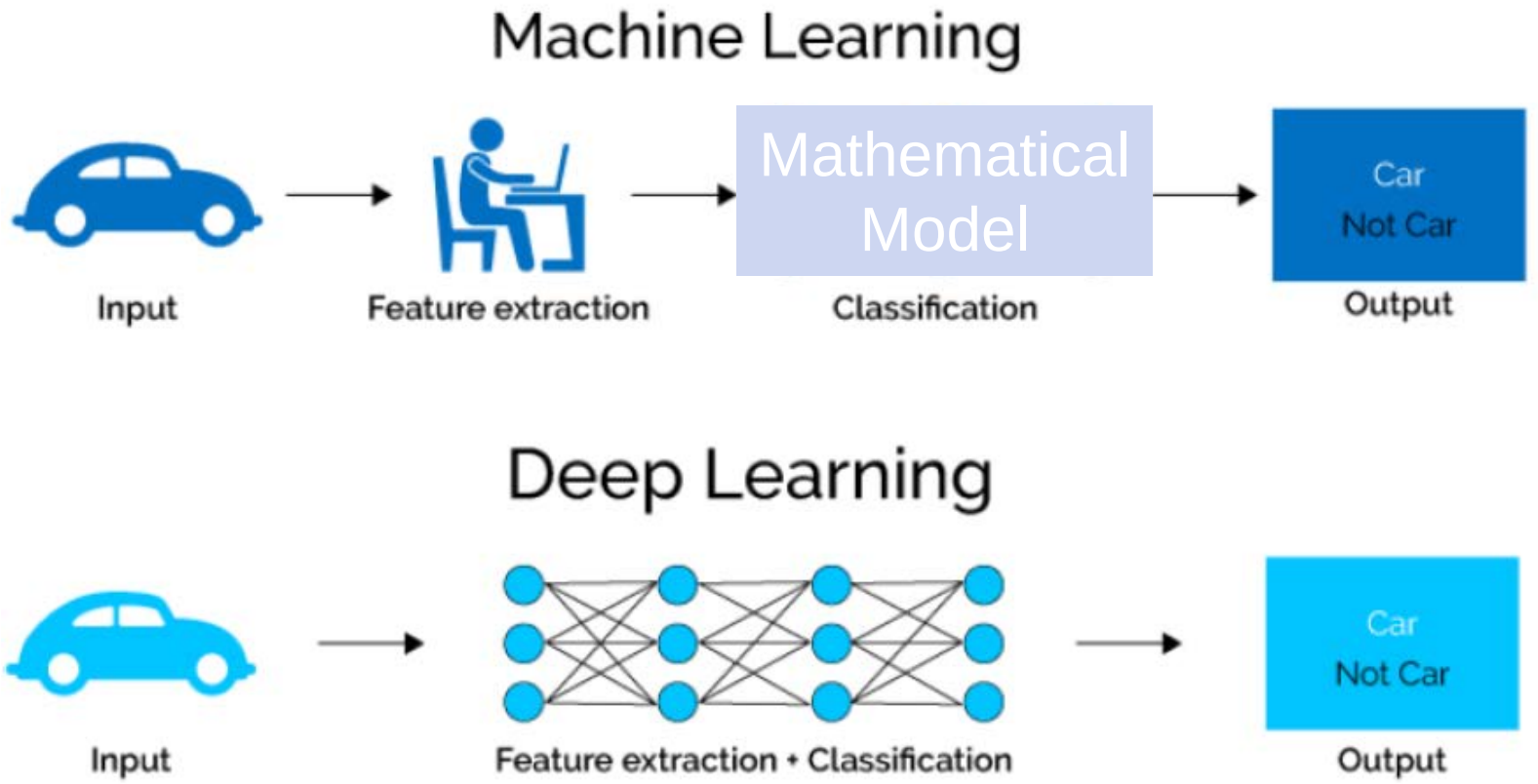
Outline

- Introduction
  - Terminology: AI / ML / DL;
  - AI4Vision:
    - From pixels to semantic;
    - Two paradigms: (AI or ML) vs. DL.

- Hand-crafted image features
  - Pixel- or patch-based features;
  - Interpreting the features:
    - AI: 3D Geometry;
    - AI: Graph-based topology;
    - ML: Random ferns, AdaBoost, Support Vector Machines.

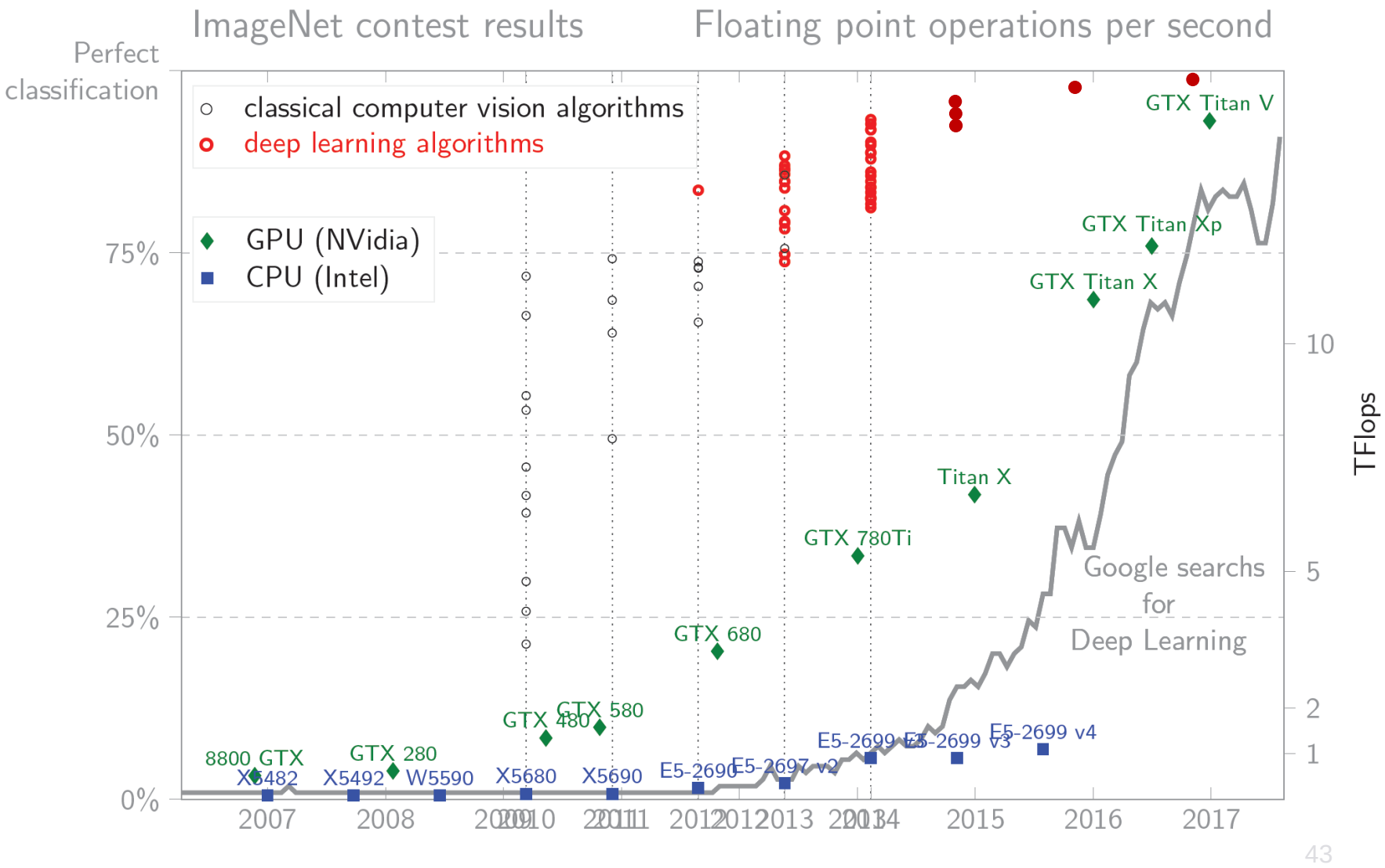
- **DL in vision : Convolutional Neural Networks (CNNs)**
  - **Paradigm shift & revolution;**
  - CNNs basics;
  - CNNs heads & backbones: network architecture examples;
  - CNNs mysteries.

Paradigm shift



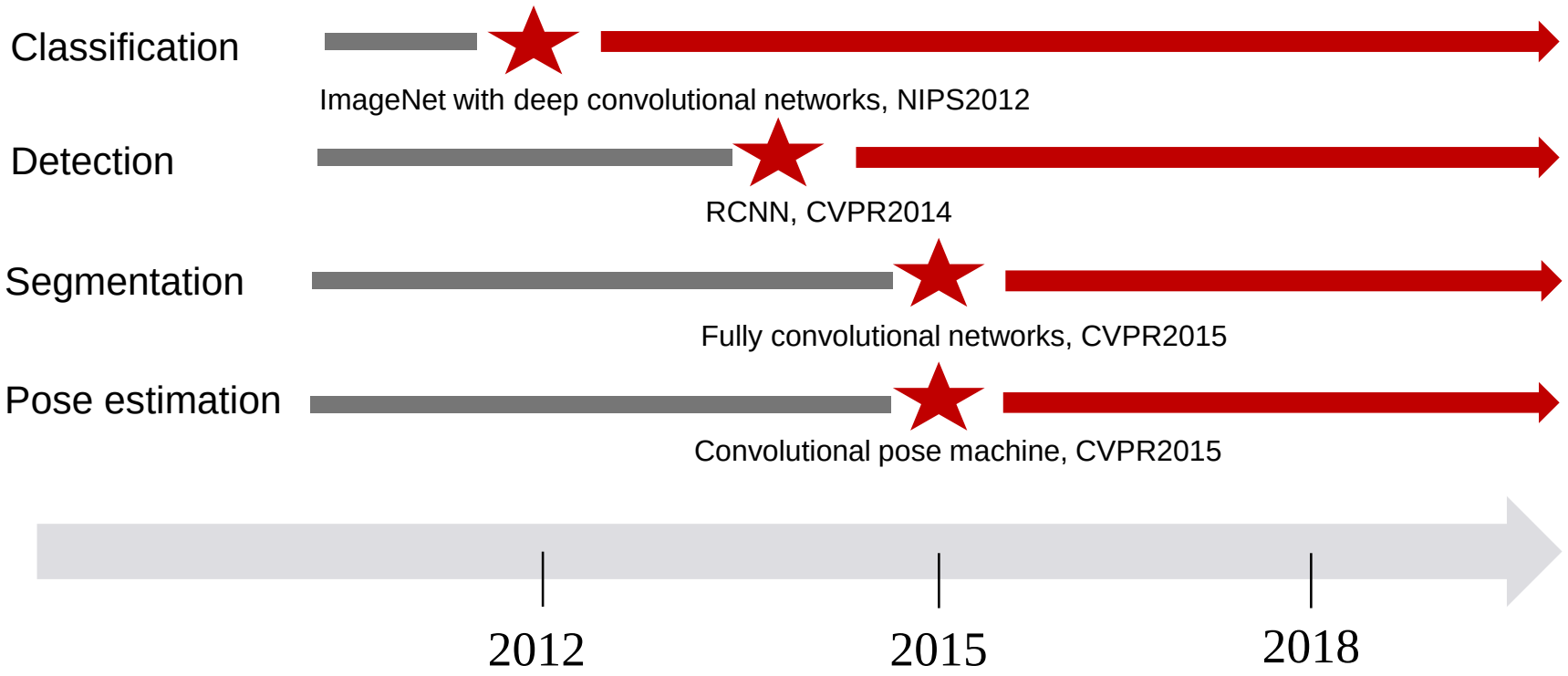
Deep learning revolution !

A representative use case: image classification



Emergence of Deep Learning

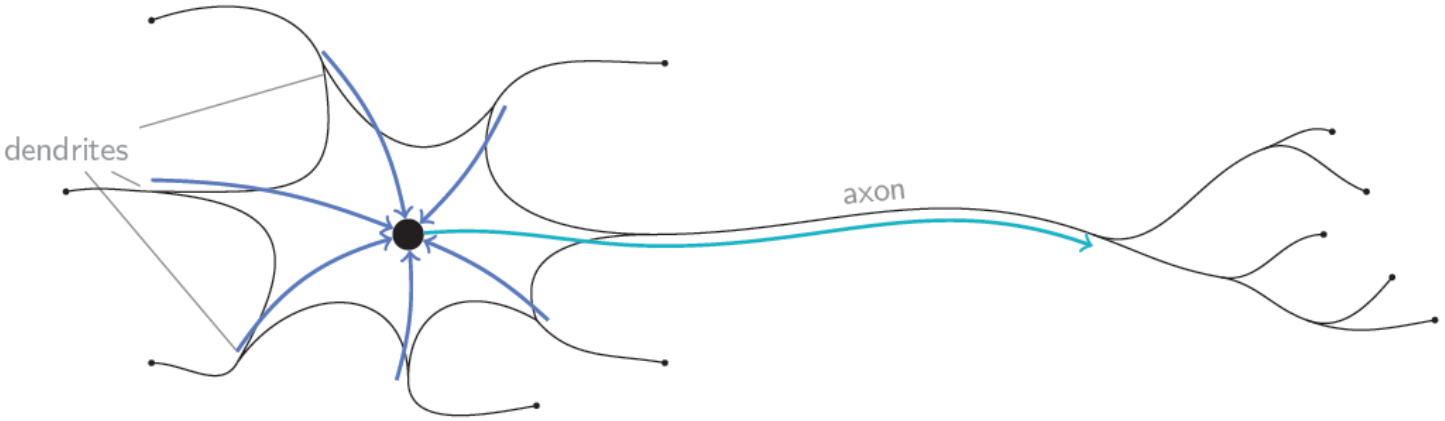
in many fields of computer vision.

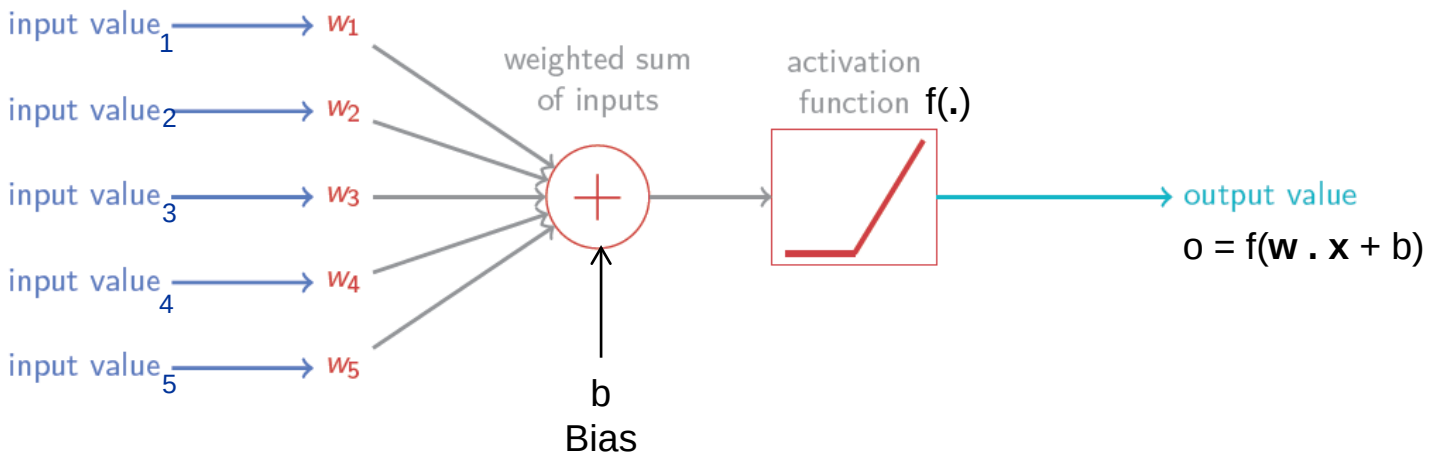


Outline

- Introduction
  - Terminology: AI / ML / DL;
  - AI4Vision:
    - From pixels to semantic;
    - Two paradigms: (AI or ML) vs. DL.
- Hand-crafted image features
  - Pixel- or patch-based features;
  - Interpreting the features:
    - AI: 3D Geometry;
    - AI: Graph-based topology;
    - ML: Random ferns, AdaBoost, Support Vector Machines.
- DL in vision : Convolutional Neural Networks (CNNs)
  - Paradigm shift & revolution;
  - **CNNs basics;**
  - CNNs heads & backbones: network architecture examples;
  - CNNs mysteries.

Neural networks basics





Neural networks basics

Multilayer perceptron, e.g. for predicting house prices

Taining sample

Output = price

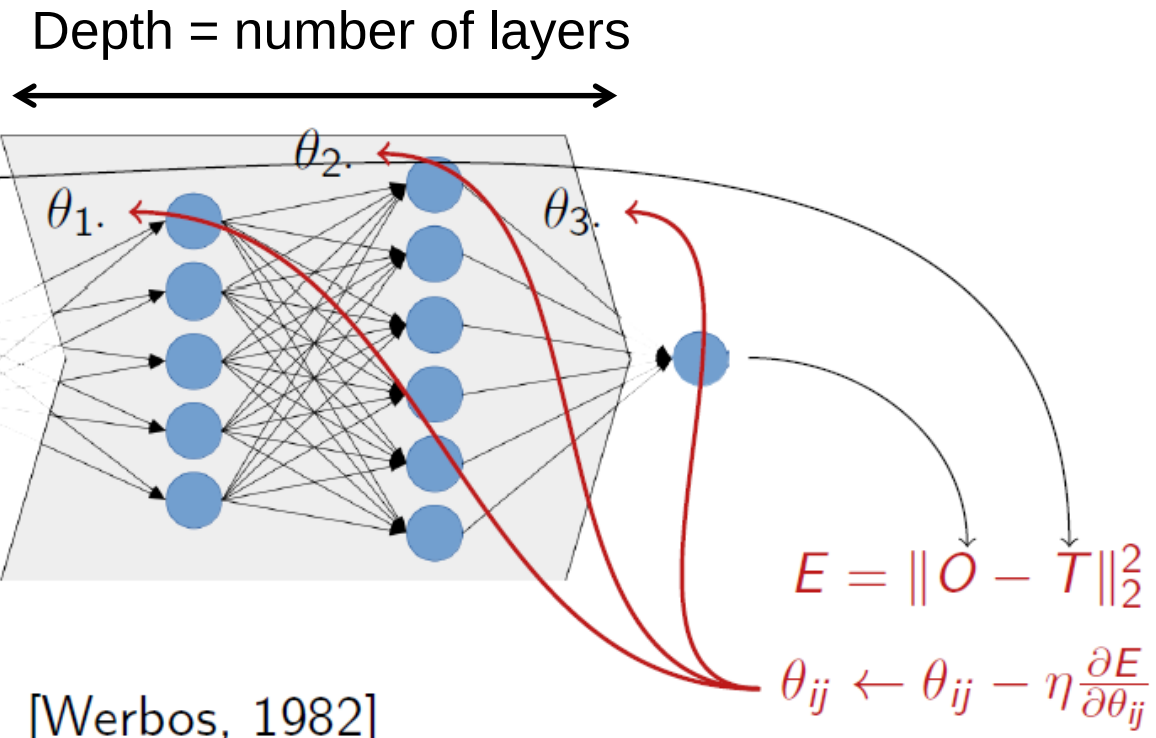
Inputs

=

YearBuilt,

SqMeters,

GPS coordinates → …



[Hornik et al, 1988] [Werbos, 1982]

θ = NN weight and bias parameters.

Learning with gradient descent

- *N* training samples (x_i,y_i).
- ***Loss or cost function***, e.g.

$$C(w,b) = \sum_i \left\| y_i - o(x_i) \right\|^2$$

Sum over all training samples.

- Gradient descent:

$$w \leftarrow w - \eta \cdot \frac{\partial C(w,b)}{\partial w}$$

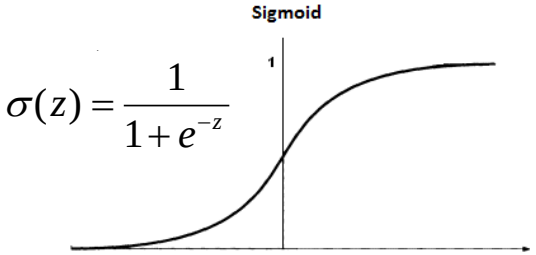
$$b \leftarrow b - \eta \cdot \frac{\partial C(w,b)}{\partial b}$$

with η = learning rate (small and positive).

O Loss function *C* definition:

Analysis for a single input sigmoid neuron:

- $o(x) = \sigma(w \cdot x + b)$



- $C = (y\text{-}o(x))^2$?

$$\frac{\partial C}{\partial b} \sim (y\text{-}o).\ \sigma'(w \cdot x + b)$$

→ Slow learning when output saturates on wrong value (e.g. due to poor init.).

- How to **make gradient high when prediction is bad**? **! Key principle !**

$$\text{We want}: \frac{\partial C}{\partial b} = (o - y), \text{ with } o = \sigma(w \cdot x + b).$$

σ′ when σ is a sigmoid

$$\text{We have}: \frac{\partial C}{\partial b} = \frac{\partial C}{\partial o} \cdot \sigma'(w \cdot x + b) = \frac{\partial C}{\partial o} \cdot \sigma(w \cdot x + b) \cdot (1 - \sigma(w \cdot x + b))$$

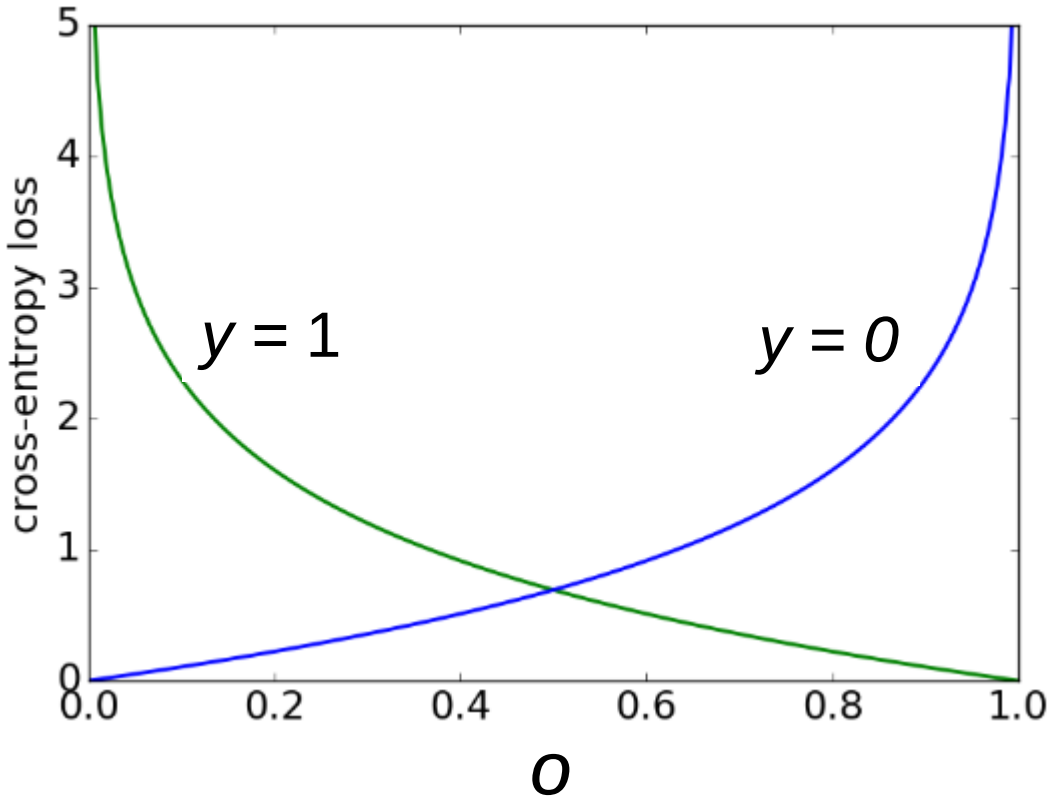
$$\Rightarrow \frac{\partial C}{\partial o} = \frac{(o - y)}{o \cdot (1 - o)} \Rightarrow C = -[y \cdot \ln(o) + (1 - y) \cdot \ln(1 - o)] + \text{cte}$$

In general for *n* training samples and *d* output neurons, the cross-entropy writes:

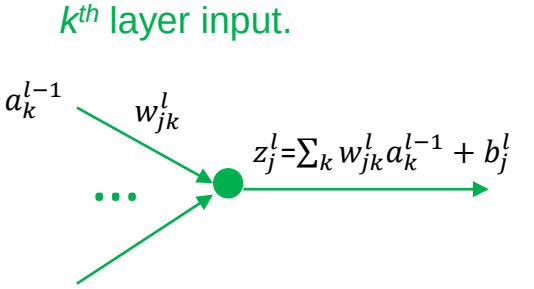
$$C = -\frac{1}{n} \sum_{k=1}^{n} \sum_{j=1}^{d} [y_j \cdot \ln(o_j) + (1 - y_j) \cdot \ln(1 - o_j)]$$

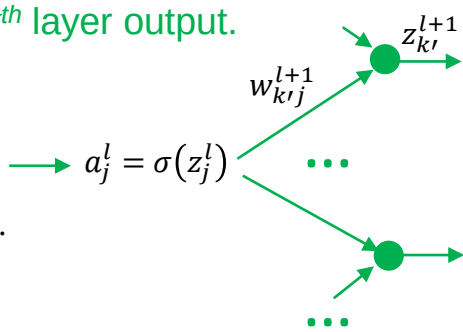
Plotting the cross-entropy loss confirms that the loss induces larger gradients for samples associated to worse predictions.

As a consequence, the network updates will primarily favour the improvement of the worse predictions.



Gradient descent with back-propagation





Definitions:

$z_j^l = \text{pre-activation } j^{th} \text{ neuron in layer } l.$

$a_j^l = \sigma(z_j^l) = \text{activation } j^{th} \text{ neuron in layer } l.$

$w_{jk}^l = \text{link from } k^{th} \text{ neuron in layer } l\text{-}1 \text{ to } j^{th} \text{ in layer } l.$

$b_j^l = \text{bias for } j^{th} \text{ in layer } l.$

Notations:

$\mathbf{z}^l, \mathbf{a}^l, \mathbf{b}^l$ denote vectors whose j^{th} component are z_j^l, a_j^l, b_j^l, respectively.

$\mathbf{W}^l$ denotes a matrix, with $(\mathbf{W}^l)_{i,j} = w_{i,j}^l \quad \rightarrow \quad \mathbf{z}^l = \mathbf{W}^l \cdot \mathbf{a}^{l-1} + \mathbf{b}^l$.

$\circ$ = Hadamard product, i.e. $(\mathbf{x} \circ \mathbf{y})_{i,j} = (\mathbf{x})_{i,j} \cdot (\mathbf{y})_{i,j}$.

Vector of partial derivatives: $(\nabla_{\mathbf{v}} \sigma)_i = \dfrac{\partial \sigma}{\partial v_i}$.

Functions of vectors are applied component by component: $(\sigma(\mathbf{v}))_i = \sigma(v_i)$.

Back-propagation equations (based on chain rule):

What we need to update the network parameters.

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \equiv \delta_j^l$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot a_k^{l-1} = \delta_j^l \cdot a_k^{l-1}$$

$$\frac{\partial C}{\partial z_j^l} = \sigma'(z_j^l) \cdot \frac{\partial C}{\partial a_j^l} = \sigma'(z_j^l) \cdot \sum_{k'} \left(\frac{\partial C}{\partial z_{k'}^{l+1}} \cdot w_{kj}^{l+1} \right) \quad , \; l < L \quad \rightarrow \quad \boldsymbol{\delta}^l = \sigma'(\mathbf{z}^l) \circ \left((\mathbf{W}^{l+1})^T \cdot \boldsymbol{\delta}^{l+1} \right)$$

$$= \sigma'(z_j^L) \cdot \frac{\partial C}{\partial a_j^L} \quad , \; l = L \quad \rightarrow \quad \boldsymbol{\delta}^L = \sigma'(\mathbf{z}^L) \circ \nabla_{\mathbf{a}} C$$

Gradient descent in practice:

Mini-batches:

In practice, only a small random subset of samples (=mini-batch) is considered at each gradient descent step.

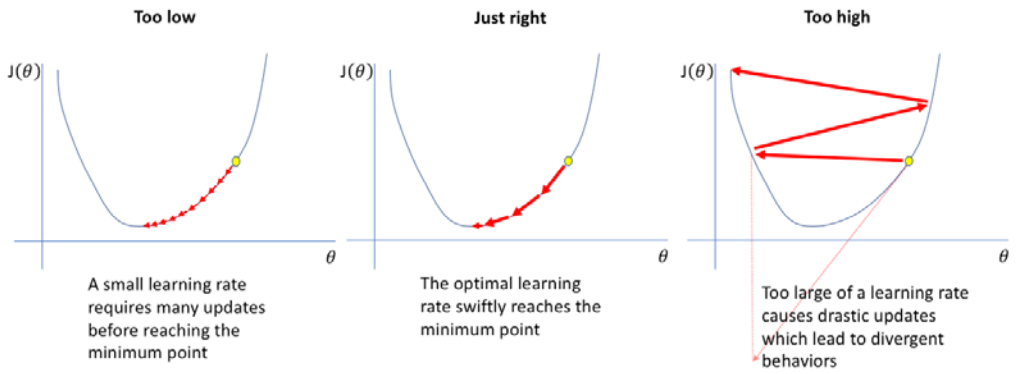
➡ ***Stochastic Gradient Descent*** (SGD)

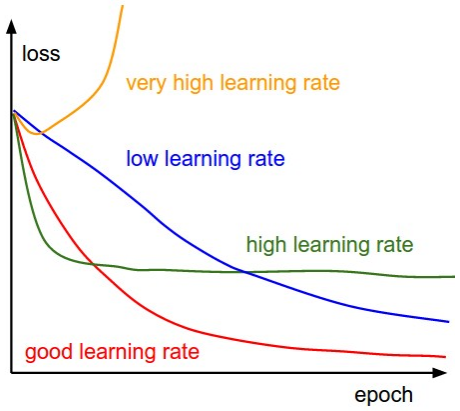
One epoch = one complete pass over the training set.

After each epoch the samples are usually shuffled so as to avoid cyclical repetitions.

Learning rate:

- In general:





- Learning rate:
  - In practice:

    Cool down – or decay – the learning rate over time.

    Use previous gradients to define current one,
    e.g. momentum update average gradients over time as follows

$$\mathbf{v}^{(m+1)} = \mu \mathbf{v}^{(m)} - \eta \frac{\partial E}{\partial \mathbf{w}}(\mathbf{w}^{(m)})$$
$$\mathbf{w}^{(m+1)} = \mathbf{w}^{(m)} + \mathbf{v}^{(m+1)}$$

with $\mu \approx 0.9$

    Other techniques: Nesterov's momentum, Adam, Nadam, RMSprop.
    Idea: increase the update strength for parameters that have had smaller updates in the past.

  - ! Speed is not everything :

    learning rate also appears to impact the generalization !

    (see 'CNN mysteries' later in the slides)

How to train with the many parameters of deep nets ?

- Weights initialization: critical to avoid exploding/vanishing gradients
  - Gaussian process with zero mean.
  - Choose Gaussian variance inversely proportional to number of weights, so that each feature has similar variance.

- Regularization: improves generalization (= test time accuracy)
  - Data augmentation (e.g. easy to associate several inputs to a category by cropping) and/or multi-task.
  - L2 regularization:

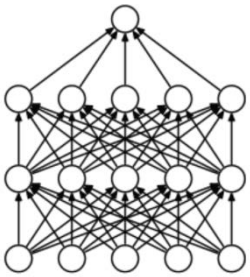
$$C_\lambda = C + \frac{\lambda}{2n}\sum w^2$$

$$w \leftarrow w - \eta \cdot \frac{\partial C}{\partial w} - \frac{\eta \cdot \lambda}{n} w = (1 - \frac{\eta \cdot \lambda}{n}) \cdot w - \eta \cdot \frac{\partial C}{\partial w}$$

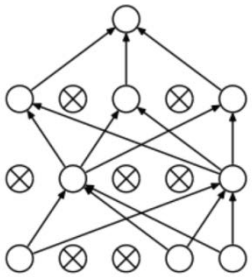
  - Batch-normalization (BN) is a layer that scales and shifts its inputs to normalize their distribution.

    At training, the μ and σ normalization parameters are set as the mean and standard deviation of one neuron over the mini-batch.
    At test time, μ and σ are averaged over the entire training set (for each neuron).
    BN helps in achieving higher learning rates and be less careful about optimization considerations such as initialization or Dropout.
    See details BN backpropagation @ https://chrisyeh96.github.io/2017/08/28/deriving-batchnorm-backprop.html

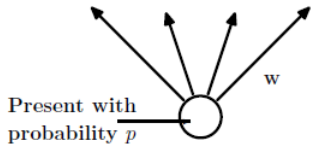
  - Dropout:



Standard Neural Net



After applying dropout.
Crossed units have been dropped.

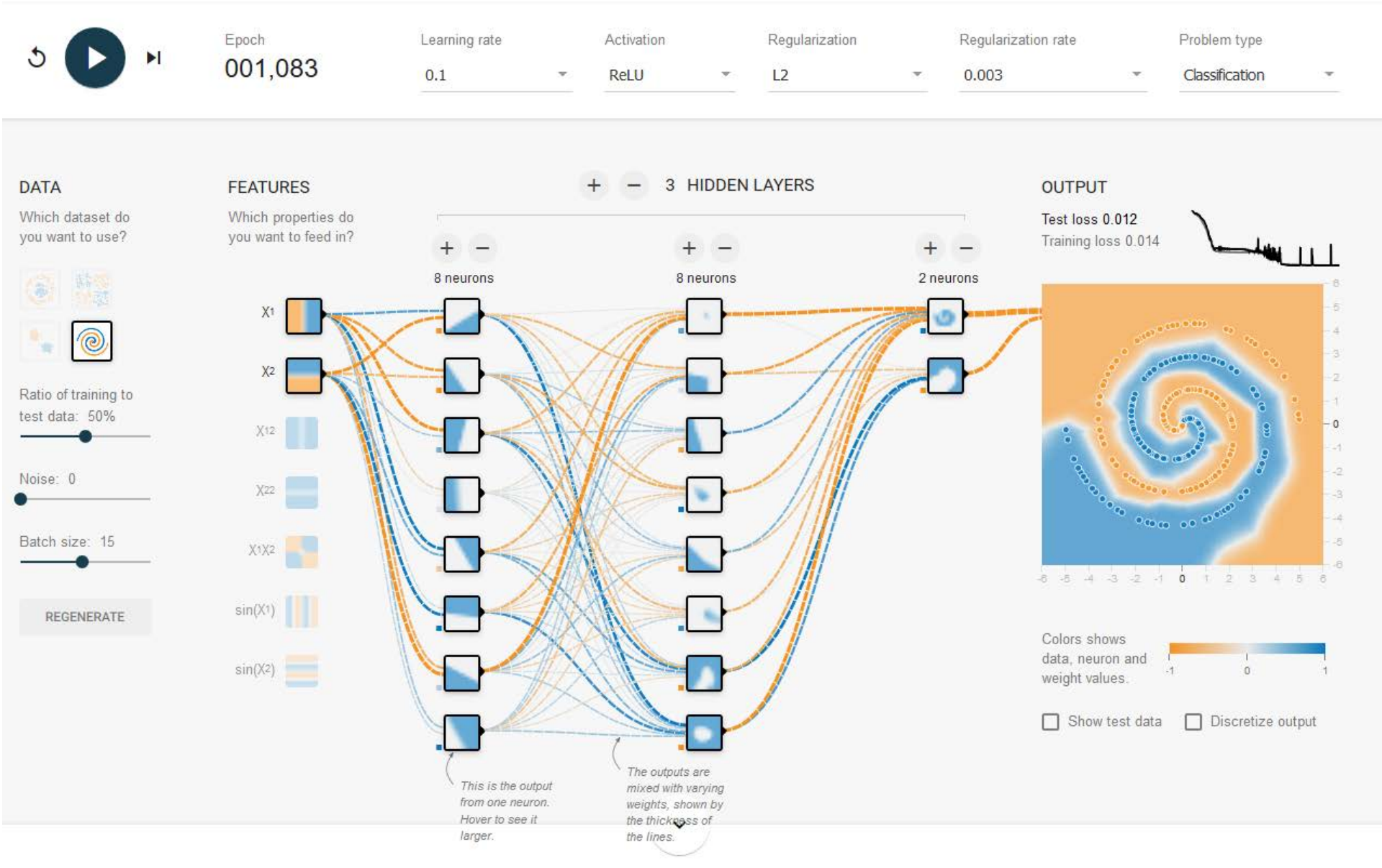


(a) At training time (b) At test time

2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, vol. 15, 2014.

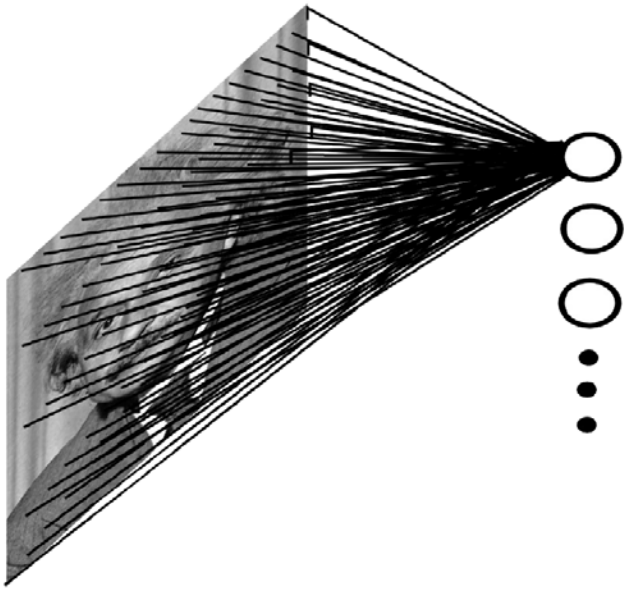
http://playground.tensorflow.org/



Convolutional Neural Networks (CNN)

- Image and Neural Nets:

  Naïve approach does not work



Fully connected NNs:

- Huge number of parameters;
- Does not explicitly model image structures locality and translation invariance.