RegML 2020
Class 8
Deep learning

Lorenzo Rosasco
UNIGE-MIT-IIT

Supervised vs unsupervised learning?

So far we have been thinking of learning schemes made in two steps

$$f(x) = \langle w, \Phi(x) \rangle_{\mathcal{F}}, \quad \forall x \in \mathcal{X}$$

- *unsupervised* learning of Φ
- *supervised* learning of w

Supervised vs unsupervised learning?

So far we have been thinking of learning schemes made in two steps

$$f(x) = \langle w, \Phi(x) \rangle_{\mathcal{F}}, \quad \forall x \in \mathcal{X}$$

- *unsupervised* learning of Φ
- *supervised* learning of w

But can we perform only **one learning step**?

In practice all is multi-layer!
(an old slide)

Typical data representation schemes, e.g. in vision or speech, involve **multiple stages** (*layers*).

Pipeline

Raw data are often processed:

- first computing some of **low level** features,
- then learning some **mid level** representation,
- ...
- finally using **supervised** learning.

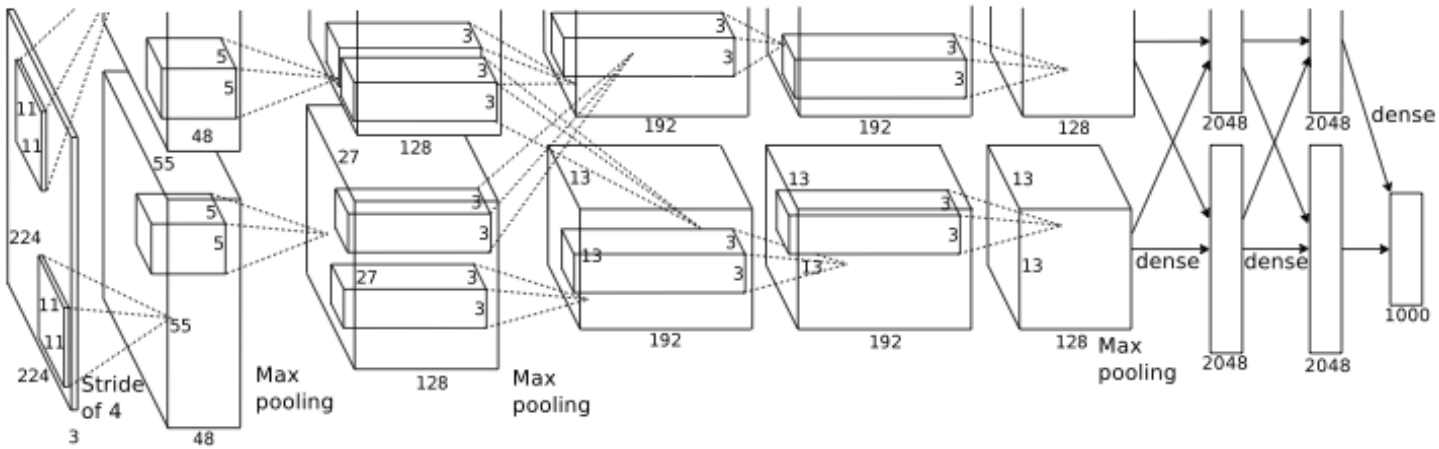
These stages are often done separately, but is it possible to design **end-to-end** learning systems?

In practice all is (becoming) deep-learning! (updated slide)

Typical data representation schemes e.g. in vision or speech, involve **deep learning**.

Pipeline

- Design some **wild- but "differentiable"** hierarchical architecture.
- Proceed with **end-to-end** learning!!



Ok, maybe not all is deep learning but let's take a look

Shallow nets

$$f(x) = \langle w, \Phi(x) \rangle \,, \quad \underbrace{x \mapsto \Phi(x)}_{\textbf{Fixed}}$$

Shallow nets

$$f(x) = \langle w, \Phi(x) \rangle \,, \quad \underbrace{x \mapsto \Phi(x)}_{\textbf{Fixed}}$$

Empirical Risk Minimization (ERM)

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle w, \Phi(x_i) \rangle)^2$$

Neural Nets

Basic idea of neural networks: functions obtained by **composition**.

$$\Phi = \Phi_L \circ \cdots \circ \Phi_2 \circ \Phi_1$$

Neural Nets

Basic idea of neural networks: functions obtained by **composition**.

$$\Phi = \Phi_L \circ \cdots \circ \Phi_2 \circ \Phi_1$$

Let $d_0 = d$ and

$$\Phi_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \ell = 1, \ldots, L$$

Neural Nets

Basic idea of neural networks: functions obtained by **composition**.

$$\Phi = \Phi_L \circ \cdots \circ \Phi_2 \circ \Phi_1$$

Let $d_0 = d$ and

$$\Phi_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \ell = 1, \ldots, L$$

and in particular

$$\Phi_\ell = \sigma \circ W_\ell, \quad \ell = 1, \ldots, L$$

Neural Nets

Basic idea of neural networks: functions obtained by **composition**.

$$\Phi = \Phi_L \circ \cdots \circ \Phi_2 \circ \Phi_1$$

Let $d_0 = d$ and

$$\Phi_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \ell = 1, \dots, L$$

and in particular

$$\Phi_\ell = \sigma \circ W_\ell, \quad \ell = 1, \dots, L$$

where

$$W_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \ell = 1, \dots, L$$

linear/affine

Neural Nets

Basic idea of neural networks: functions obtained by **composition**.

$$\Phi = \Phi_L \circ \cdots \circ \Phi_2 \circ \Phi_1$$

Let $d_0 = d$ and

$$\Phi_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \ell = 1, \dots, L$$

and in particular

$$\Phi_\ell = \sigma \circ W_\ell, \quad \ell = 1, \dots, L$$

where

$$W_\ell : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}, \quad \ell = 1, \dots, L$$

linear/affine and σ is a non linear map acting component-wise

$$\sigma : \mathbb{R} \to \mathbb{R}.$$

Deep neural nets

$$f(x) = \langle w, \Phi_L(x) \rangle \,, \qquad \underbrace{\Phi_L = \overline{\Phi}_L \circ \cdots \circ \overline{\Phi}_1}_{\text{compositional representation}}$$

$$\overline{\Phi}_1 = \sigma \circ W_1 \quad \dots \quad \overline{\Phi}_L = \sigma \circ W_L$$

Deep neural nets

$$f(x) = \langle w, \Phi_L(x) \rangle \,, \qquad \underbrace{\Phi_L = \overline{\Phi}_L \circ \cdots \circ \overline{\Phi}_1}_{\text{compositional representation}}$$

$$\overline{\Phi}_1 = \sigma \circ W_1 \quad \dots \quad \overline{\Phi}_L = \sigma \circ W_L$$

ERM

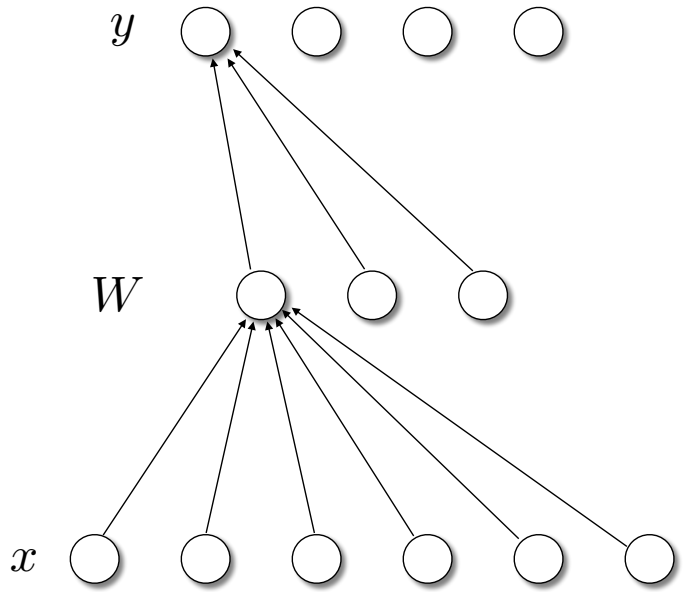
$$\min_{w,(W_j)_j} \frac{1}{n} \sum_{i=1}^{n} (y_i - \langle w, \Phi_L(x_i) \rangle)^2$$

Neural networks terminology

$$\Phi_L(x) = \sigma(W_L \dots \sigma(W_2 \sigma(W_1 x)))$$

- Each intermediate representation corresponds to a **(hidden) layer**
- The dimensionalities $(d_\ell)_\ell$ correspond to the number of **hidden units**
- the non linearity is called **activation function**

Neural networks illustrated



- Each neuron compute an **inner product** based on a column of a weight matrix W
- The non-linearity σ is the **neuron activation** function.

Activation functions

- **logistic** function $s(\alpha) = (1 + e^{-\alpha})^{-1}$, $\alpha \in \mathbb{R}$,
- **hyperbolic tangent** $s(\alpha) = (e^{\alpha} - e^{-\alpha})/(e^{\alpha} + e^{-\alpha})$, $\alpha \in \mathbb{R}$,
- **hinge** $s(\alpha) = |s|_+$, $\alpha \in \mathbb{R}$.

Note:
-If the activation is chosen to be **linear** the architecture is equivalent to **one layer**.

Neural networks function spaces

Consider the non linear space of functions of the form $f_{w,(W_\ell)_\ell} : \mathcal{X} \to \mathbb{R}$,

$$f_{w,(W_\ell)_\ell}(x) = \left\langle w, \Phi_{(W_\ell)_\ell}(x) \right\rangle, \qquad \Phi_{(W_\ell)_\ell} = \sigma(W_L \dots \sigma(W_2 \sigma(W_1 x)))$$

Very little structure, but we can :

- train by **gradient descent** (next)
- get (some) **approximation/statistical** guarantees (later)

One layer neural networks

Consider only one hidden layer:

$$f_{w,W}(x) = \langle w, \sigma(Wx) \rangle = \sum_{j=1}^{u} w_j \sigma\left(\langle W^j, x \rangle\right)$$

typically optimized given supervised data

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - f_{w,W}(x_i))^2,$$

possibly with norm constraints on the weights (*regularization*).

One layer neural networks

Consider only one hidden layer:

$$f_{w,W}(x) = \langle w, \sigma(Wx) \rangle = \sum_{j=1}^{u} w_j \sigma\left(\langle W^j, x \rangle\right)$$

typically optimized given supervised data

$$\frac{1}{n} \sum_{i=1}^{n} (y_i - f_{w,W}(x_i))^2,$$

possibly with norm constraints on the weights (*regularization*).

Problem is non-convex! (maybe possibly smooth depending on σ)

Back-propagation

Empirical risk minimization,

$$\min_{w,W} \widehat{\mathcal{E}}(w,W), \qquad \widehat{\mathcal{E}}(w,W) = \sum_{i=1}^{n} (y_i - f_{(w,W)}(x_i)))^2.$$

An approximate minimizer is computed via the following **update rules**

$$\begin{aligned} w_j^{t+1} &= w_j^t - \gamma_t \frac{\partial \widehat{\mathcal{E}}}{\partial w_j}(w^t, W^t) \\ W_{j,k}^{t+1} &= W_{j,k}^t - \gamma_t \frac{\partial \widehat{\mathcal{E}}}{\partial W_{j,k}}(w^{t+1}, W^t) \end{aligned}$$

where the step-size $(\gamma_t)_t$ is often called learning rate.

Back-propagation & chain rule

Direct computations show that:

$$\frac{\partial \widehat{\mathcal{E}}}{\partial w_j}(w, W) = -2 \sum_{i=1}^{n} \underbrace{(y_i - f_{(w,W)}(x_i))}_{\Delta_{j,i}} h_{j,i}$$

$$\frac{\partial \widehat{\mathcal{E}}}{\partial W_{j,k}}(w, W) = -2 \sum_{i=1}^{n} \underbrace{(y_i - f_{(w,W)}(x_i)) w_j \sigma'(\langle w_j, x \rangle)}_{\eta_{i,k}} x_i^k$$

Back-prop equations: $\eta_{i,k} = \Delta_{j,i} c_j s'(\langle w_j, x \rangle)$

Using above equations, the updates are performed in two steps:

- **Forward pass** compute function values keeping weights fixed,
- **Backward pass** compute errors and propagate
- Hence the weights are updated.

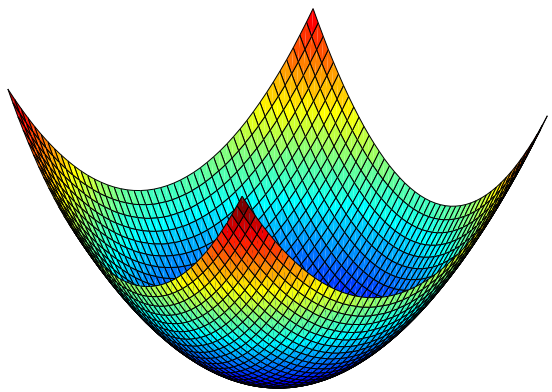
Few remarks

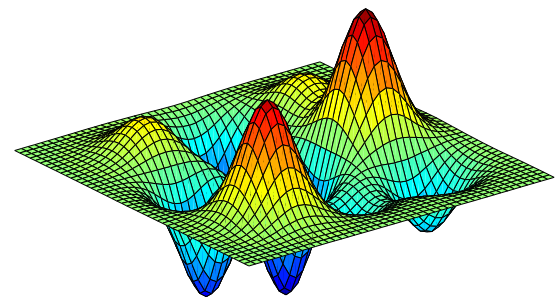
- **Multiple layers** can be analogously considered
- **Batch** gradients descent can be replaced by **stochastic** gradient.
- **Faster** iterations are available, e.g. *variable metric/accelerated gradient*. . . .
- **Online** update rules are potentially biologically plausible– **Hebbian learning** rules describing neuron **plasticity**.

Computations

$$\min_{w,W} \widehat{\mathcal{E}}(w, W), \qquad \widehat{\mathcal{E}}(w, W) = \sum_{i=1}^{n} (y_i - f_{(w,W)}(x_i)))^2.$$

In practice, no access to $\widehat{f}_u$ but only to approximate minimizers.





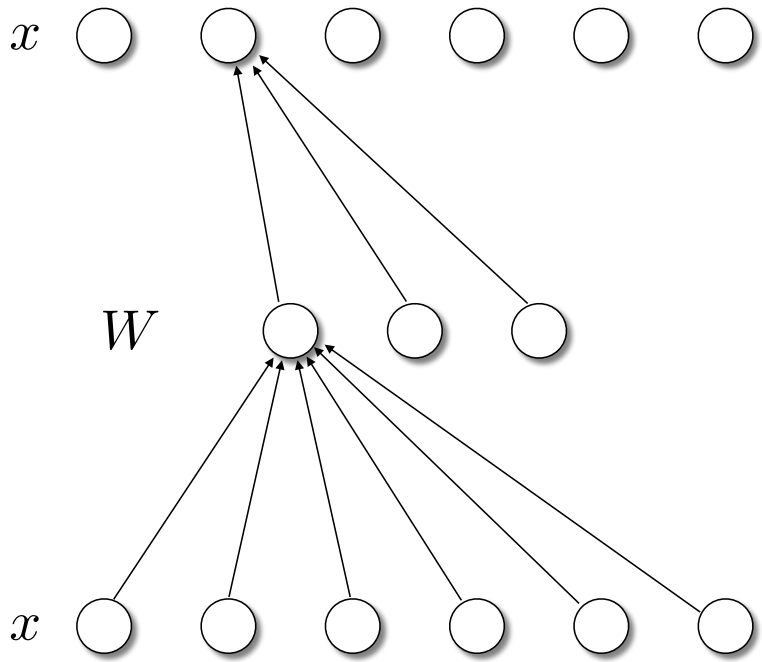
- **Non-convex** problem
- Convergence of back-prop to a reasonable local minimum can depend heavily on the **initialization**.
- Empirically: the more the layers the easier to find good minima.

An older idea: pre-training and unsupervised learning

Pre-training

- Use unsupervised training of each layer to **initialize** supervised training.
- Potential **benefit** of unlabeled data.

Auto-encoders



- A neural network with **one input layer, one output layer and one (or more) hidden layers** connecting them.
- The output layer has **equally** many nodes as the input layer,
- It is trained to **predict the input** rather than some target output.

Auto-encoders (cont.)

An auto encoder with one hidden layer of k units, can be seen as a **representation-reconstruction** pair:

$$\Phi : \mathcal{X} \to \mathcal{F}_k, \quad \Phi(x) = \sigma\left(Wx\right), \quad \forall x \in \mathcal{X}$$

with $\mathcal{F}_k = \mathbb{R}^k$, $k < d$ and

$$\Psi : \mathcal{F}_k \to \mathcal{X}, \quad \Psi(\beta) = \sigma\left(W'\beta\right), \quad \forall \beta \in \mathcal{F}_k.$$

Auto-encoders & dictionary learning

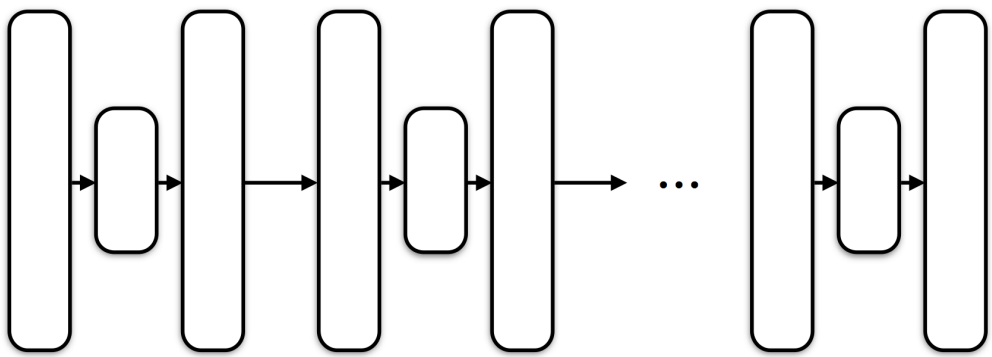
$$\Phi(x) = \sigma\left(Wx\right), \qquad \Psi(\beta) = \sigma\left(W'\beta\right)$$

- The above formulation is closely related to **dictionary learning**.
- The weights can be seen as dictionary **atoms**.
- Reconstructive approaches have connections with so called **energy models** [LeCun et al.. . .]
- Possible **probabilistic/Bayesian** interpretations/variations (e.g. Boltzmann machine [Hinton et al.. . .])

Stacked auto-encoders

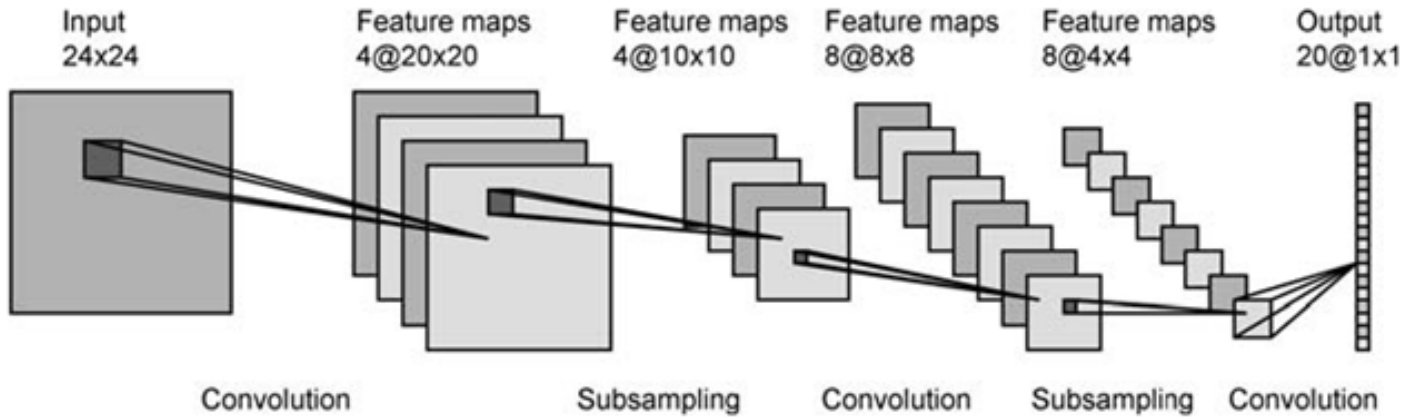
Multiple layers of auto-encoders can be **stacked** [Hinton et al '06]...

$$\underbrace{(\Phi_1 \circ \Psi_1)}_{\text{Autoencoder}} \circ (\Phi_2 \circ \Psi_2) \cdots \circ (\Phi_\ell \circ \Psi_\ell)$$



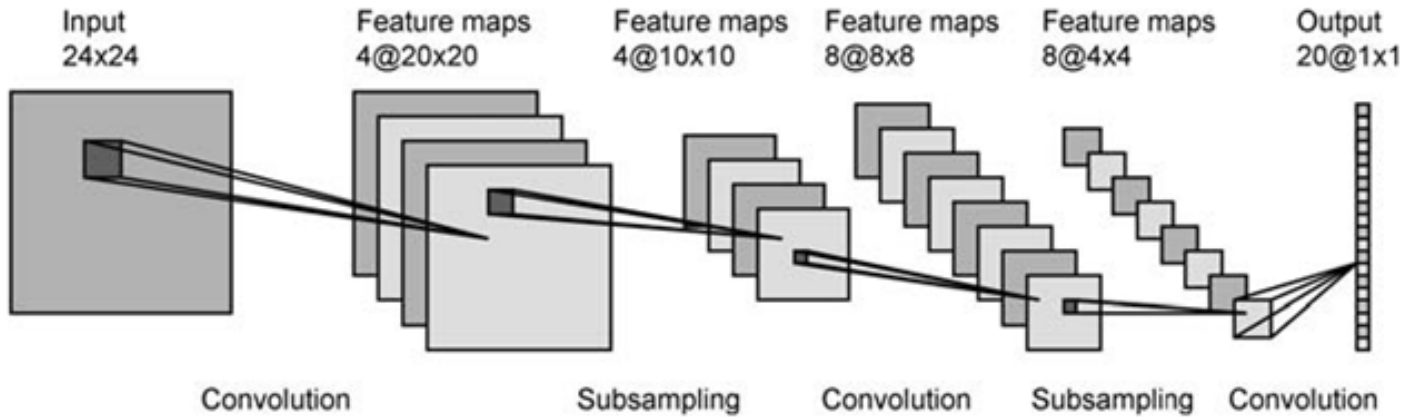
...with the potential of obtaining **richer** representations.

Beyond reconstruction



In many applications the **connectivity** of neural networks is limited in a specific way.

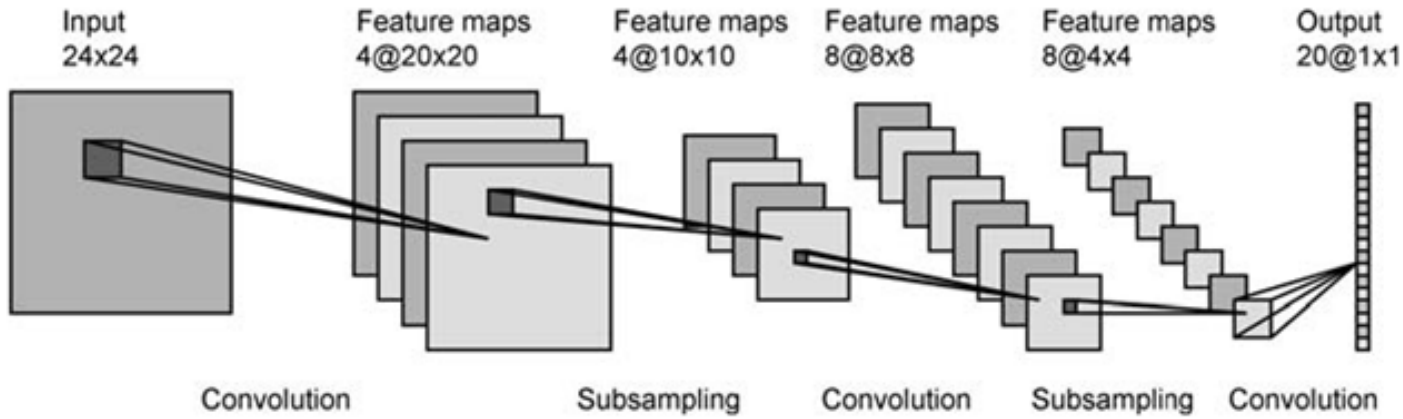
Beyond reconstruction



In many applications the **connectivity** of neural networks is limited in a specific way.

- Weights in the first few layers have **smaller support** and are **repeated**.

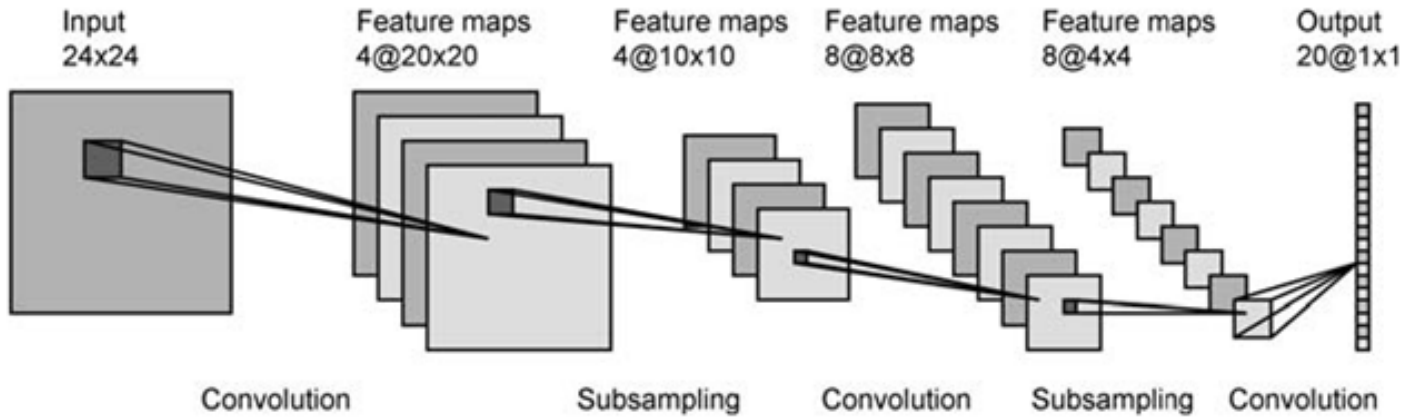
Beyond reconstruction



In many applications the **connectivity** of neural networks is limited in a specific way.

- Weights in the first few layers have **smaller support** and are **repeated**.
- **Subsampling** (*pooling*) is interleaved with standard neural nets computations.

Beyond reconstruction



In many applications the **connectivity** of neural networks is limited in a specific way.

- Weights in the first few layers have **smaller support** and are **repeated**.
- **Subsampling** (*pooling*) is interleaved with standard neural nets computations.

The obtained architectures are called **convolutional neural networks**.

Convolutional layers

Consider the composite representation

$$\Phi : \mathcal{X} \to \mathcal{F}, \quad \Phi = \sigma \circ W,$$

with

- representation by **filtering** $W : \mathcal{X} \to \mathcal{F}'$,
- representation by **pooling** $\sigma : \mathcal{F}' \to \mathcal{F}$.

Convolutional layers

Consider the composite representation

$$\Phi : \mathcal{X} \to \mathcal{F}, \quad \Phi = \sigma \circ W,$$

with

- representation by **filtering** $W : \mathcal{X} \to \mathcal{F}'$,
- representation by **pooling** $\sigma : \mathcal{F}' \to \mathcal{F}$.

Note: σ, W are more complex than in standard NN.

Convolution and filtering

The matrix W is made of blocks

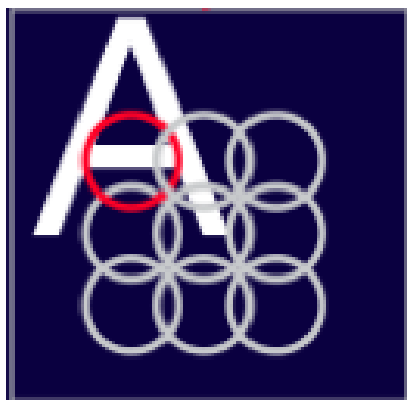
$$W = (G_{t_1}, \dots, G_{t_T})$$

each block is a *convolution matrix* obtained transforming a vector (template) t, e.g.

$$G_t = (g_1 t, \dots, g_N t).$$

e.g.

$$G_t = \begin{bmatrix} t^1 & t_2 & t_3 & \dots & t^d \\ t^d & t^1 & t_2 & \dots & t^{d-1} \\ t^{d-1} & t^d & t^1 & \dots & t^{d-2} \\ \dots & \dots & \dots & \dots & \dots \\ t^2 & t^3 & t^4 & \dots & t^1 \end{bmatrix}$$



For all $x \in \mathcal{X}$,

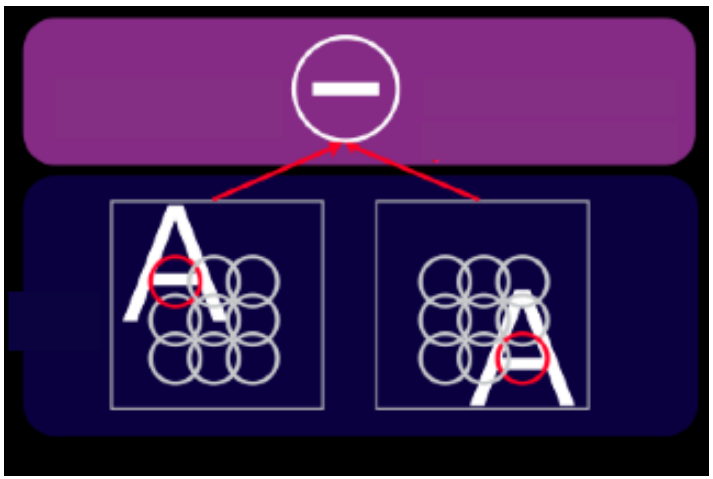
$$W(x)(j,i) = \langle g_i t_j, x \rangle .$$

Pooling

The **pooling** map **aggregates** (pools) the values corresponding to the same transformed template

$$\langle g_1 t, x \rangle, \ldots, (\langle g_N t, x \rangle,$$

and can be seen as a form of **subsampling**.



Pooling functions

Given a template t, let

$$\beta = (s(\langle g_1 t, x \rangle), \dots, s(\langle g_N t, x \rangle)) \,.$$

for some non-linearity s, e.g. $s(\cdot) = |\cdot|_+$.

Pooling functions

Given a template t, let

$$\beta = (s(\langle g_1 t, x\rangle), \dots, s(\langle g_N t, x\rangle)) \,.$$

for some non-linearity s, e.g. $s(\cdot) = |\cdot|_+$.

Examples of pooling

- max pooling

$$\max_{j=1,\dots,N} \beta^j,$$

- average pooling

$$\frac{1}{N}\sum_{j=1}^{N} \beta^j,$$

- ℓ_p pooling

$$\|\beta\|_p = \left(\sum_{j=1}^{N} |\beta^j|^p\right)^{\frac{1}{p}}.$$

Why pooling?

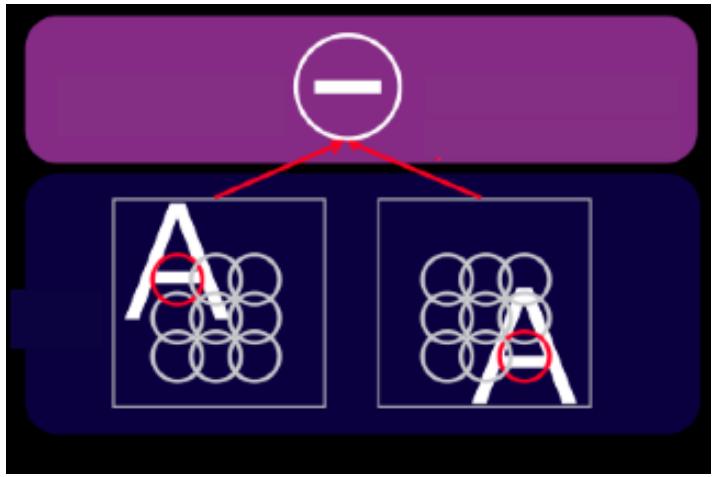
The intuition is that pooling can provide some form of robustness and even **invariance** to the transformations.

Invariance & selectivity

- A good representation should be **invariant** to **semantically irrelevant** transformations.
- Yet, it should be **discriminative** with respect to **relevant** information (**selective**).

Basic computations: simple & complex cells

(Hubel, Wiesel '62)



- Simple cells

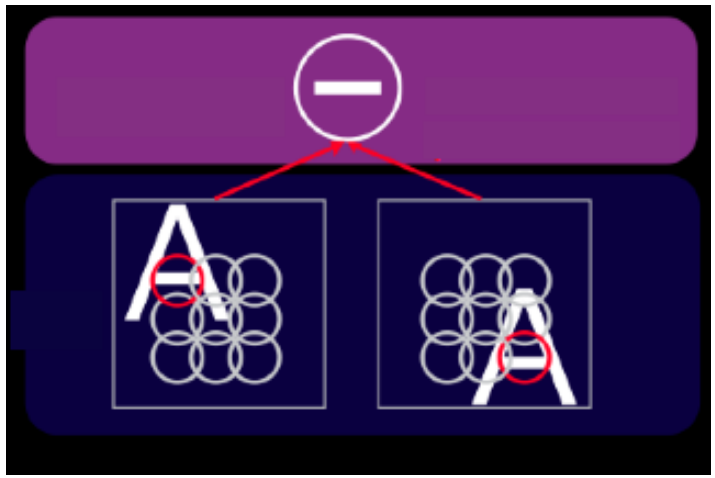
$$x \mapsto \langle x, g_1 t \rangle, \ldots, \langle x, g_N t \rangle$$

- Complex cells

$$\langle x, g_1 t \rangle, \ldots, \langle x, g_N t \rangle \ldots, \langle x, g_N t \rangle \mapsto \sum_g | \langle x, gt \rangle |_+$$

Basic computations: convolutional networks

(Le Cun '88)



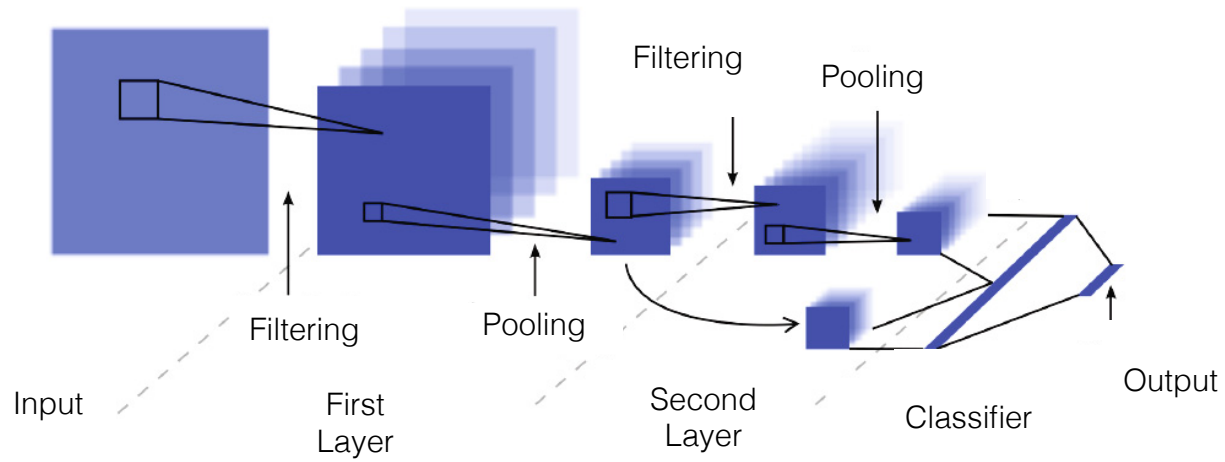
- Convolutional filters

$$x \mapsto \langle x, g_1 t \rangle, \ldots, \langle x, g_N t \rangle$$

- Subsampling/pooling

$$\langle x, g_1 t \rangle, \ldots, \langle x, g_N t \rangle \ldots, \langle x, g_N t \rangle \mapsto \sum_g | \langle x, gt \rangle |_+$$

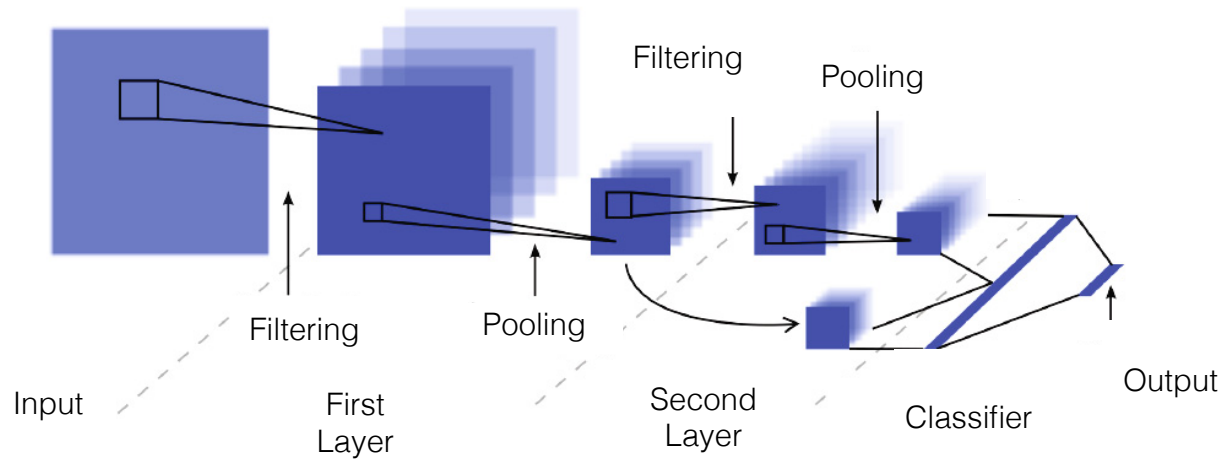
Deep convolutional networks



In practice:

- multiple convolution layers are **stacked**,

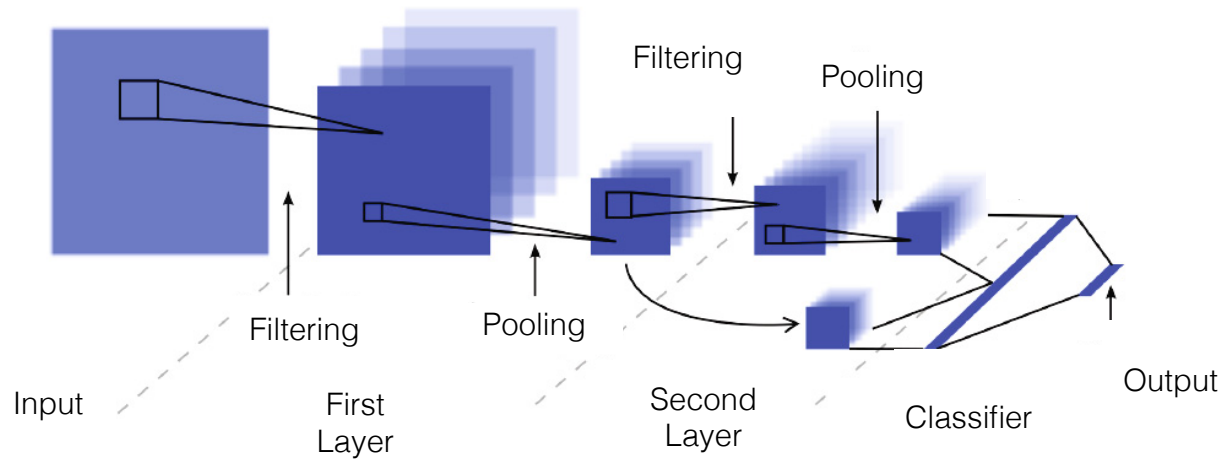
Deep convolutional networks



In practice:

- multiple convolution layers are **stacked**,
- pooling is not global, but over a subset of transformations (**receptive field**),

Deep convolutional networks



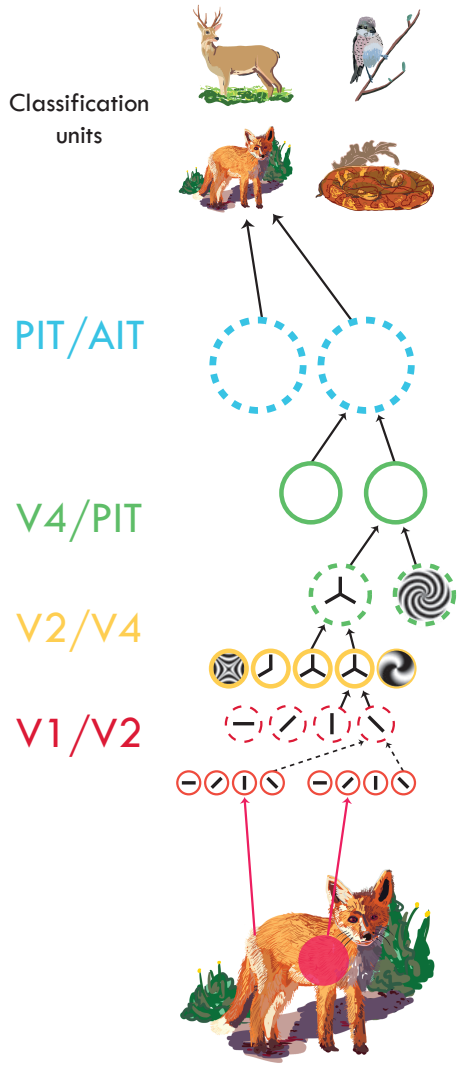
In practice:

- multiple convolution layers are **stacked**,
- pooling is not global, but over a subset of transformations (**receptive field**),
- the receptive fields size increases in **higher layers**.

A biological motivation

Visual cortex

The processing in DCN has analogies with computational neuroscience models of the **information processing in the visual cortex see [Poggio et al. . . .]**.



Theory

$$\Phi_L(x) = \sigma(W_L \dots \sigma(W_2(\sigma(W_1 x)))$$

- **No pooling**: metric properties of networks with random weights – connection with compressed sensing [Giryes et al. '15]

Theory

$$\Phi_L(x) = \sigma(W_L \dots \sigma(W_2(\sigma(W_1 x)))$$

- **No pooling**: metric properties of networks with random weights – connection with compressed sensing [Giryes et al. '15]
- **Invariance**

$$x' = gx \Rightarrow \Phi(x') = \Phi(x)$$

[Anselmi et al. '12, R. Poggio '15, Mallat '12, Soatto, Chiuso '13] and covariance for multiple layers [Anselmi et al. '12].

Theory

$$\Phi_L(x) = \sigma(W_L \dots \sigma(W_2(\sigma(W_1 x)))$$

- **No pooling**: metric properties of networks with random weights – connection with compressed sensing [Giryes et al. '15]
- **Invariance**

$$x' = gx \Rightarrow \Phi(x') = \Phi(x)$$

[Anselmi et al. '12, R. Poggio '15, Mallat '12, Soatto, Chiuso '13] and covariance for multiple layers [Anselmi et al. '12].

- **Selectivity/Maximal Invariance**, i.e. injectivity modulo transformations

$$\Phi(x') = \Phi(x) \Rightarrow x' = gx$$

[R. Poggio '15, Soatto, Chiuso '15]

Theory (cont.)

- **Similarity** preservation

$$\|\Phi(x') - \Phi(x)\| \asymp \min_g \|x' - gx\| ???$$

- Stability to **diffeomorphisms** [Mallat, '12]

$$\|\Phi(x) - \Phi(d(x))\| \lesssim \|d\|_\infty \|x\|$$

- **Reconstruction**: connection to phase retrieval/one bit compressed sensing [Bruna et al '14].

This class

- Neural nets
- Autoencoders
- Convolutional neural nets

FINE

Looking for postdocs, mail me if interested!!!