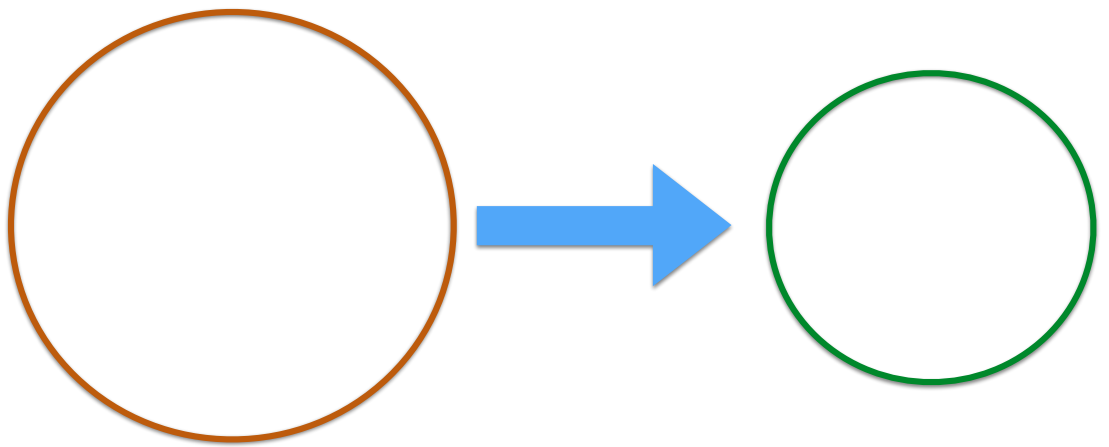
MIT 9.520/6.860
Statistical Learning Theory and Applications

Class 11: Neural Networks (aka Deep Learning)

Lorenzo Rosasco

Learning data representation



$$\Phi : \mathcal{F} \to X$$

Reconstruction based learning of data representation

$$\min_{\Psi,\Phi} \frac{1}{n} \sum_{i=1}^{n} \|x_i - \Psi \circ \Phi(x_i)\|^2 ,$$

- Ψ reconstruction,
- (Ψ, Φ) autoencoder.

Dictionary learning

$$\underbrace{\min_{\Psi \in \mathcal{D}}}_{\text{Dictionary learning}} \quad \frac{1}{n} \sum_{i=1}^{n} \underbrace{\min_{\beta_i \in \mathcal{F}_\lambda} \|x_i - \Psi \beta_i\|^2}_{\text{Representation learning}},$$

- Ψ linear,
- Φ nearest neighbor encoding,

$$\Phi(x) = \arg\min_{\beta \in \mathcal{F}_\lambda \subseteq \mathcal{F}} \|x - \Phi\beta\|^2 .$$

Beyond reconstruction: supervised data representation learning

Given $(x_1, y_1), \ldots, (x_n, y_n)$ consider

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

We need to parameterize Φ.

Linear representations

$$\Phi(x) = Wx, \qquad W \in \mathbb{R}^{pd}.$$

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

- The above problem is non convex.
- We are considering only linear functions,

$$f(x) = w^\top W x = \beta^\top x, \qquad \beta = W^\top w.$$

Beyond linear representations

$$f(x) = w^\top Wx.$$

- Insert non linearity $\varphi : \mathbb{R}^d \to\to \mathbb{R}^p$ in front of x,

$$f(x) = w^\top W\varphi(x), \quad \mapsto \text{back to kernels.}$$

- Insert non linearity $\sigma : \mathbb{R}^d \to\to \mathbb{R}^p$ in front of W,

$$f(x) = w^\top \sigma(Wx), \quad \mapsto \text{neural nets.}$$

Beyond linear representations

$$\Phi(x) = \sigma(Wx), \qquad \sigma : \mathbb{R}^p \to \mathbb{R}^p, \quad \text{non-linear.}$$

$$\min_{w,W} \frac{1}{n} \sum_{i=1}^{n} (y_i - w^\top \sigma(Wx_i))^2.$$

- The above problem is non convex, possibly non differentiable.
- We are considering non linear functions,

$$f(x) = w^\top \sigma(Wx).$$

Meet a neural net

Assume σ acts component wise, i.e.

$$\sigma(Wx) = (\sigma(x^\top W^1), \ldots, \sigma(x^\top W^p))$$

where $\sigma : \mathbb{R} \to \mathbb{R}$, and $W^1, \ldots, W^p$ are the rows of W.

Then,

$$f(x) = w^\top \sigma(Wx) = \sum_{j=1}^{p} w^j \sigma(x^\top W^j).$$

Examples of nonlinearities

$$f(x) = \sum_{j=1}^{p} w^j \sigma(x^\top W^j).$$

The classical and most used nonlinearities are respectively,

- Logistic $\sigma(a) = (1 + e^{-a})$,
- ReLU $\sigma(a) = |a|_+ = \max\{a, 0\}$.

Next

$$f(x) = w^\top \sigma(Wx) = \sum_{j=1}^{p} w^j \sigma(x^\top W^j).$$

- We compare with other function models we have seen.
- We make sense of the name.
- We go deep and convolutional.

Comparison with other functions

So far:

- Linear
$$f(x) = w^\top x.$$

- Features
$$f(x) = w^\top \Phi(x) = \sum_{j=1}^{p} w^j \varphi_j(x).$$

- Kernels
$$f(x) = \sum_{i=1}^{n} K(x, x_i)\, c_i.$$

Random feature maps

An example of feature map:

Random features

$\Phi(x) = \sigma(Wx) = (\sigma(x^\top W^1), \ldots, \sigma(x^\top W^p)),$

$$f(x) = w^\top \sigma(Wx) = \sum_{j=1}^{p} w^j \sigma(x^\top W^j),$$

where,

- $\sigma : \mathbb{R} \to \mathbb{R}$ can be nonlinear,
- $(W^1, \ldots, W^p) = W$ random.

Neural networks vs (random) features

$$f(x) = w^\top \sigma(Wx) = \sum_{j=1}^{p} w^j \sigma(x^\top W^j),$$

where,

- $\sigma : \mathbb{R} \to \mathbb{R}$ can be nonlinear.
- $(W^1, \ldots, W^p) = W$ free parameters to be estimated.
- Offsets are typically added $\sigma(x^\top W^j + b_j)$.

Weights are optimized instead of being chosen randomly.

RBF neural networks

Other forms of neural networks can be considered.

For example radial basis functions (RBF) networks,

$$f(x) = \sum_{j=1}^{M} w^j \sigma(\| W^j - x \|),$$

e.g. $\sigma(\| W^j - x \|) = e^{-\| W^j - x \|^2 \gamma}$.

RBF neural networks and kernels

$$f(x) = \sum_{j=1}^{M} w^j \sigma(\| W^j - x \|).$$

- Weights (also called centers) are training set points.
- Optimal choice (no more than this are needed).
- Neural nets: any non linearity, any W.

Neural nets vs features/kernels

$$f(x) = \sum_{j=1}^{M} w^j \sigma(x^\top W^j), \qquad f(x) = \sum_{j=1}^{M} w^j \sigma(\left\| W^j - x \right\|).$$

- Random features: any non linearity, random W.
- Kernels: pos. def. non linearity, W is the training set.
- Neural nets: any non linearity, any W.

Why are they called neural nets?

The name stems from a biological interpretation.

A Neuron

$$z^i(x) = \sigma(x^\top W^i) = \sigma(\sum_{j=1}^{d} x^j W^{ij}).$$

- For an input $x \in R^d$, the output is $z^i(x) \in \mathbb{R}$.
- A neuron z^i computes an inner product based on a vector weight matrix W^i.
- The non-linearity σ is the neuron activation function.

Network of neurons

A neuron can be used to combine neurons.

The outputs of p neurons,

$$z^i(x) = \sigma(x^\top W^i) = \sigma(\sum_{j=1}^{d} x^j W^{ij}), \qquad i = 1, \ldots, p$$

are given as inputs to another neuron f with associate weight w,

$$f(x) = \sigma(w^\top z(x))$$

where $z(x) = (z^1(x), \ldots, z^p(x))$.

Then,

$$f(x) = \sigma(w^\top z(x)) = \sigma(\sum_{j=1}^{p} w^j \sigma(x^\top W^j).$$

The last nonlinearity was omitted before.

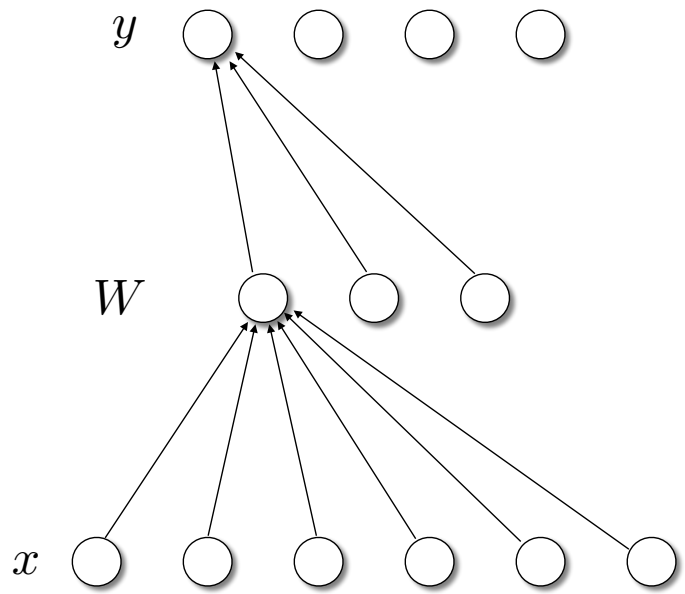
Next

$$f(x) = w^\top \sigma(Wx) = \sum_{j=1}^{p} w^j \sigma(x^\top W^j).$$

- We compare with other function models we have seen.
- We make sense of the name.
- **We go deep.**

Multilayer neural networks (deep)

More neurons can be stacked to obtain more complex representation and functions.



$$f(x) = w^\top \sigma(\, W_L \ldots \sigma(\, W_2 \sigma(\, W_1 x)))$$

Here w, $W_1, \ldots, W_L$ are weights associated to different stacks of neurons.

Neural networks terminology

$$f(x) = w^\top \sigma(W_L \dots \sigma(W_2 \sigma(W_1 x)))$$

- Stacks of neurons are called layers.
- Each intermediate representation corresponds to a (hidden) layer.
- The wieghts $W_\ell \in \mathbb{R}^{p_\ell, p_{\ell-1}}$ correspond to hidden units ($p_0 = d$).
- The dimensionalities $(p_\ell)_\ell$ are the number of hidden units per layer.
- The non linearity is called activation function.

Neural networks terminology

$$f(x) = w^\top \sigma(W_L \dots \sigma(W_2 \sigma(W_1 x)))$$

- Called multilayer networks...

- or deep networks.

Convolutional neural networks

From

$$f(x) = \sum_{j=1}^{M} w^j \sigma(x^\top W^j),$$

to

$$f(x) = \sum_{j=1}^{M} w^j \left\| \sigma(W^j \star x) \right\|_\infty ,$$

where:

- pooling

$$\|\beta\|_\infty = \max_j |\beta_j|,$$

- convolution

$$(\beta \star x)_k = \sum_{i=1}^{d} \beta^i x^{t-i}.$$

Convolutions

Convolution is the linear map

$$\beta \star x = C_\beta x$$

C_β is a circulant matrix, that is each row is a shifted copy of s.

Weight sharing

Let

$$C_W = (C_{W^1}, \ldots, C_{W^M}).$$

Then

$$(\sigma(W^1 \star x), \ldots, \sigma(W^M \star x)) = \sigma(C_W x)$$

a standard neural nets with repeated weights.

Pooling

Compared to classical neural nets CNN have structured nonlinearities.

Other form of pooling can be considered

- average pooling

$$\frac{1}{d}\sum_{j=1}^{d}\beta_j,$$

- ℓ_p pooling

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

Why CNN?

- Invariance.

- Hierarchical compositionality.

Neural networks function spaces

Back to computations.

Note that, the space of functions of the form

$$f(x) = w^\top \sigma(W_L \dots \sigma(W_2 \sigma(W_1 x)))$$

does not inherit a linear structure, hence the inner product/norm, from the corresponding weghts.

This makes function spaces of neural networks harder to study.

Function spaces by composition

It is a space of functions of the form

$$f(x) = w^\top \Phi_L \circ \ldots, \Phi_2 \circ \Phi_1(x),$$

that is is obtained composing multiple feature maps,

$$\Phi_\ell(x) = \sigma(W_\ell x), \qquad \ell = 1, \ldots, L.$$

The feature maps are learned and not fixed a priori.

ERM for neural nets

Let

$$f(x, W) = w^\top \sigma(W_L \dots \sigma(W_2 \sigma(W_1 x)))$$

with $W = (w, (W_\ell)_\ell)$ the set of coefficients.

ERM

$$\min_W \sum_{i=1}^n \ell(y_i, f(x_i, W)),$$

possibly with norm constraints on the weights (regularization).

The problem is non-convex and possibly non smooth depending on ℓ, σ.

Optimization for neural nets

Let

$$\widehat{L}(W) = \sum_{i=1}^{n} \ell(y_i, f(x_i), W).$$

In the case, of convex loss/activation SGD is the algorithm of choice,

$$W_{t+1} = W_t + \gamma_t \nabla_W \widehat{\ell}(y_t, f(x_y, W_t))$$

where with an abuse we denote by $\nabla \ell(y_t, f(x_y, W_t))$ a subgradient.

Backpropagation

Computing the (sub) gradient requires the chain rule, since

$$f(x) = w^\top \Phi_L \circ \ldots, \Phi_2 \circ \Phi_1(x).$$

Backpropagation allows for efficient computations.

Automatic differentiation, and the method of self-adjoint are relevant ideas.

Hacks and tricks

A huge number of tricks which are often used but outside our scopes.

- Batch/weight normalization.
- Dropout.
- Multiple step-sizes.
- ...

Summing up

- Data representation & neural nets.
- Comparison with other function classes.
- Biological interpretation and multilayers
- Computations
- Convolutional networks