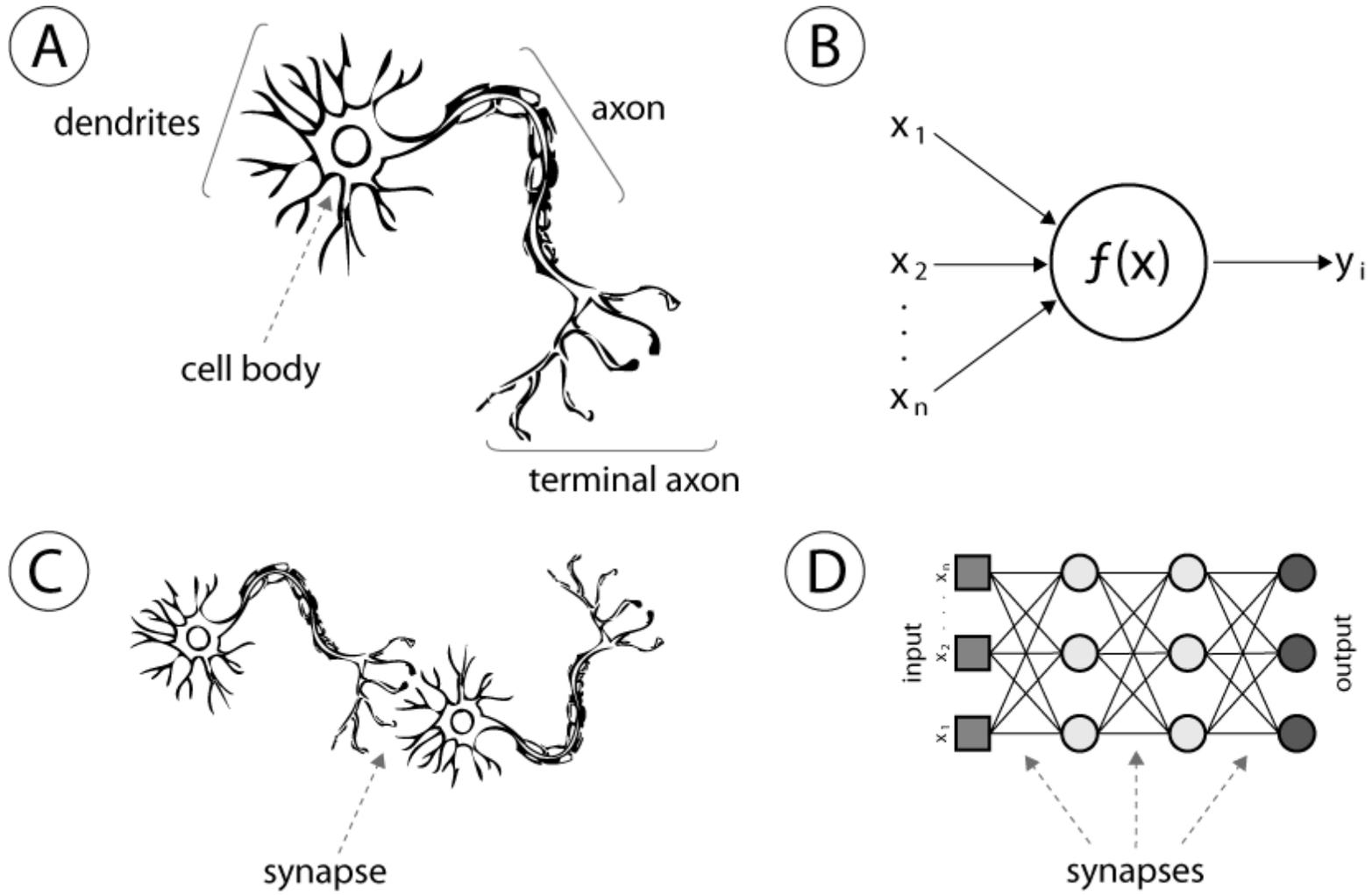


- Why *now*?
  - Compute power (GPUs etc.)
  - Advances in optimization
  - Data availability



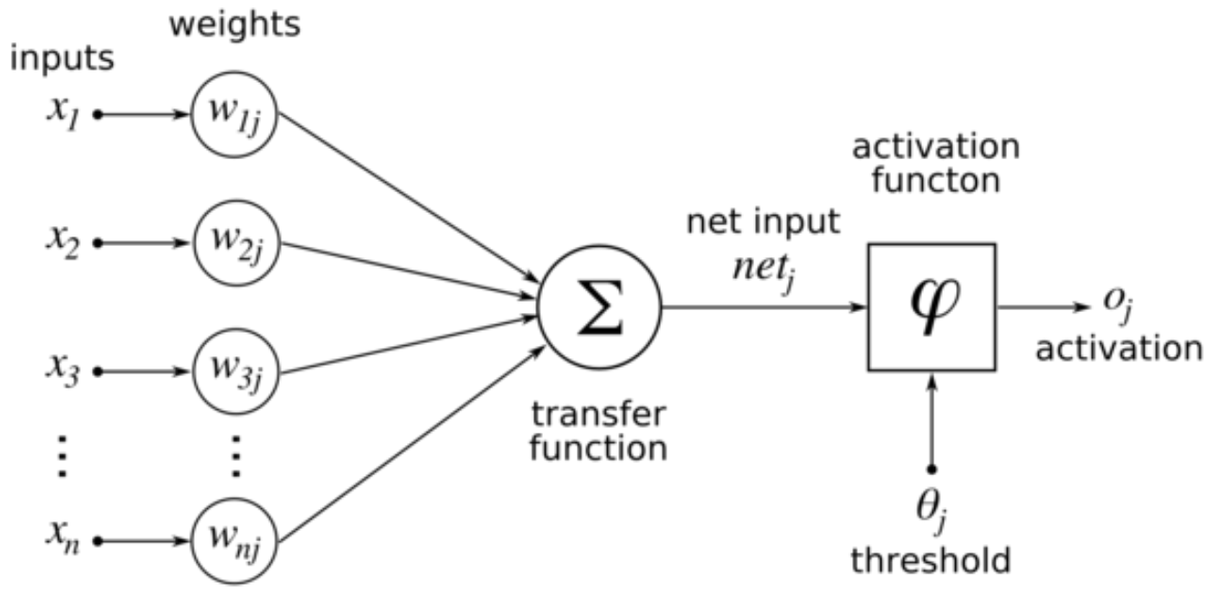
Intro: Concept (I)

- *Neural net*: a collection of connected individual neurons
- Loosely similar to biological neurons



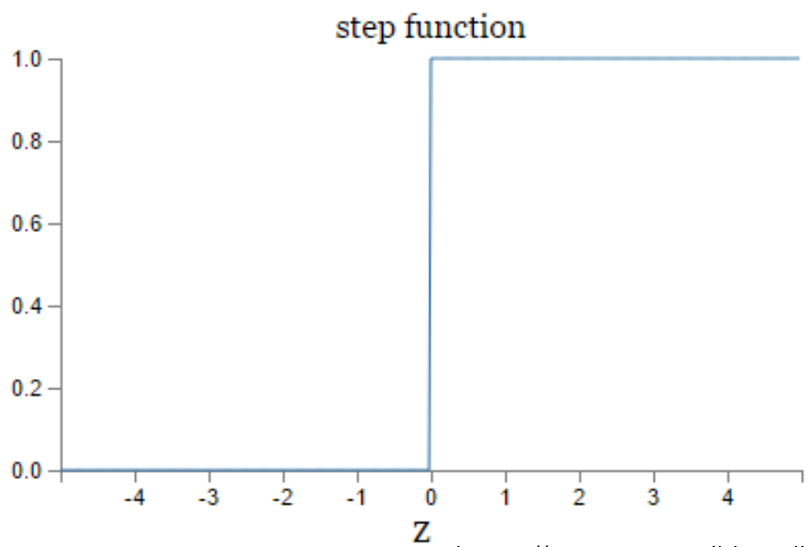
http://www.intechopen.com/source/html/39067/media/image1.png

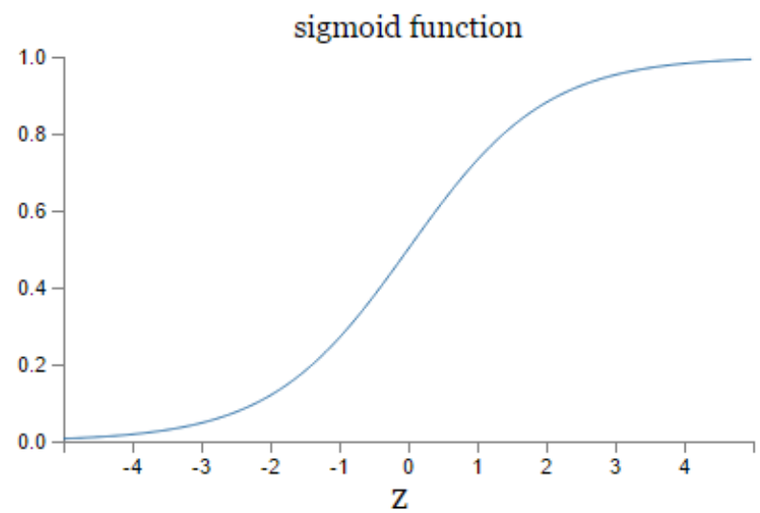
Intro: Concept (2)



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

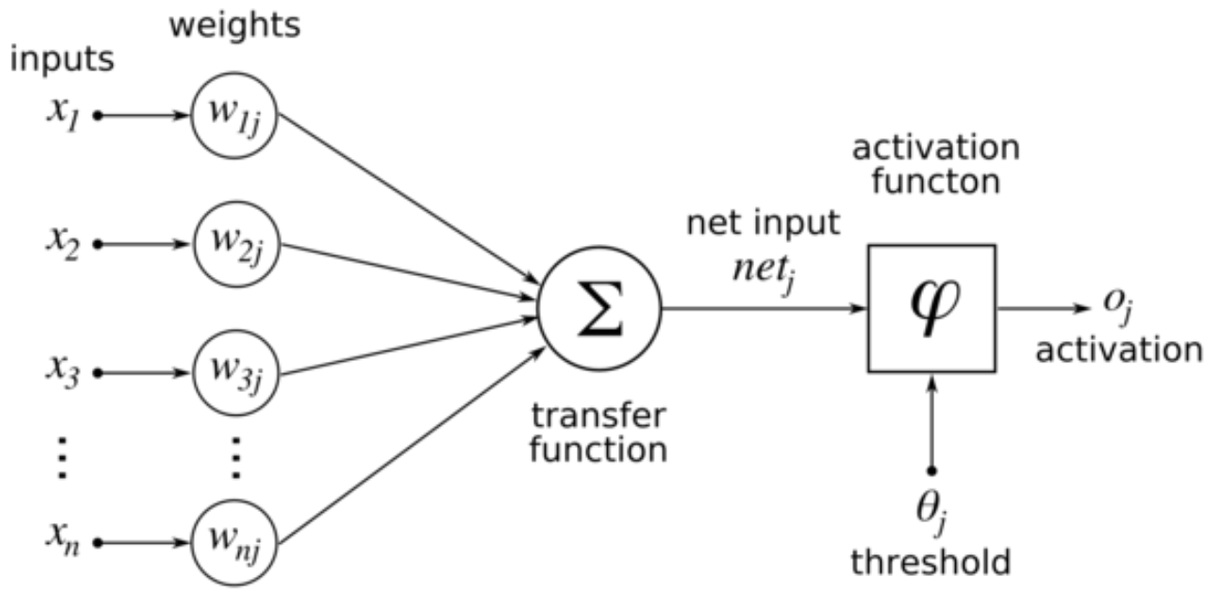
$$\text{output} = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$





https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png

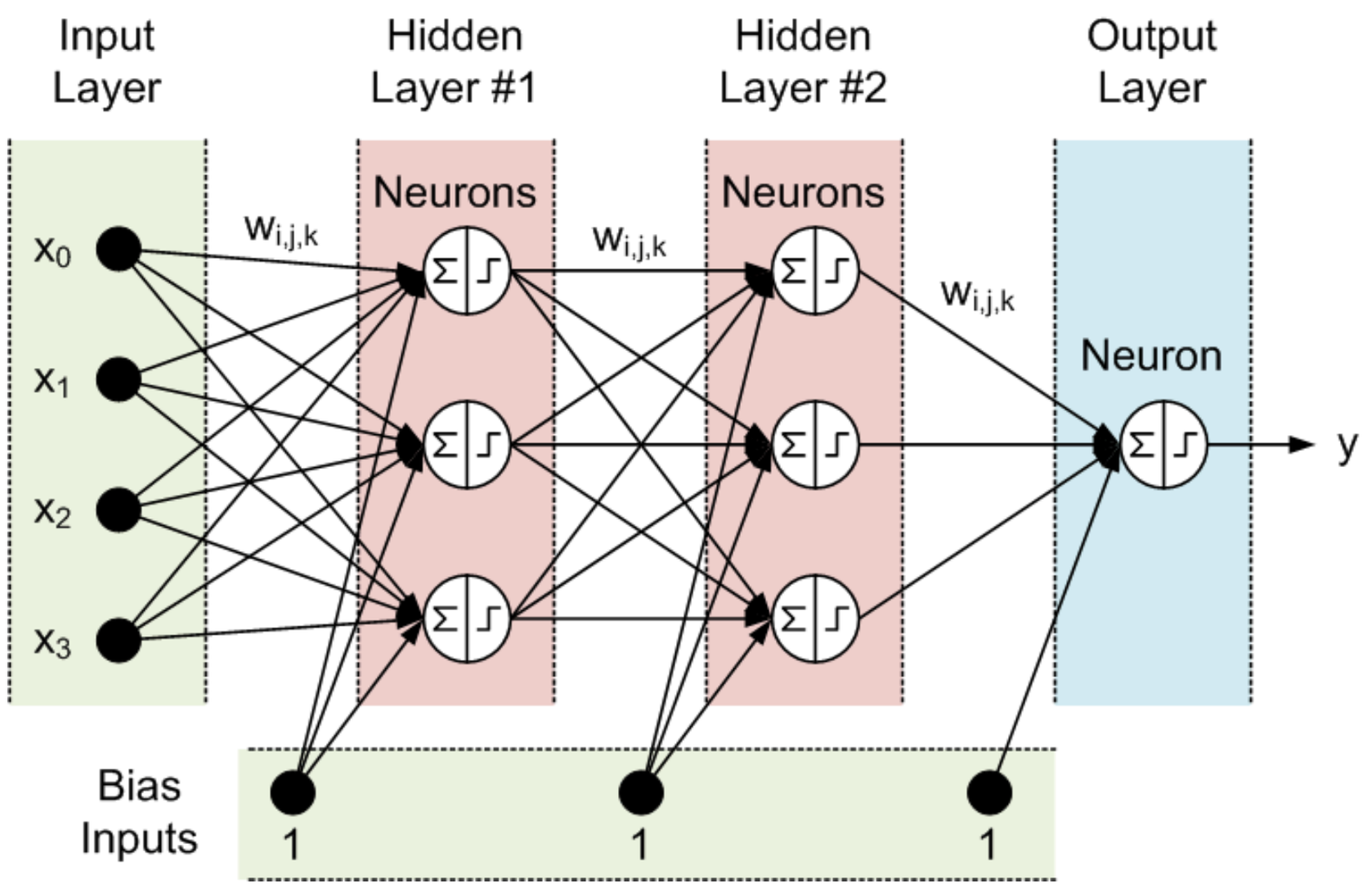
Intro: Concept (3)



- Compare this to a logistic regression (classification)
  - *Objective*: find weights that minimize error
  - *Transfer function*: Summation
  - *Activation function*: Logistic function

Intro: Concept (4)

- Many neurons linked together: *neural net*
- Many architectures to choose from (MLP, CNN, RNN, LSTM, …)



How to train neural nets

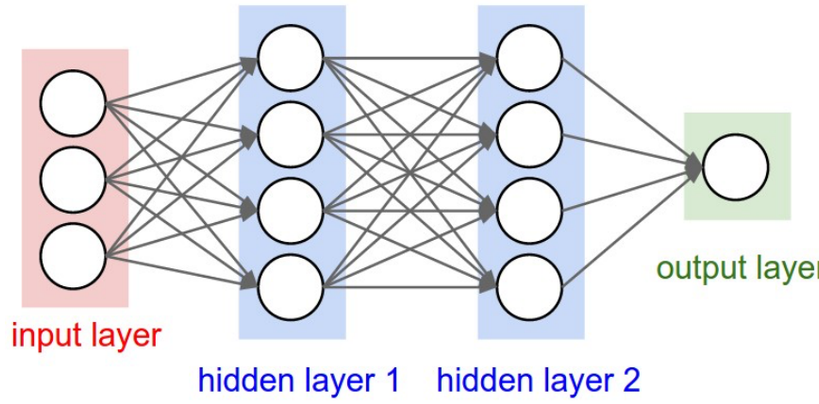
Step-by-step

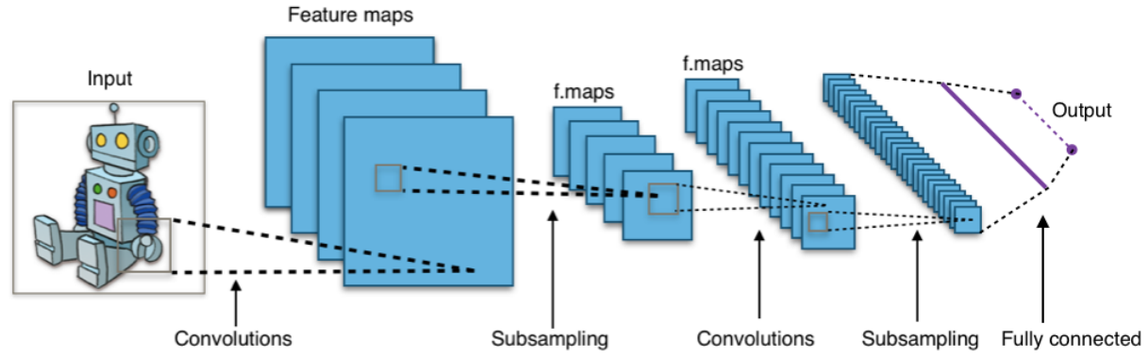
Step 0: Preprocess data

- For **classification**:
  - Put input data in vector / matrix / tensor
    - E.g. Dataframe with samples
    - E.g. Image pixels into vector or tensor
- Put labels in 'one-hot encoding'
  - E.g. `[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]` for Label 7

Step 1: Build Network

- There are many different network architectures
  - *MLP* or 'Fully-connected' network
  - *CNN*: tailored to pattern recognition
  - *RNN*: tailored to sequential data
  - *LSTM*: NLP and time series
- Choose number of hidden layers, and other params
- Mostly driven by rules of thumb

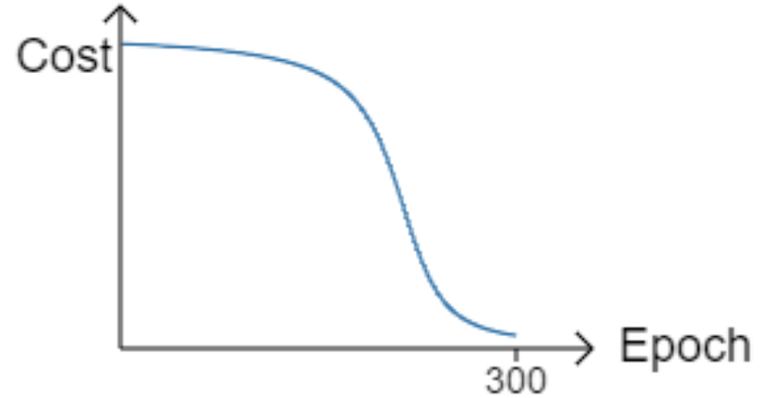




Step 2: Choose Loss Function

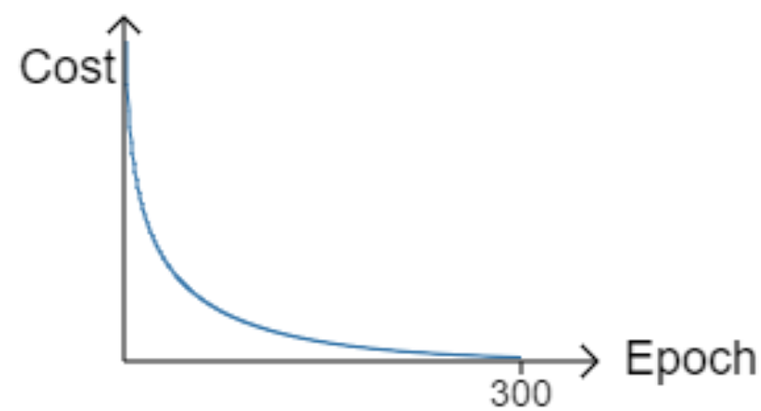
- Loss is how you evaluate performance, e.g.:
- Quadratic loss function

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2$$



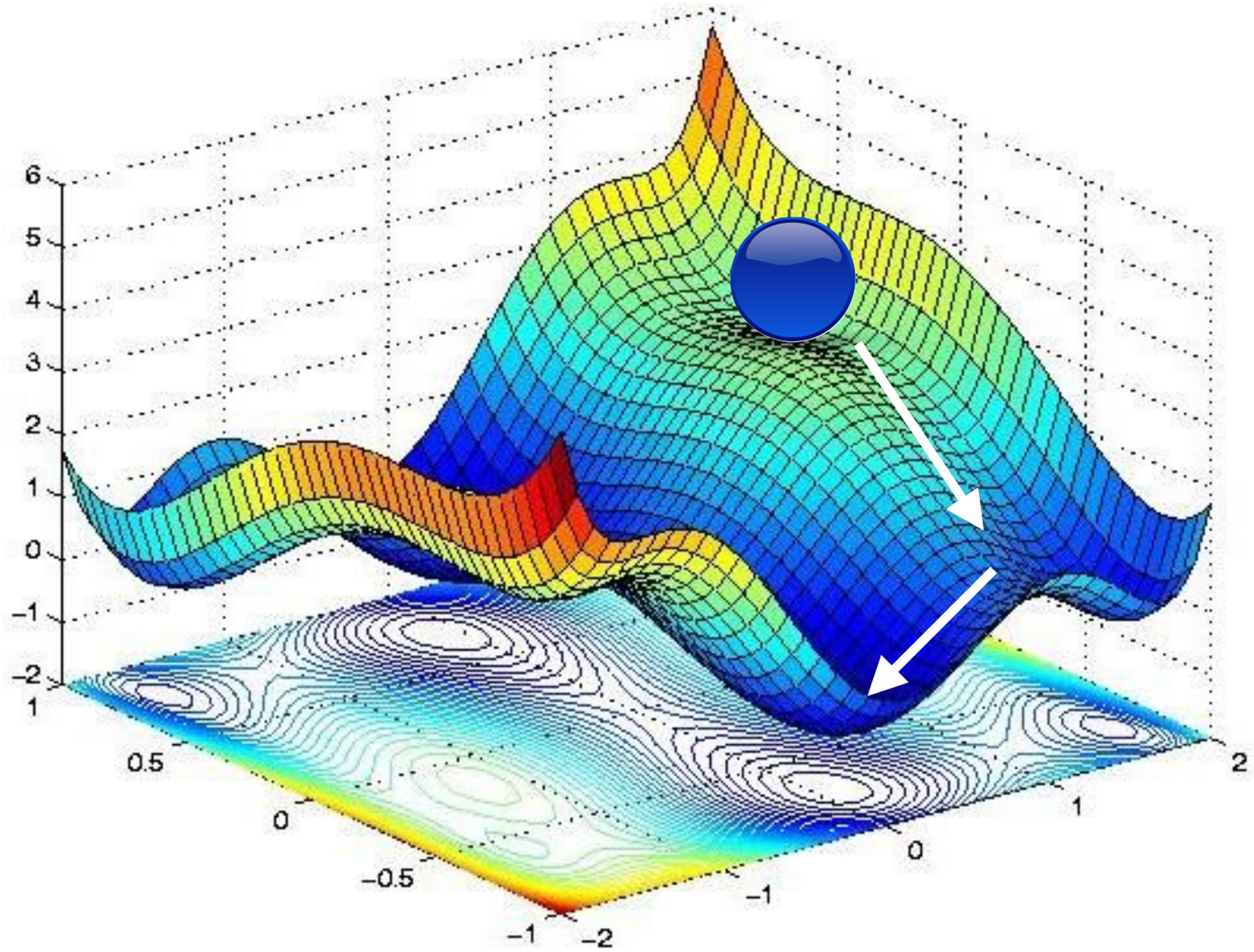
- Cross-Entropy loss function

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln (1-a)]$$



Step 3: Gradient Descent

- Objective of training:
  - *Find the weights that minimize the loss function*
- How?
  - *Gradient Descent!*
- Move in downward direction
- Can end up in local optimum!



Step 3: Gradient Descent (2)

- Need derivatives w.r.t. weights

$$\frac{\partial C}{\partial w_k} \qquad \frac{\partial C}{\partial b_l}$$

- Every time take a little step in downward direction

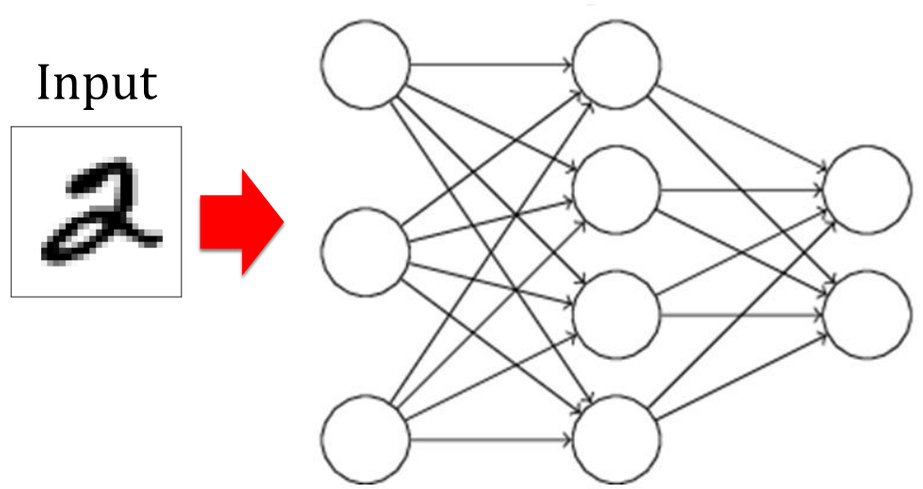
$$w_k \to w_k' = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b_l' = b_l - \eta \frac{\partial C}{\partial b_l}.$$

- Can add momentum to overcome 'bumps'

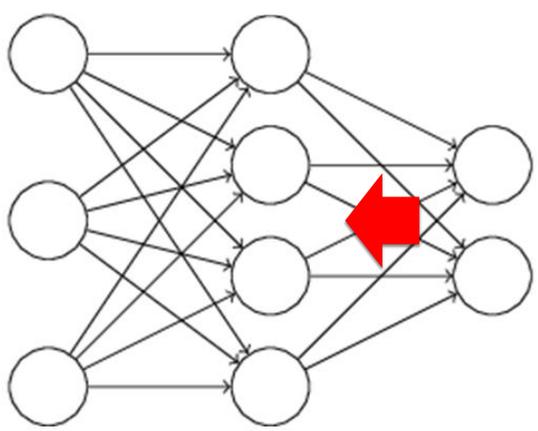
Step 3: Gradient Descent (3)

- The algorithm to determine these partial derivatives is called: **Backpropagation**



A. *Compute error from input (Forward step)*

B. *Compute gradient from error, and update weights (Backward step)*



Step 3: Gradient Descent (4)

$$z^l \equiv w^l a^{l-1} + b^l$$

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$$

BP 1: Error in the output (final) layer

BP 2: Error in a layer as a function of the error in its next layer:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

BP 3: Rate of change of the error wrt to any weight

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

BP 4: Rate of change of the error wrt to any bias

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$

Full example: https://github.com/andreaazzini/DL_computefest_2017/blob/master/Slides/i.pdf

Step 4: Evaluate Performance

- Assess best network by lowest validation accuracy

- Use model to predict on test set
- Obtain test score

Visual demo

How does everything add up?

Click here

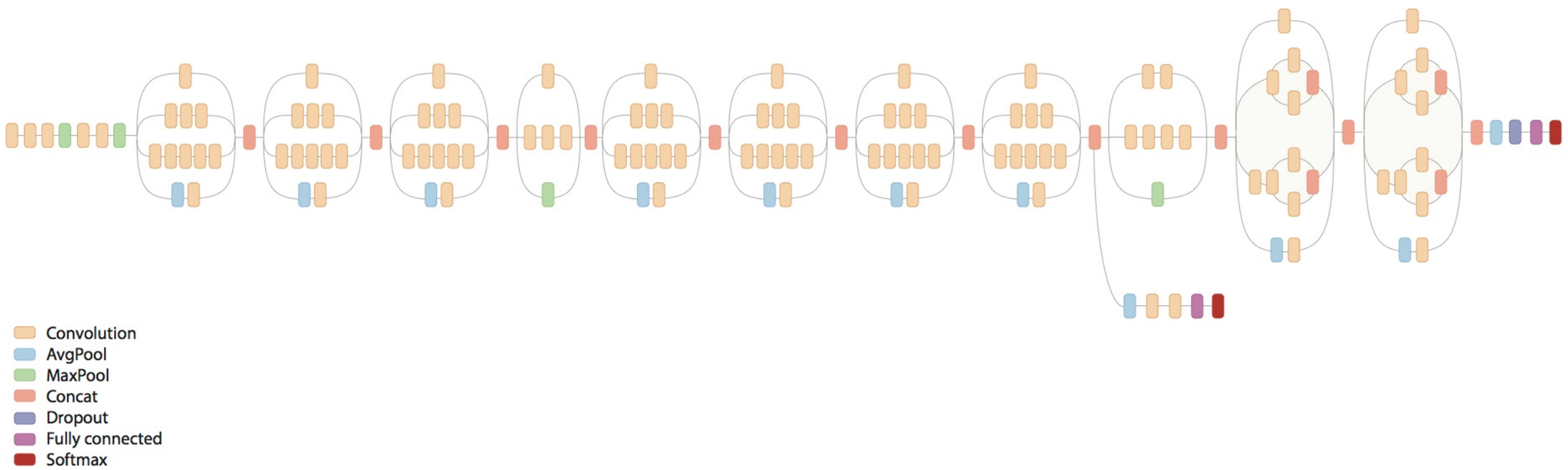
Packages

Packages

- Torch
- Caffe
- Keras
- Theano
- TensorFlow
- … many more!

Packages (2)

- The idea behind a lot of these packages is defining a **computational graph**, rather than running stuff right away
- Symbolic differentiation for gradient descent
- Irrespective of hardare type (CPU vs. GPU)



Digit Recognition Example

In TensorFlow (MNIST)