MIT 9.520/6.7910
Statistical Learning Theory and Applications

Class 06: Neural networks

Lorenzo Rosasco

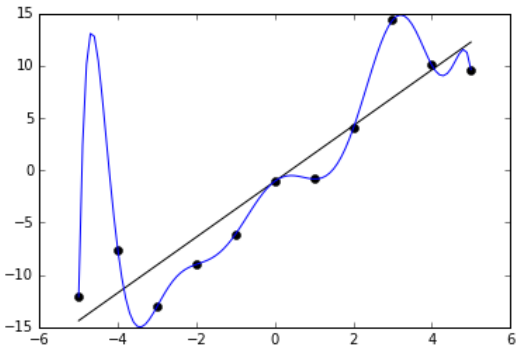
Learning algorithm design so far

- ERM+bias/regularization+optimization
- Implicit regularization

So far only linear models...

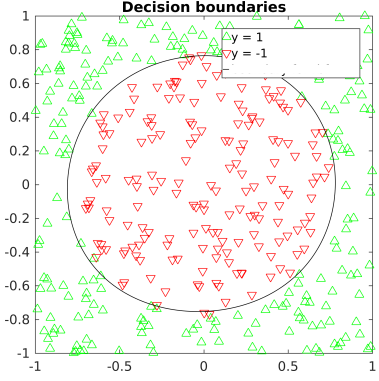
Limits of linear functions

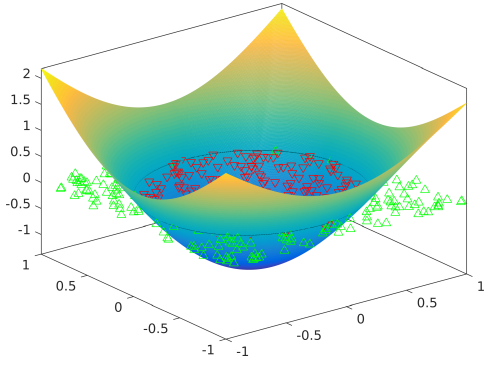
Regression



Limits of linear functions

Classification





Beyond linear models

$$x^\top w$$

Beyond linear models

$$x^\top w$$

- Kernels

$$\Phi(x)^\top w.$$

Beyond linear models

$$x^\top w$$

- Kernels

$$\Phi(x)^\top w.$$

- **Neural nets**

$$\sigma(x^\top w).$$

Outline

Neural nets are functions

$$\sigma(x^\top w)$$

Neural nets are functions

$$\sigma(x^\top w)$$

$$\Downarrow$$

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

Non linear functions

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

Examples of non linearities:

- ReLU $\sigma(\alpha) = |\alpha|_+$ (aka ramp, hinge),
- sigmoid $\sigma(\alpha) = 1/(1 + e^{-\alpha})$,
- hyperbolic tangent $\sigma(\alpha) = (e^{\alpha} - e^{-\alpha})/(e^{\alpha} + e^{-\alpha})$,
- softplus $\sigma(\alpha) = \log(1 + e^{\alpha})$.

RBF nets

Another class of non linearities,

$$f_\theta(x) = \sum_{j=1}^{u} w^j \sigma(\left\| W^j - x \right\|).$$

an example is $\sigma(\alpha) = e^{-\alpha^2 \gamma}$, that is

$$f_\theta(x) = \sum_{j=1}^{u} w^j e^{-\left\| W^j - x \right\|^2 \gamma}.$$

Outer non linearity

$$\sum_{j=1}^{u} w^j \sigma(x^\top W^j) \quad \text{or} \quad \sigma(\sum_{j=1}^{u} w^j \sigma(x^\top W^j)) \quad ?$$

Outer non linearity

$$\sum_{j=1}^{u} w^j \sigma(x^\top W^j) \quad \text{or} \quad \sigma(\sum_{j=1}^{u} w^j \sigma(x^\top W^j)) \quad ?$$

- Regression: none,

Outer non linearity

$$\sum_{j=1}^{u} w^j \sigma(x^\top W^j) \quad \text{or} \quad \sigma(\sum_{j=1}^{u} w^j \sigma(x^\top W^j)) \quad ?$$

- Regression: none,
- binary classification: sigmoid,

Outer non linearity

$$\sum_{j=1}^{u} w^j \sigma(x^\top W^j) \quad \text{or} \quad \sigma(\sum_{j=1}^{u} w^j \sigma(x^\top W^j)) \quad ?$$

- Regression: none,
- binary classification: sigmoid,
- multiclass: softmax.

Non linear parameterization

$$f_\theta(x) = \sum_{j=1}^{u} w^j \sigma(x^\top W^j), \qquad \theta = (w, W).$$

Outline

Learning representations

Data representation:

$$x \mapsto \Phi(x)$$

aka encoding, embedding, transform, features map.

Learning representations

Data representation:

$$x \mapsto \Phi(x)$$

aka encoding, embedding, transform, features map.

Let $W = (W^1, \ldots, W^u)$,

Learning representations

Data representation:

$$x \mapsto \Phi(x)$$

aka encoding, embedding, transform, features map.

Let $W = (W^1, \dots, W^u)$, then

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

Learning representations

Data representation:

$$x \mapsto \Phi(x)$$

aka encoding, embedding, transform, features map.

Let $W = (W^1, \ldots, W^u)$, then

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

Representation learning

$$\sum_{j=1}^{u} w^j \sigma(x^\top W^j) = w^\top \sigma(Wx) = w^\top \Phi(x)$$

Representation learning

$$\sum_{j=1}^{u} w^j \sigma(x^\top W^j) = w^\top \sigma(Wx) = w^\top \Phi(x)$$

Neural net representation

$$\Phi(x) = \sigma(Wx)$$

with W learned from data.

Representation learning vs engineering

$$w^\top \Phi(x), \qquad \Phi(x) = \sigma(Wx)$$

- Fourier, wavelet, sift, hog...

Representation learning vs engineering

$$w^\top \Phi(x), \qquad \Phi(x) = \sigma(Wx)$$

- Fourier, wavelet, sift, hog...
- Sparse coding, PCA.

Representation learning vs engineering

$$w^\top \Phi(x), \qquad \Phi(x) = \sigma(Wx)$$

- Fourier, wavelet, sift, hog...
- Sparse coding, PCA.
- End to end learning.

$$\min_{\theta=(w,W)} \widehat{L}(f_\theta)$$

Outline

Why neural?

Meet a neuron:

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

Network of neurons

Playing lego with neurons:

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

Jargoon

Playing lego with neurons:

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

- Activation functions σ.

Jargoon

Playing lego with neurons:

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

- Activation functions σ.
- Network weights $\theta = (w, W)$, $W = (W^1, \ldots, W^u)$.

Jargoon

Playing lego with neurons:

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

- Activation functions σ.
- Network weights $\theta = (w, W)$, $W = (W^1, \ldots, W^u)$.
- Hidden units $\sigma(x^\top W^j)$, $j = 1, \ldots, u$.

Jargoon

Playing lego with neurons:

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

- Activation functions σ.
- Network weights $\theta = (w, W)$, $W = (W^1, \dots, W^u)$.
- Hidden units $\sigma(x^\top W^j)$, $j = 1, \dots, u$.
- Hidden layer $\sigma(Wx)$.

Jargoon

Playing lego with neurons:

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

- Activation functions σ.
- Network weights $\theta = (w, W)$, $W = (W^1, \ldots, W^u)$.
- Hidden units $\sigma(x^\top W^j)$, $j = 1, \ldots, u$.
- Hidden layer $\sigma(Wx)$.
- Network width u.

Outline

Weight decay and normalization

$$x^\top w$$

Weight decay and normalization

$$x^\top w$$

- Weight decay

$$\|w\| \leq R.$$

Weight decay and normalization

$$x^\top w$$

- Weight decay

$$\|w\| \leq R.$$

- Weight normalization

$$x^\top w \mapsto \rho x^\top v,$$

with $\rho \in \mathbb{R}$ and $\|v\| = 1$.

Going deep

Playing lego with neurons is fun!

A more compact notation

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

A more compact notation

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

Deep nets with L layers,

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

A more compact notation

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

Deep nets with L layers,

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

Alternatively $h_0 = x$, $\ell = 1, \dots, L-1$,

$$h_\ell = \sigma(W_\ell h_{\ell-1})$$

and $h_L = w^\top h_{L-1}$.

Going deep: but why?

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

Why deep? Compositionality

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

Why deep? Compositionality

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

- Compositionality

$$f(x) = f_L \circ \dots f_2 \circ f_1(x).$$

e.g. $d = 4$, $\quad f(x) = f(x^1, \dots, x^4) = f_3(f_2(x^1, x^2), f_1(x^3, x^4))$

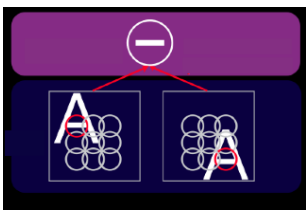
Why deep? Invariance

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

Why deep? Invariance

$$w^\top \sigma(W_{L-1} \dots W_2 \sigma(W_1 x)).$$

- Invariance



Convolutional networks

$$\sigma(x^\top w)$$

$$\Downarrow$$

$$P(x \star w)$$

Convolutional networks

$$\sigma(x^\top w)$$

$$\Downarrow$$

$$P(x \star w)$$

- Convolution (weight sharing)

$$(x \star w)^k = \sum_{j=0}^{d} x^j w^{k-j}$$

Convolutional networks

$$\sigma(x^\top w)$$

$$\Downarrow$$

$$P(x \star w)$$

- Convolution (weight sharing)

$$(x \star w)^k = \sum_{j=0}^{d} x^j w^{k-j}$$

- Pooling

$$P(z) = \max_j |z^j|$$

but also soft max, average, etc.

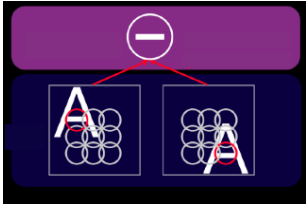
Convolutional networks (cont.)

$$P(x \star w)$$

Convolutional networks (cont.)

$$P(x \star w)$$

- Invariance.



Residual networks

Let $h_0 = x$,

$$h_\ell = \sigma(W_\ell h_{\ell-1})$$

and $h_L = w^\top h_{L-1}$.

Residual networks

Let $h_0 = x$,

$$h_\ell = \sigma(W_\ell h_{\ell-1})$$

and $h_L = w^\top h_{L-1}$.

$$\Downarrow$$

Let $h_0 = x$,

$$h_\ell = \sigma(W_\ell h_{\ell-1}) + h_{\ell-1},$$

and $h_L = w^\top h_{L-1}$.

Autoencoders

Autoencoders

$$f(x) \sim x$$

Autoencoders

$$f(x) \sim x$$

Encoder and decoder

$$f = \Psi \circ \Phi$$

Autoencoders

$$f(x) \sim x$$

Encoder and decoder

$$f = \Psi \circ \Phi$$

with a "bottleneck"

$$\Phi(x) = \sigma(Wx) \qquad \text{and} \qquad \Psi(h) = \sigma(W^\top h).$$

Recurrent networks

$$f(x_t) \sim x_{t+1}$$

Recurrent networks

$$f(x_t) \sim x_{t+1}$$

$$\begin{aligned} h_t &= q(h_{t-1}, x_t) \\ x_{t+1} &= g(h_t). \end{aligned}$$

Networks for all!

Networks for all!

There are many more tricks!

Networks for all!

There are many more tricks!

- Dropout.
- Batch normalization.
- ...

Networks for all!

There are many more tricks!

- Dropout.
- Batch normalization.
- ...

And more networks!

Networks for all!

There are many more tricks!

- Dropout.
- Batch normalization.
- ...

And more networks!

- Neural ODE.
- Generative networks.
- Transformers.
- Diffusion models.
- ...

Outline

Training with SGD

$$\min_{\theta=(w,W)} \widehat{L}(f_\theta), \qquad \widehat{L}(f_\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i))$$

Training with SGD

$$\min_{\theta=(w,W)} \widehat{L}(f_\theta), \qquad \widehat{L}(f_\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i))$$

For deep nets $\theta = (w, (W_\ell)_\ell)$.

Training with SGD

$$\min_{\theta=(w,W)} \widehat{L}(f_\theta), \qquad \widehat{L}(f_\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_\theta(x_i))$$

For deep nets $\theta = (w, (W_\ell)_\ell)$.

SGD

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Training with SGD+ tricks

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Many commonly used variations...

Training with SGD+ tricks

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Many commonly used variations...

- Acceleration (e.g. momentum).

Training with SGD+ tricks

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Many commonly used variations...

- Acceleration (e.g. momentum).
- Adaptive step-size choice.

Training with SGD+ tricks

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Many commonly used variations...

- Acceleration (e.g. momentum).
- Adaptive step-size choice.
- Mini-batching.

Training with SGD+ tricks

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Many commonly used variations...

- Acceleration (e.g. momentum).
- Adaptive step-size choice.
- Mini-batching.
- Stochastic noise.
- ...

Computing gradients

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

Computing gradients

$$\theta_{t+1} = \theta_t - \gamma_t \nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})).$$

$$\nabla_\theta \ell(y_{i_t}, f_{\theta_t}(x_{i_t})) = \nabla_{f_{\theta_t}(x_{i_t})} \ell(y_{i_t}, f_{\theta_t}(x_{i_t})) \nabla_\theta f_{\theta_t}(x_{i_t})$$

Gradients of composite function

$$\nabla_\theta f_{\theta_t}(x_{i_t})$$

Gradients of composite function

$$\nabla_\theta f_{\theta_t}(x_{i_t})$$

Let

$$f_\theta(x) = g_x(\theta)$$

Gradients of composite function

$$\nabla_\theta f_{\theta_t}(x_{i_t})$$

Let

$$f_\theta(x) = g_x(\theta) = g(\theta),$$

Gradients of composite function

$$\nabla_\theta f_{\theta_t}(x_{i_t})$$

Let

$$f_\theta(x) = g_x(\theta) = g(\theta),$$

then

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

Gradients of composite function

$$\nabla_\theta f_{\theta_t}(x_{i_t})$$

Let

$$f_\theta(x) = g_x(\theta) = g(\theta),$$

then

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

Equivalently $v_0 = \theta$, $\ell = 1, \dots L$

$$v_\ell = g_\ell(v_{\ell-1}).$$

Chain rule

Compute derivative of

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

[1] Otherwise use gradients and Jacobians!

Chain rule

Compute derivative of

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

Pedagogic simplification: g_ℓ real valued and θ is a scalar [1].

[1] Otherwise use gradients and Jacobians!

Chain rule

Compute derivative of

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

Pedagogic simplification: g_ℓ real valued and θ is a scalar [1].

$$L = 2, \qquad g'(\theta) = g_2'(h_1(\theta))g_1'(\theta).$$

[1] Otherwise use gradients and Jacobians!

Chain rule

Compute derivative of

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

Pedagogic simplification: g_ℓ real valued and θ is a scalar [1].

$$\begin{aligned} L &= 2, \qquad g'(\theta) = g_2'(h_1(\theta))g_1'(\theta). \\ L &= 3, \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta). \end{aligned}$$

[1] Otherwise use gradients and Jacobians!

Chain rule

Compute derivative of

$$g(\theta) = g_L \circ \dots g_2 \circ g_1(\theta).$$

Pedagogic simplification: g_ℓ real valued and θ is a scalar [1].

$$\begin{aligned} L &= 2, \qquad g'(\theta) = g_2'(h_1(\theta))g_1'(\theta). \\ L &= 3, \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta). \end{aligned}$$

Idea: Compute derivatives recursively!

[1] Otherwise use gradients and Jacobians!

Backprop

Let $g = g_L \circ \dots g_2 \circ g_1$, compute $g'(\theta)$.

Backprop

Let $g = g_L \circ \dots g_2 \circ g_1$, compute $g'(\theta)$.

Keep in mind

$$g(\theta) = g_3(g_2(g_1(\theta))), \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

Backprop

Let $g = g_L \circ \dots g_2 \circ g_1$, compute $g'(\theta)$.

Keep in mind

$$g(\theta) = g_3(g_2(g_1(\theta))), \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

Forward pass For $\ell = 1, \dots, L$, $v_0 = \theta$

$$v_\ell = g_\ell(v_{\ell-1}).$$

Backprop

Let $g = g_L \circ \dots g_2 \circ g_1$, compute $g'(\theta)$.

Keep in mind

$$g(\theta) = g_3(g_2(g_1(\theta))), \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

Forward pass For $\ell = 1, \dots, L$, $v_0 = \theta$

$$v_\ell = g_\ell(v_{\ell-1}).$$

Backward pass For $\ell = L, \dots, 1$, $z_{L+1} = 1$

$$z_\ell = z_{\ell+1} g_\ell'(v_{\ell-1}).$$

Backprop

Forward pass For $\ell = 1, \ldots, L$, $v_0 = \theta$

$$v_\ell = g_\ell(v_{\ell-1}).$$

The forward pass evaluate the function recursively.

Backprop

Forward pass For $\ell = 1, \dots, L$, $v_0 = \theta$

$$v_\ell = g_\ell(v_{\ell-1}).$$

The forward pass evaluate the function recursively.

Backward pass For $\ell = L, \dots, 1$, $z_{L+1} = 1$

$$z_\ell = z_{\ell+1} g'_\ell(v_{\ell-1}).$$

The backward pass evaluate the function derivative recursively.

Automatic differentiation

$$g(\theta) = g_3(g_2(g_1(\theta))), \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

Backprop is an example of automatic differentiation (reverse mode).

Automatic differentiation

$$g(\theta) = g_3(g_2(g_1(\theta))), \qquad g'(\theta) = g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

Backprop is an example of automatic differentiation (reverse mode).

Forward mode is starting computing derivative from the right!

Automatic differentiation

$$g(\theta) = g_3(g_2(h_1(\theta))), \qquad g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

So what we need to use backprop/automatic differentiation?

Automatic differentiation

$$g(\theta) = g_3(g_2(h_1(\theta))), \qquad g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

So what we need to use backprop/automatic differentiation?

- A "dictionary" of elementary functions $(g_\ell)_\ell$, and their derivatives $(g_\ell')_\ell$.

Automatic differentiation

$$g(\theta) = g_3(g_2(h_1(\theta))), \qquad g_3'(g_2(g_1(\theta)))g_2'(g_1(\theta))g_1'(\theta).$$

So what we need to use backprop/automatic differentiation?

- A "dictionary" of elementary functions $(g_\ell)_\ell$, and their derivatives $(g_\ell')_\ell$.
- A way to express our function h in the form

$$h = h_L \circ \dots h_2 \circ h_1.$$

  which can be described by a graph, the computational graph.

Summing up

- Neural nets are non linear functions.
- Representation learning.
- Neural architectures.
- Backprop.