STAT 535

10/31/2023

Lecture 10

Lecture Notes III – Neural Networks

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

October, 2023

Multi-layer/Deep neural networks

The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}, n_0 = n, n_L = 1$ and define the recursion:

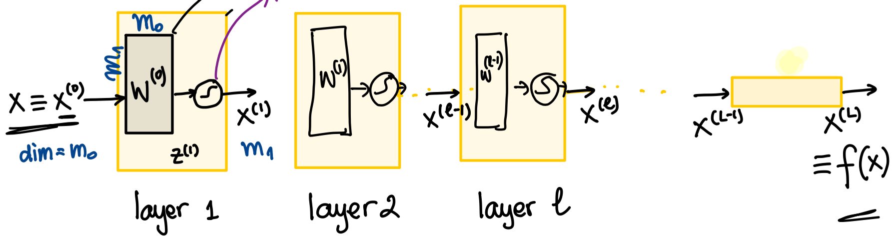
layers

$l = 1, 2, 3, \ldots, L$

$$x_j^{(l)} = \phi\left(\underbrace{(w_j^{(l)})^T x^{(l-l)}}_{\text{linear}}\right), \text{ for } j = 1 : n_l \tag{9}$$

The vector variable $x^{(l)} \in \mathbb{R}^{n_l}$ is the ouput of layer l of the network. As before, the sigmoid of the last layer may be omitted.

non-linearity



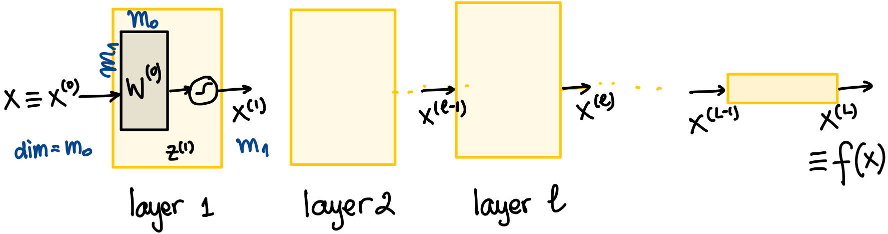
Multi-layer/Deep neural networks

The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}, n_0 = n, n_L = 1$ and define the recursion:

$$x_j^{(l)} = \phi\left(\underbrace{(w_j^{(l)})^T x^{(l-1)}}_{z} \right), \text{ for } j = 1 : n_l \tag{9}$$

$W^{(l+1)}$ $\quad x_j^{(l)} = W^{(l+1)} \phi(\quad z \quad)$

The vector variable $x^{(l)} \in \mathbb{R}^{n_l}$ is the ouput of layer l of the network. As before, the sigmoid of the last layer may be omitted.



Forward propagation: $\quad x \longmapsto f(x)$

$\Updownarrow$

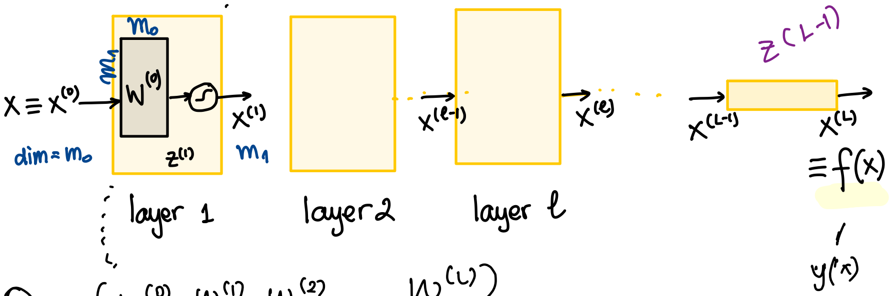
$x(0) \longmapsto x(L)$

Multi-layer/Deep neural networks

The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}$, $n_0 = n, n_L = 1$ and define the recursion:

$$x_j^{(l)} = \phi\left(\underbrace{(w_j^{(l)})^T x^{(l-l)}}_{z^{(l)}}\right), \text{ for } j = 1 : n_l \tag{9}$$

The vector variable $x^{(l)} \in \mathbb{R}^{n_l}$ is the ouput of layer l of the network. As before, the sigmoid of the last layer may be omitted.



$\Theta = (W^{(0)}, W^{(1)}, W^{(2)}, \ldots, W^{(L)})$

$L(f(x^i), y^i)$

Training data i

$$(x^{(i)}, y^{(i)}) \rightarrow f(x^i)$$

$$L(f(x^i), y^{(i)})$$

for example:

$$L(f(x^i), y^{(i)}) = (f(x^i) - y(i))^2$$

$$\nabla_\theta L(f(x^i), y(i)) = \frac{\partial L}{\partial \theta}$$

$$z^{(l)} = w^{(l)T} x^{(l-1)}$$

$$f(x_i) = \phi(z_i^{L-1})$$

chain rule

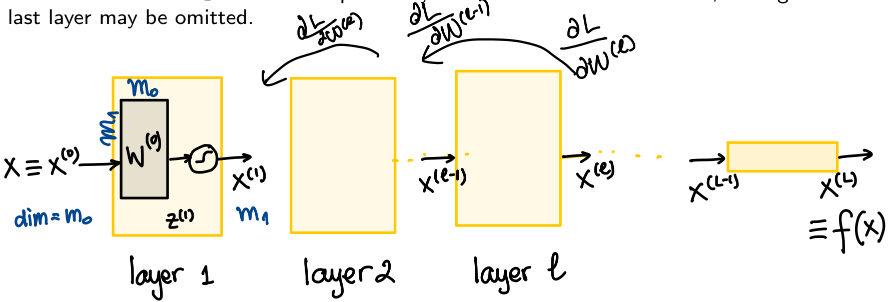
$$\frac{\partial L}{\partial W^{(L-1)}} = \frac{\partial L}{\partial f(x_i)} \cdot \frac{\partial f(x_i)}{\partial z^{(L-1)}} \cdot \underbrace{\frac{\partial z^{(L-1)}}{\partial W^{(L-1)}}}_{\underline{x}^{(L-2)}}$$

Multi-layer/Deep neural networks

The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}$, $n_0 = n, n_L = 1$ and define the recursion:

$$x_j^{(l)} = \phi\left((w_j^{(l)})^T x^{(l-l)}\right), \text{ for } j = 1 : n_l \tag{9}$$

The vector variable $x^{(l)} \in \mathbb{R}^{n_l}$ is the ouput of layer l of the network. As before, the sigmoid of the last layer may be omitted.



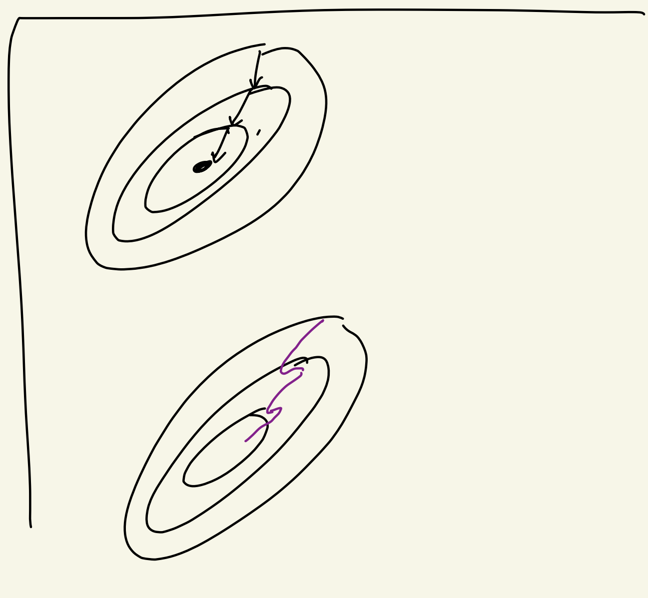
Backpropagation.

$$z^{(l-1)} = W^{(l-1)} \phi(z^{l-2})$$

$$z^{(l-2)} = W^{(l-2)} x^{(l-3)}$$

$$\frac{\partial L}{\partial W^{(L-2)}} = \boxed{\frac{\partial L}{\partial z^{(L-2)}}} \cdot \boxed{\frac{\partial z^{(L-2)}}{\partial W^{(L-2)}}} \rightarrow x^{(L-3)}$$

$$\frac{\partial L}{\partial z^{(L-2)}} = \boxed{\frac{\partial L}{\partial z^{(L-1)}}} \cdot \frac{\partial z^{(L-1)}}{\partial z^{(L-2)}}$$

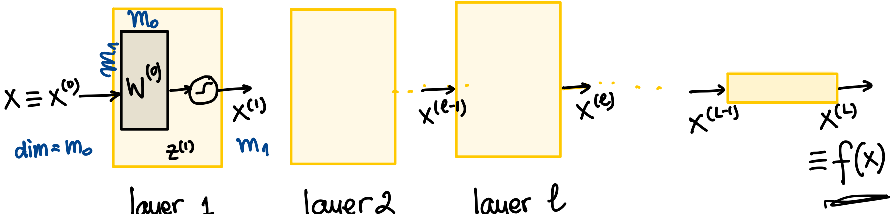


Multi-layer/Deep neural networks

The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}$, $n_0 = n, n_L = 1$ and define the recursion:

$$x_j^{(l)} = \phi\left((w_j^{(l)})^T x^{(l-l)}\right), \text{ for } j = 1 : n_l \tag{9}$$

The vector variable $x^{(l)} \in \mathbb{R}^{n_l}$ is the ouput of layer l of the network. As before, the sigmoid of the last layer may be omitted.



Forward propagation : $x_i(0) \to x_i(1) \to \cdots \to x_i(\ell) \to \cdots \to x_i(L) = f(x_i)$

Backward propagation : $\frac{\partial L}{\partial z^{L-1}}, \frac{\partial L}{\partial w^{L-1}}, \frac{\partial L}{\partial z^{L-2}}, \frac{\partial L}{\partial w^{L-2}}, \cdots \frac{\partial L}{\partial z^{(1)}}, \frac{\partial L}{\partial w^{(1)}}$

Are multiple layers necessary?

- **1990's: NO**
- **2000's: YES**
- **2020's: The more the better!**

- A theoretical result

Theorem (Cybenko,≈1986)

Any continuous function from $[0,1]^d$ to $\mathbb{R}$ can be approximated arbitrarily closely by a linear output, two layer neural network defined in (2) *with a sufficiently large number of hidden units* m.

- A practical result

Deep learning = multi-layer neural net

- So, what is new?
  - small variations in the "units", e.g. switch stochastically w.p. $\phi(w^T x^{in})$ (Restricted Bolzmann Machine), Rectified Linear units
  - training method stochastic gradient, auto-encoders vs. back-propagation (we will return to this when we talk about training predictors)
  - lots of data
  - **double descent**

Resnets – Residual networks

Idea What is the "simplest" input-output function? $\boxed{f_0(x) = x}$ (identity function)

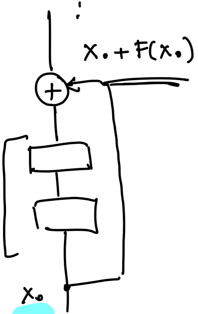
- Hence, a NN layer should learn the difference w.r.t. identity f_0

$$x_{l+1} \;=\; B_l \phi(W_l x_l) + x_l \qquad (10)$$

non-linear

Generalization DenseNet

- Layer l gets inputs from $l-1, l-2, \ldots$



"residual function"

$$\boxed{\begin{aligned} &\text{Goal: } f_0(x) = x \\ &F(x) = f_0(x) - \cancel{y}\, x \end{aligned}}$$

y

$f(x) = mx + b$

ConvNets – Convolutional Networks

- **discrete convolution** let $f, g : \mathbb{Z} \to \mathbb{R}$
  $\mathbb{Z} =$ all integers

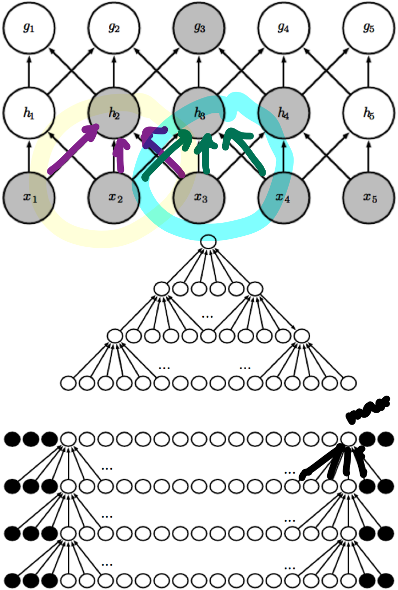
w(i), wi

$$(f * g)(t) \;=\; \sum_{i \in \mathbb{Z}} f(t-i)g(i) \tag{11}$$

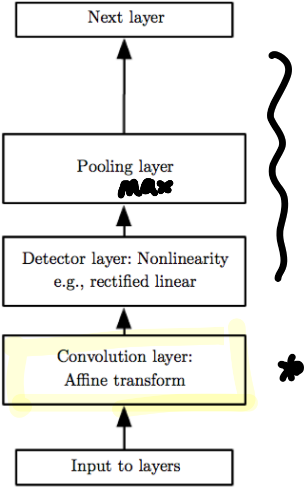
- convolution as **Toeplitz** matrix vector multiplication
- in ConvNets, $\mathbb{Z}$ is replaced by $1:m$, f is **padded with 0's**
  - g is a (smoothing) kernel
  - i.e. $g(i) = g(-i) > 0$ and $|\operatorname{supp} g| = 2s+1 \ll m$, $\sum_i g(i) = 1$
- Convolutional layer $f \leftarrow x$ input, $g \leftarrow w$ weights, s output

$$s(t) = \sum_{i=t-s}^{t+s} w_i s(t-i) \quad = (s * w)(t) \tag{12}$$

- Pooling



from www.deeplearningbook.org Chapter 9



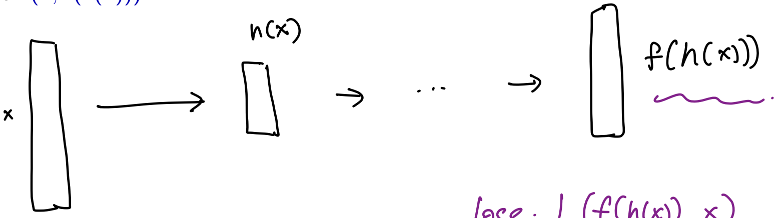
Autoencoders ← unsupervised learning

estion How to learn from data without outputs y?
This is unsupervised learning, not prediction

Idea Learn a low dimensional/sparse representation $h(x)$ of data $x \in \mathbb{R}^d$

$$h(x) \in \mathbb{R}^{m}, \text{ with } m \leq d \quad f(h(x)) \approx x! \tag{13}$$

- Optimize $L(x, f(h(x)))$



loss : $L(f(h(x)), x)$
↓
optimize this

- compressing input data

Autoencoders

estion How to learn from data without outputs y?
This is unsupervised learning, not prediction

Idea Learn a low dimensional/sparse representation $h(x)$ of data $x \in \mathbb{R}^d$

$$h(x) \in \mathbb{R}^m, \text{ with } m < d \quad f(h(x)) \approx x! \tag{13}$$

- Optimize $L(x, f(h(x)))$