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

\[ f(x) = \langle w, \Phi(x) \rangle_F, \quad \forall x \in \mathcal{X} \]

- **unsupervised** learning of \( \Phi \)
- **supervised** learning of \( w \)
Supervised vs unsupervised learning?

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

\[ f(x) = \langle w, \Phi(x) \rangle_F, \quad \forall x \in \mathcal{X} \]

- unsupervised learning of \( \Phi \)
- 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

L. Rosasco, RegML 2016
Shallow nets

\[ f(x) = \langle w, \Phi(x) \rangle, \quad x \mapsto \Phi(x) \]

Fixed

L.Rosasco, RegML 2016
Shallow nets

\[ f(x) = \langle w, \Phi(x) \rangle, \quad x \mapsto \Phi(x) \]

Empirical Risk Minimization (ERM)

\[
\min_w \frac{1}{n} \sum_{i=1}^{n} \left( y_i - \langle w, \Phi(x_i) \rangle \right)^2
\]
Neural Nets

Basic idea of neural networks: functions obtained by \textit{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

\[ \Phi_\ell : \mathbb{R}^{d_{\ell-1}} \rightarrow \mathbb{R}^{d_\ell}, \quad \ell = 1, \ldots, L \]

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

\[ \Phi_\ell : \mathbb{R}^{d_{\ell-1}} \rightarrow \mathbb{R}^{d_\ell}, \quad \ell = 1, \ldots, L \]

and in particular

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

where

\[ W_\ell : \mathbb{R}^{d_{\ell-1}} \rightarrow \mathbb{R}^{d_\ell}, \quad \ell = 1, \ldots, L \]

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

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

and in particular

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

where

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

linear/affine and $\sigma$ is a non linear map acting component-wise

$$\sigma : \mathbb{R} \rightarrow \mathbb{R}.$$
Deep neural nets

\[ f(x) = \langle w, \Phi_L(x) \rangle, \quad \Phi_L = \overline{\Phi}_L \circ \cdots \circ \overline{\Phi}_1 \]

compositional representation

\[ \overline{\Phi}_1 = \sigma \circ W_1 \quad \cdots \quad \overline{\Phi}_L = \sigma \circ W_L \]
Deep neural nets

\[ f(x) = \langle w, \Phi_L(x) \rangle, \quad \Phi_L = \Phi_L \circ \cdots \circ \Phi_1 \]

compositional representation

\[
\Phi_1 = \sigma \circ W_1 \quad \ldots \quad \Phi_L = \sigma \circ W_L
\]

ERM

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

L. Rosasco, RegML 2016
Neural networks terminology

\[ \Phi_L(x) = \sigma(W_L \ldots \sigma(W_2\sigma(W_1x))) \]

- 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
Each neuron compute an **inner product** based on a column of a weight matrix $W$

The non-linearity $\sigma$ 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 \( f_{w,(W_\ell)_{\ell}} : \mathcal{X} \to \mathbb{R} \),

\[ f_{w,(W_\ell)_{\ell}}(x) = \langle w, \Phi_{(W_\ell)_{\ell}}(x) \rangle, \quad \Phi_{(W_\ell)_{\ell}} = \sigma(W_L \ldots \sigma(W_2 \sigma(W_1 x))) \]

Very little structure, but we can:

- train by \textit{gradient descent} (next)
- get (some) \textit{approximation/statistical} guarantees (later)
One layer neural networks

Consider only one hidden layer:

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

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:

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

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 \( \sigma \))
Empirical risk minimization,

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

An approximate minimizer is computed via the following update rules

$$w_{j}^{t+1} = w_{j}^{t} - \gamma_{t} \frac{\partial \hat{E}}{\partial w_{j}^{t}}(w^{t}, W^{t})$$

$$W_{j,k}^{t+1} = W_{j,k}^{t} - \gamma_{t} \frac{\partial \hat{E}}{\partial W_{j,k}^{t}}(w^{t+1}, W^{t})$$

where the step-size ($\gamma_{t}$) is often called learning rate.
Back-propagation & chain rule

Direct computations show that:

\[
\frac{\partial \hat{E}}{\partial w_j}(w, W) = -2 \sum_{i=1}^{n} (y_i - f_{(w,W)}(x_i)) h_{j,i} \\
\frac{\partial \hat{E}}{\partial W_{j,k}}(w, W) = -2 \sum_{i=1}^{n} (y_i - f_{(w,W)}(x_i)) w_j \sigma'(\langle w_j, x \rangle) x_i^k
\]

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**.
\[
\min_{w,W} \hat{\mathcal{E}}(w, W), \quad \hat{\mathcal{E}}(w, W) = \sum_{i=1}^{n} (y_i - f_{w,W}(x_i))^2.
\]

In practice, no access to \(\hat{f}_u\) 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.

L. Rosasco, RegML 2016
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.
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.
An auto encoder with one hidden layer of $k$ units, can be seen as a representation-reconstruction pair:

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

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

$$\Psi : \mathcal{F}_k \rightarrow \mathcal{X}, \quad \Psi(\beta) = \sigma(W'\beta), \quad \forall \beta \in \mathcal{F}_k.$$
Auto-encoders & dictionary learning

\[ \Phi(x) = \sigma(Wx), \quad \Psi(\beta) = \sigma(W'\beta) \]

- 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]...

\[
(\Phi_1 \circ \Psi_1) \circ (\Phi_2 \circ \Psi_2) \cdots \circ (\Phi_\ell \circ \Psi_\ell)
\]

...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**.
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** \( \sigma : \mathcal{F}' \to \mathcal{F} \).

Note: \( \sigma, W \) are more complex than in standard NN.

L. Rosasco, RegML 2016
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: \( \sigma, W \) are more complex than in standard NN.
The matrix $W$ is made of blocks

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

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

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

e.g.

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

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

\[
\beta = (s(\langle g_1 t, x \rangle), \ldots, s(\langle g_N t, x \rangle))
\]

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

Given a template $t$, let

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

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

Examples of pooling

- **max pooling**
  $$\max_{j=1,...,N} \beta_j,$$

- **average pooling**
  $$\frac{1}{N} \sum_{j=1}^{N} \beta_j,$$

- **$\ell_p$ 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

\[ \langle x, g_1 t \rangle, \ldots, \langle x, g_N t \rangle, \ldots, \langle x, g_N t \rangle \mapsto \sum_g |\langle x, g t \rangle|_+ \]

L.Rosasco, RegML 2016
The role of layer 1 is to decorrelate variables and accentuate the differences (or ratios) between them, while eliminating variations of the absolute energy so that the non-linearity of layers 3 can always operate at its sweet spot. Decorrelation (and mean removal) has the additional advantage of accelerating gradient-based learning [8].

Layer 2 and 3 detect conjunctions of features or motifs on the previous stage. Its role is to non-linearly embed the input into a higher-dimensional space, so that inputs that are semantically different are likely to be represented by different patterns of activity. This expansion plays a similar role as using a non-linear kernel function in a kernel machine: in high-dimensional spaces, categories are easier to separate. More generally, a function interestingly likely to belong in the mathematical space where a variable is embedded in a high-dimensional space. The difference with kernel machine is that our filter banks will be trained from data, rather than simply selected from the training set.

Layer 4 serves to merge semantically similar things that have been partitioned into different patterns of activity by the simple cells. This is where invariance is built. Rather than producing invariance in the mathematical sense, the pooling layer merely "smoothes out" the input-output mapping so that irrelevant variations in the input affect the output smoothly, and in ways that can be easily dealt with (eliminated, if necessary).

The pooling operation can consist of any symmetric aggregation function, such as an average, a max, a log-mixture \( \log \left( \sum_i e^{x_i} \right) \), or an \( L_p \) norm \( \left( \sqrt[\text{p}]{\sum_i |x_i|^\text{p}} \right) \), particularly with \( p = 1, 2, \) or \( \infty \) (max). A theoretical analysis of pooling operations suggests that \( L_\infty \) is best when the features are sparse and the number of pooled variables is small, while average, \( L_1 \) or \( L_2 \) are best when the features are less sparse or the pooling area is large [9]. In practice \( L_2 \) pooling is a good tradeoff.

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),

![Diagram of a deep convolutional network](image)

Filtering
First Layer
Pooling
Second Layer
Classifier

L.Rosasco, RegML 2016
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

\[ \Phi_L(x) = \sigma(W_L \ldots \sigma(W_2(\sigma(W_1 x)))) \]

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

- Selectivity/Maximal Invariance, i.e. injectivity modulo transformations \[ \Phi(x') = \Phi(x) \Rightarrow x' = g(x) \] [R. Poggio ’15, Soatto, Chiuso ’15]
Theory

\[ \Phi_L(x) = \sigma(W_L \ldots \sigma(W_2(\sigma(W_1 x)))) \]

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

\[ x' = g x \Rightarrow \Phi(x') = \Phi(x) \]

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

\[ \Phi_L(x) = \sigma(W_L \ldots \sigma(W_2(\sigma(W_1x)))) \]

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

- **Invariance**

  \[ x' = gx \Rightarrow \Phi(x') = \Phi(x) \]


- **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

  \[ \| \Phi(x') - \Phi(x) \| \lesssim \min_g \| x' - gx \| \]

- **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