Seminars in Artificial Intelligence and Robotics

Computer Vision for Intelligent Robotics

Basics and hints on CNNs

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

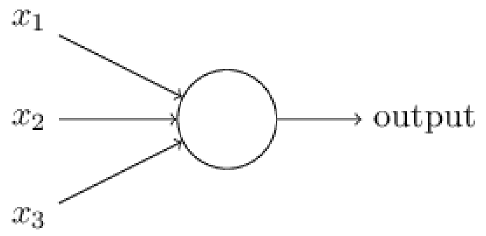
Sapienza Università di Roma

Alberto Pretto

What is a neural network?

We start from the first type of artifical neuron, the **perceptron**.

A perceptron takes several binary inputs, x1,x2,… , compute a weighted sum of the inputs and produces a single binary output using a fixed threshold:

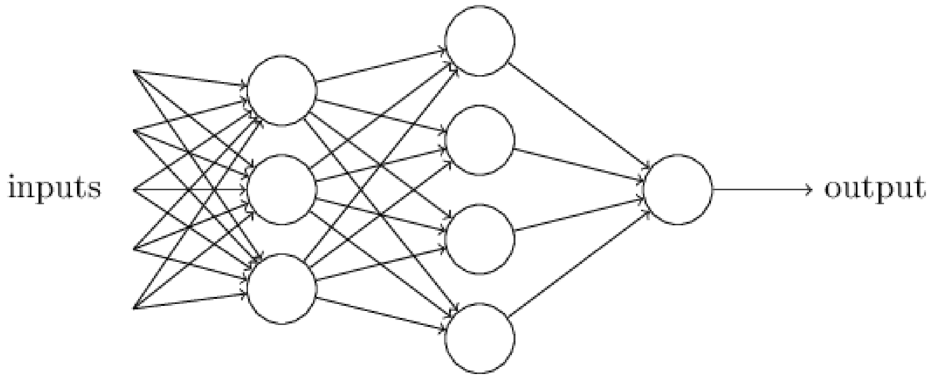


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

We can use the perceptron to take decisions: by varying the weights and the threshold, we can get different models of decision-making.

Multi-level perceptrons

More complex networks of perceptrons can deal with more complex decision problems:



The first column (i.e., the first **layer**)of perceptrons is making simple, low level decisions, by directly weighing the inputs. The perceptrons in the second layer is making a decision by weighing the results from the first layer: the second layer **can make a decision at a more complex and more abstract level**.

A **fully connected layer** (as in this case) is a layer where all its neurons have full connections to all the output in the previous layer.

Note: It is trivial to show that perceptrons can be used to sintetize logical functions (AND, OR, ecc...)

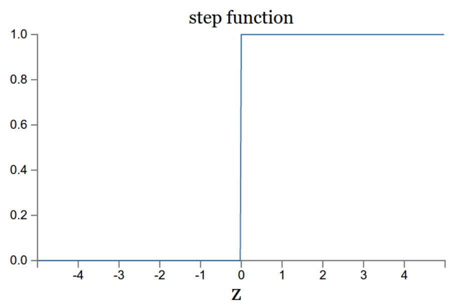
From perceptrons to artificial neuron

1) Write the weighted sum as dot product.

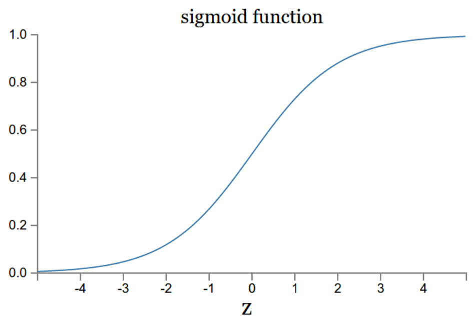
2) Replace the threshold with the bias b = -threshold

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

3) "Smooth" the output using the sigmoid function:



$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

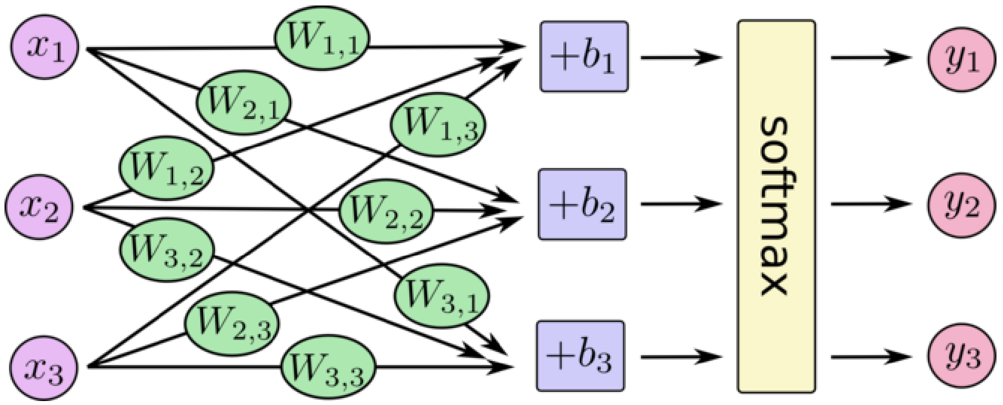


This is called a **sigmoid neuron**: that small changes in the weights and bias cause only a small change in the output. That's the crucial fact which will allow a network of sigmoid neurons to *learn*.

Softmax

From outputs to probability distribution:

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$



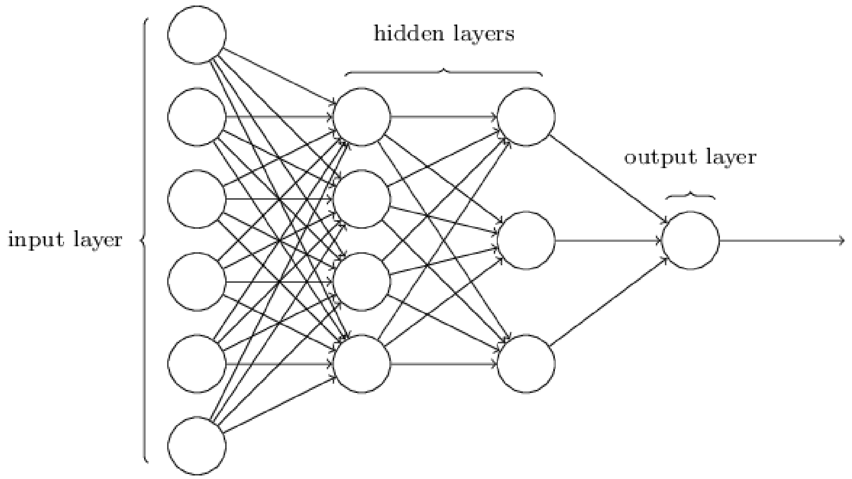
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

Toward deep learning

A deeper network (i.e., with hidden layers) can breaks down a very complicated question (e.g., does this image show a face or not) into simple questions

Networks with this kind of many-layer structure, two or more hidden layers, are **called deep neural networks**.

<u>Deep learning methods aim at learning feature hierarchies with features from higher levels of the hierarchy formed by the composition of lower level features</u>. [Glorot and Bengio]



Learning the network parameters

Given a labeled dataset x = {x1, x2, …} of inputs with associated outputs y(x) = {y(x1), y(x2), …}, find the weights *w* and biases *b* that minimize the cost function (a is the output of the network given the current parameters *w* and *b*):

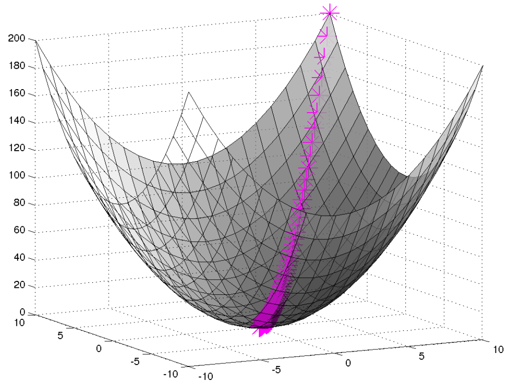
$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2 .$$

Easy answer! Gradient descent! → Correct but … very difficult implementation in practice, due to:

- Very large parameters set
- Very slow convergence rate
- Huge amount of data.
- Weight saturation
- ….

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

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



Solve the learning problem (no details)

Backpropagation

Stochastic gradient descent → Dataset divided in batches!

Massive parallelization

….

Backpropagation insights (1/2)

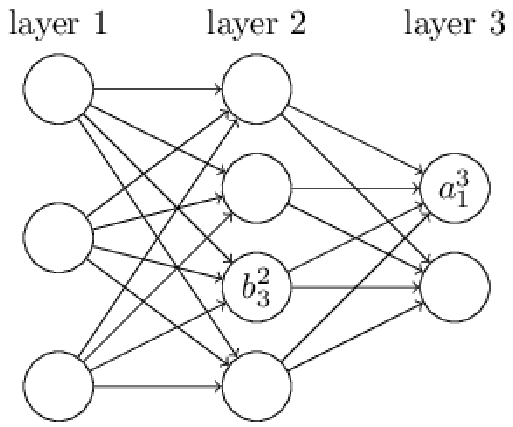
Goal: minimize $C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$

(This cost function can be written as an average over cost functions for individual training examples: $C = \frac{1}{n} \sum_x C_x$)

We need to compute *all* the partial derivatives $\frac{\partial C}{\partial w_k}$ and $\frac{\partial C}{\partial b_l}$

The goal of backpropagation is to compute **efficiently** these derivatives.

Backpropagation insights (2/2)



$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

$$\longrightarrow \quad a^l = \sigma(w^l a^{l-1} + b^l)$$

Let define the the “error” of a neuron *j* in layer *l* as $\delta_j^l \equiv \frac{\partial C}{\partial a_j^l}$

Backpropagation equations: $\longrightarrow$

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

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

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

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

The Backpropagation Algorithm

1. **Input x:** Set the corresponding activation a^1 for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

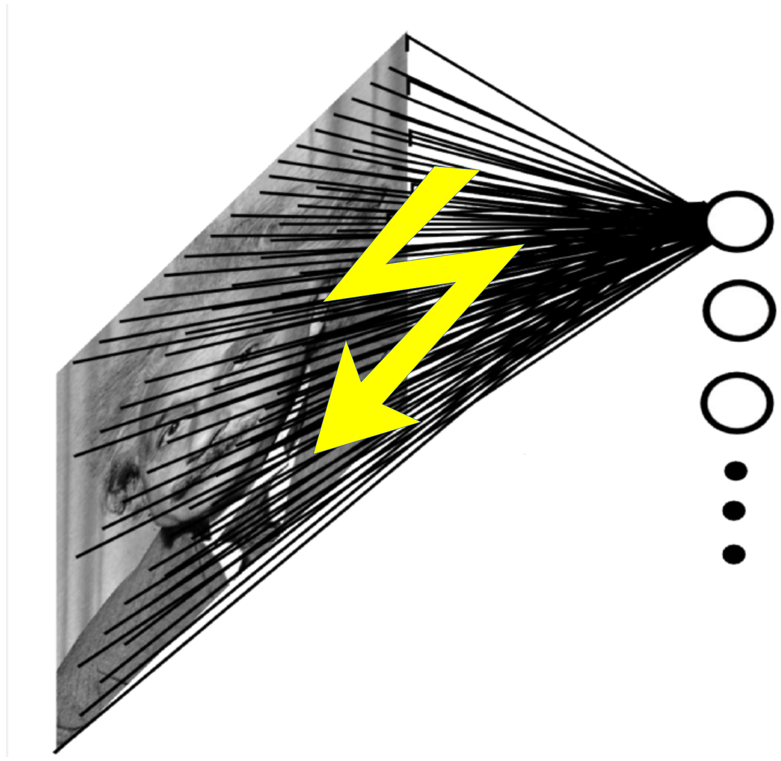
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

The full algorithm

1. **Input a set of training examples**
2. **For each training example x:** Set the corresponding input activation $a^{x,1}$, and perform the following steps:
   - **Feedforward:** For each $l = 2, 3, \dots, L$ compute $z^{x,l} = w^l a^{x,l-1} + b^l$ and $a^{x,l} = \sigma(z^{x,l})$.
   - **Output error $\delta^{x,L}$:** Compute the vector $\delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L})$.
   - **Backpropagate the error:** For each $l = L-1, L-2, \dots, 2$ compute $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$.
3. **Gradient descent:** For each $l = L, L-1, \dots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

From Neural Network to CNNs (1/2)

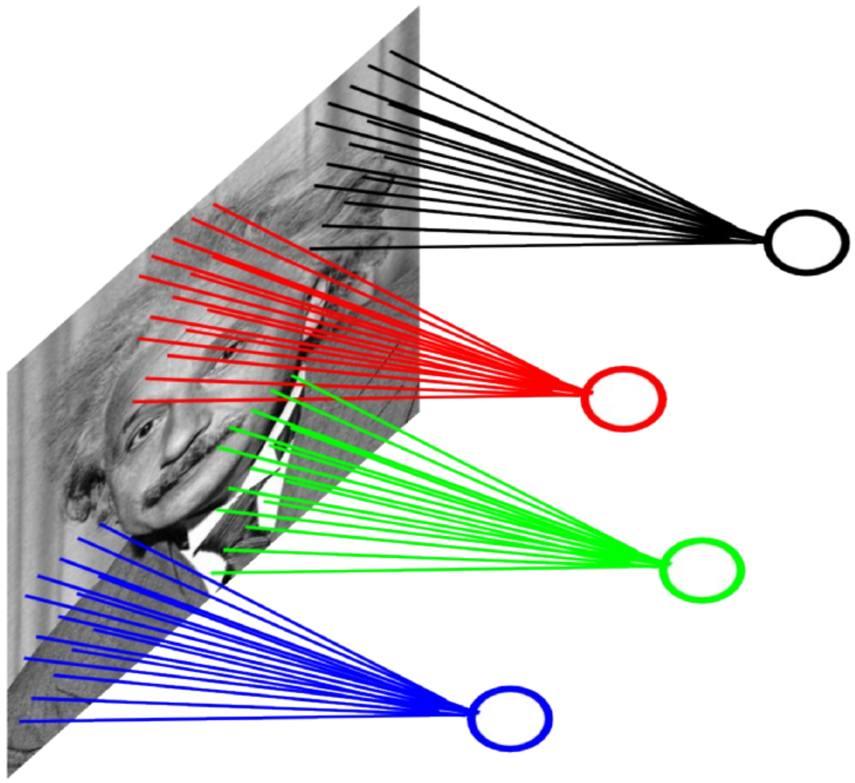
Apply NN to images to perform classification, detection,etc… using the classical “fully connected layers” but … for a 200x200 image ~**2B parameters**!!



Picture from M.A. Ranzato

From Neural Network to CNNs (2/2)

Convolutional neural networks use three basic ideas: **local receptive fields**, **shared weights**, and **pooling**.

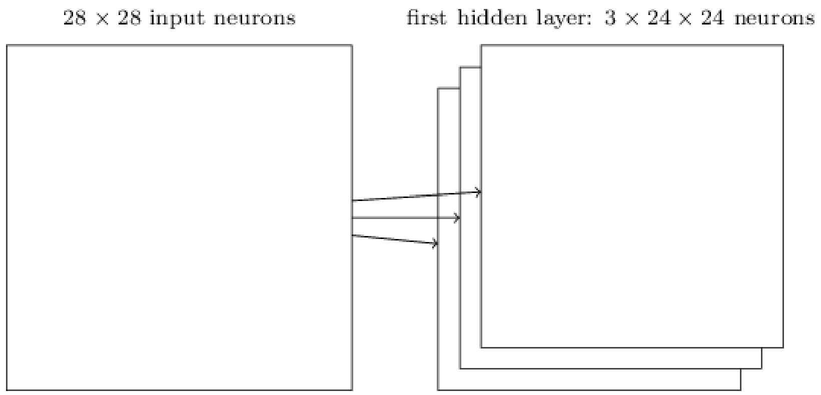


Convolutional layers

Local receptive fields and shared weights: all the neurons in the first hidden layer detect exactly the same feature, (edges, textures) just at different locations in the input image!!

Convolutional networks are well adapted to the translation invariance of images.

Another idea is to apply, for each layer, different weights:

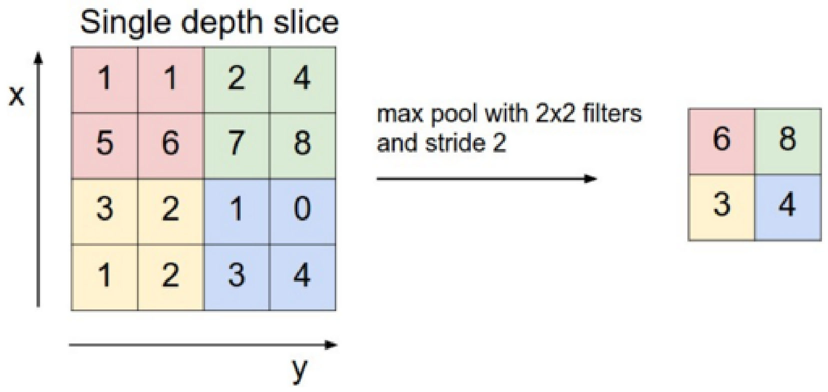


Pooling layers

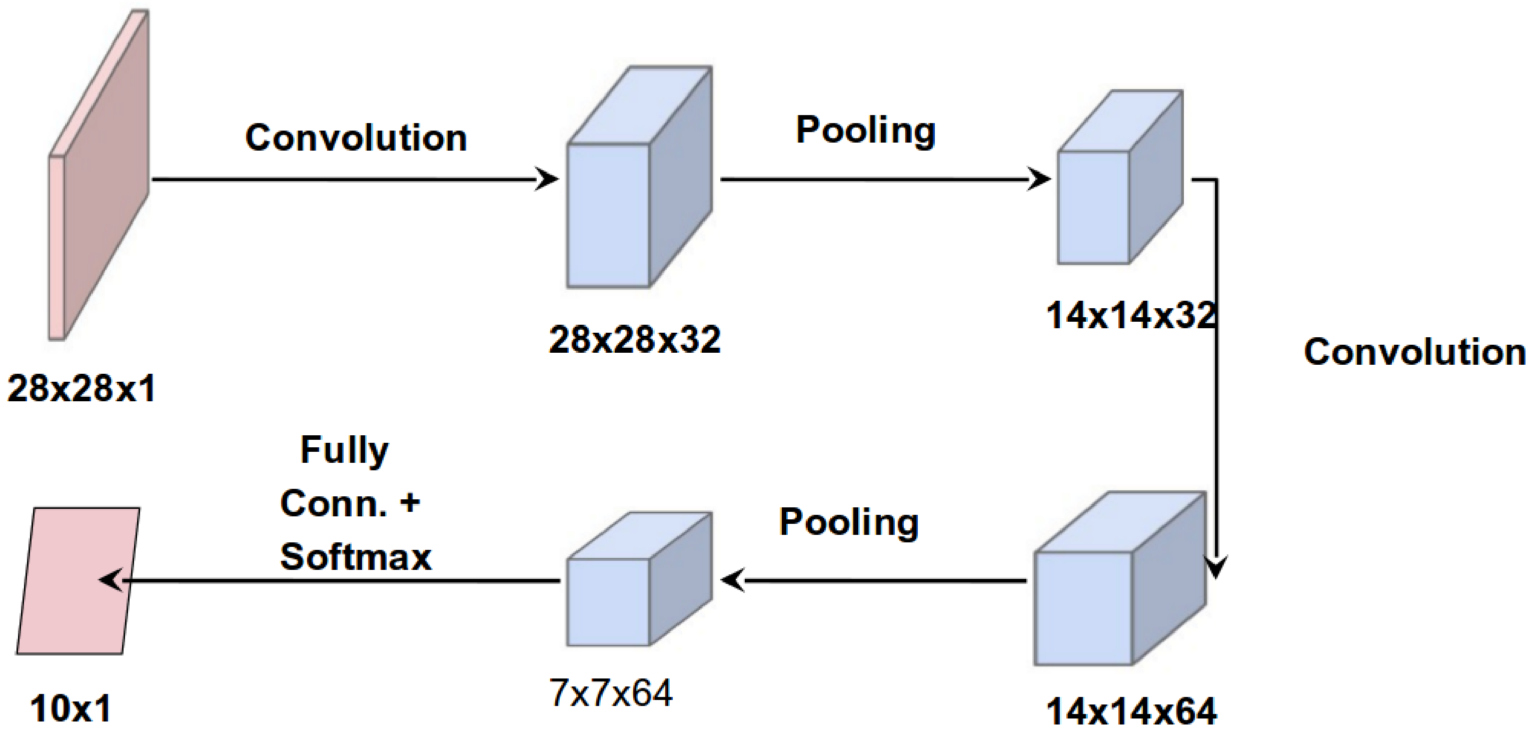
Convolutional neural networks also contain pooling layers. Pooling layers are usually used immediately after convolutional layers.

Pooling layers **simplify** the information in the output from the convolutional layer.

In max-pooling, a pooling unit simply outputs the maximum.



A typical (small) CNN

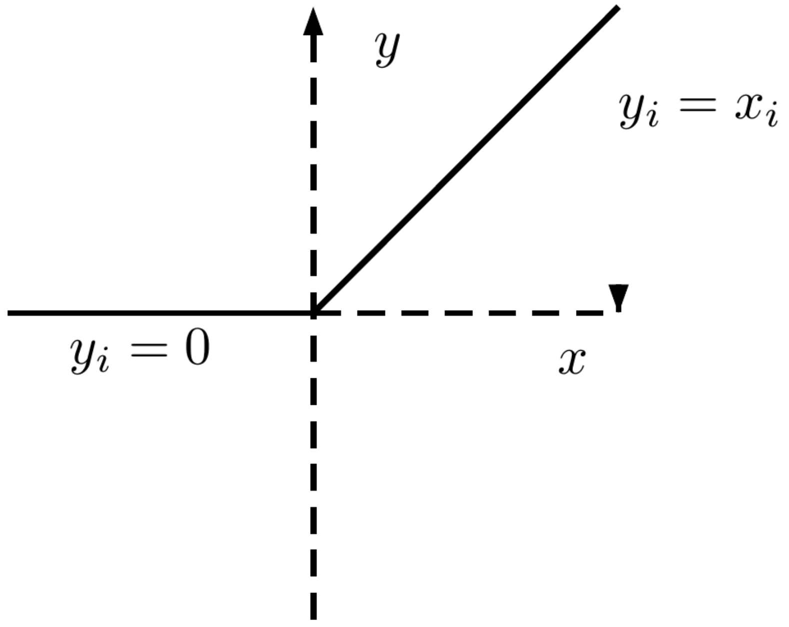


Running 20000 iteration steps, we can reach an accuracy of 99.2% in the MNIST dataset (see below).

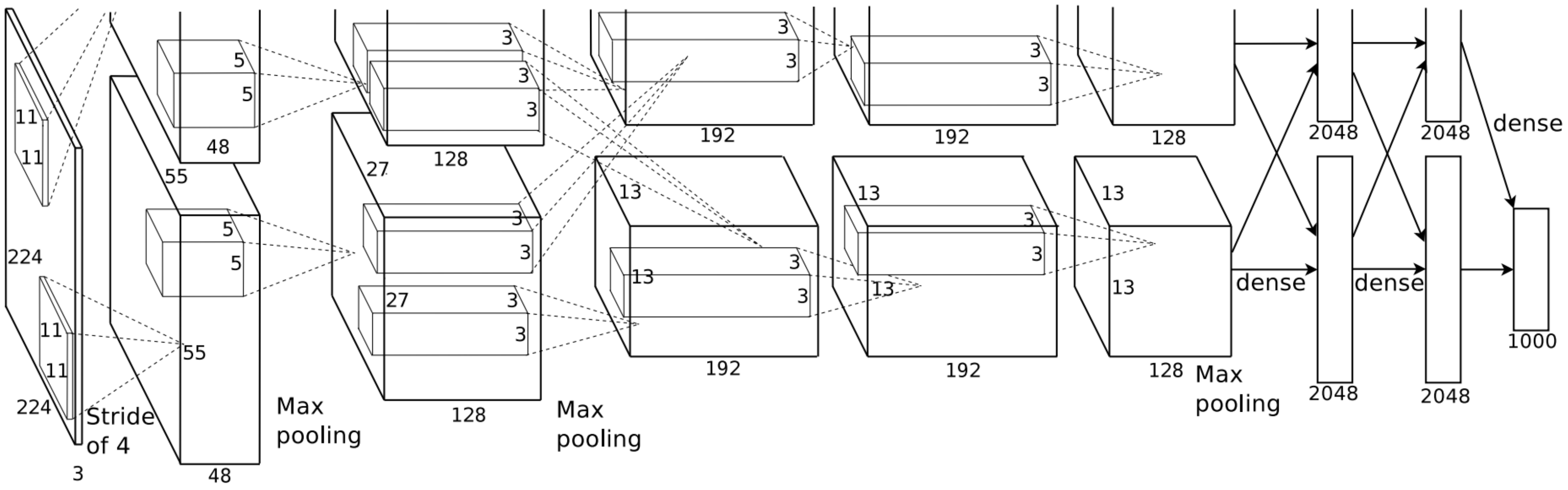
ReLU activation function

It has been shown that non-saturated activation functions such as the **rectified linear unit** (ReLU) outperforms the classical activation functions (e.g. sigmoid).

ReLU(x) = max(0,x)



The popular A. Krizhevsky's CNN



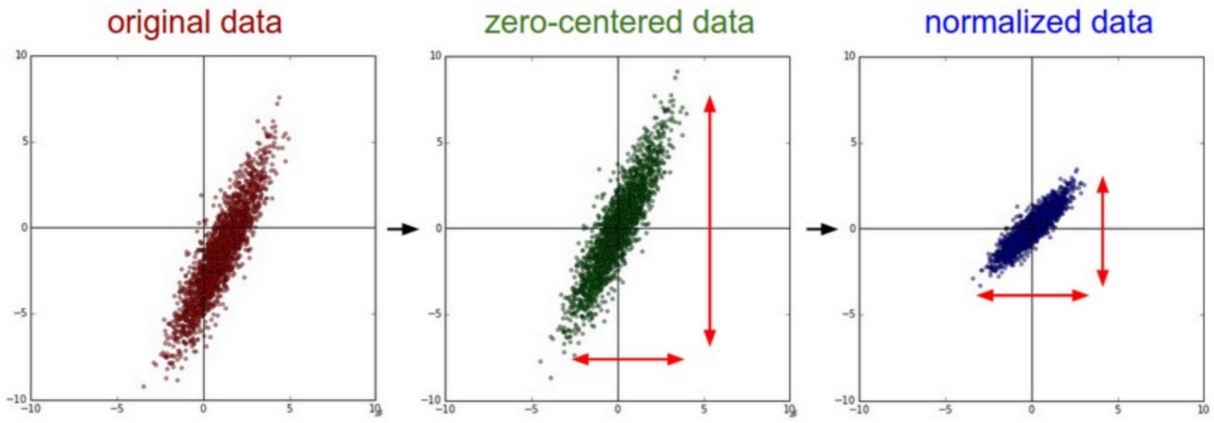
Data Preprocessing

Preprocessing helps to simplify the classification problem.

First preprocessing issue: data normalization

The aim is to remove all redundant information from the data.

Common solution is to subtract the mean (calculated only on train dataset) and normalize with respect to the covariance.



Avoid overfitting (1/3)

Several ways to prevent overfitting: **Regularization**, **Dropout** and **Data Augmentation**

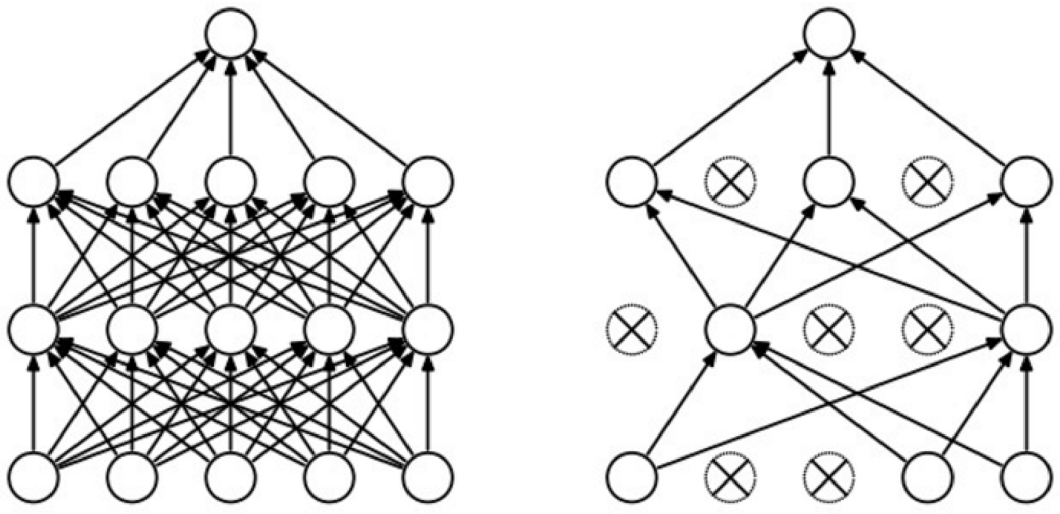
Regularization methods are used for model selection, in particular to prevent overfitting by penalizing models with extreme parameter values. Common solution are:

L2 regularization
L1 regularization
Max norm constraints

Avoid overfitting (2/3)

Dropout is implemented by only keeping a neuron active with some probability or by setting it to zero otherwise.

This affects also the back propagation, training only the activated neurons.

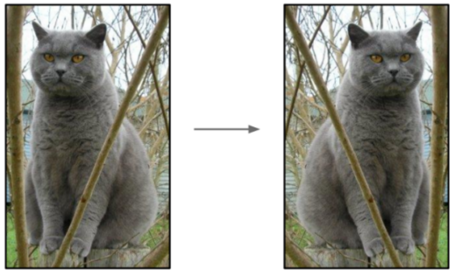


(a) Standard Neural Net (b) After applying dropout.

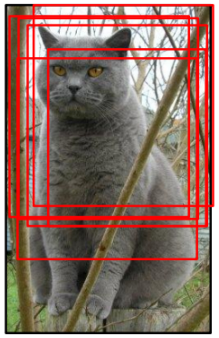
Avoid overfitting (3/3)

In **Data Augmentation**, “fake” data is simulated, encoding image transformations that shouldn’t change object identity.

Flip horizontally



Random Crop



Random mix/combinations of:

Translation

Rotation

Stretching

...

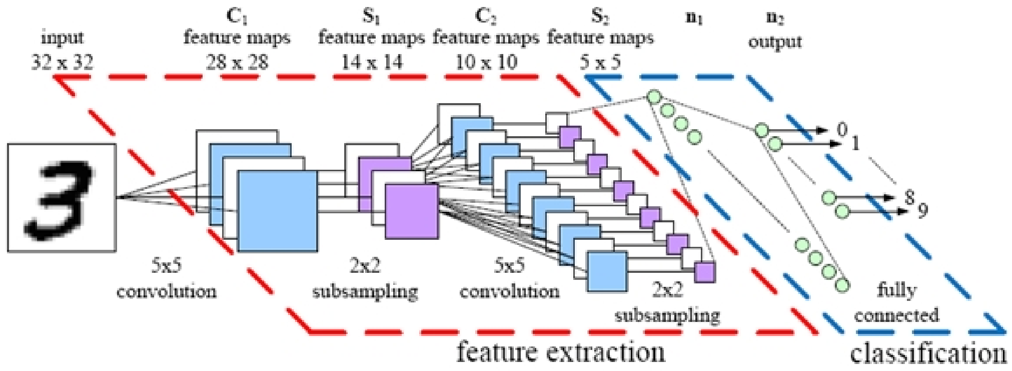
Color Jittering

A CNN for MNIST

MNIST: a subset of the NIST* database of handwritten digits

- 2 convolutional + Max pooling layers
- 2 fully connected layers

*National Institute of Standards and Technology



[Jonatan Ward *et al*. **Efficient mapping of the training of Convolutional Neural Networks to a CUDA-based cluster**]

References

[1] Michael Nielsen, **Neural Networks and Deep Learning**

- Free online version: http://neuralnetworksanddeeplearning.com/

[2] Krizhevsky, A., Sutskever, I. and Hinton, G. E. **ImageNet Classification with Deep Convolutional Neural Networks** NIPS 2012: Neural Information Processing Systems

[3] Abadi *et al.* TensorFlow: **Large-Scale Machine Learning on Heterogeneous Distributed Systems**

[4] www.tensorflow.org

Seminars in Artificial Intelligence and Robotics

Computer Vision for Intelligent Robotics

Basics and hints on CNNs

Dipartimento di Ingegneria informatica automatica e gestionale Antonio Ruberti

Sapienza Università di Roma

Alberto Pretto