STAT 535

11/1/22

Happy  !

Lecture 10

Multilayer NNs

- L II.1 KR updated
- L IV-1 Training posted
- HW 3 due tomorrow
- Q2 ≥ Tuesday 11/8
  Basic concepts
- Q + HW grading policy UPDATE
- Sol 1 → available

Lecture Notes III – Neural Networks

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

October, 2020

Two-layer Neural Networks

Multi-layer neural networks

A zoo of multilayer networks

Reading HTF Ch.: 11.3 Neural networks, Murphy Ch.: (16.5 neural nets) and Dive Into Deep Learning 4.1-4.3

Two-layer Neural Networks

- The **activation function** (a term borrowed from neuroscience) is any continuous, bounded and strictly increasing function on $\mathbb{R}$. Almost universally, the activation function is the **logistic** (or **sigmoid**)

$$\phi(z) = \frac{1}{1+e^{-z}} \tag{1}$$

because of its nice additional computational and statistical properties.

- We build a **two-layer neural network** in the following way:

Inputs	x_k	$k = 1 : n$
Bottom layer[1]	$z_j = \phi(w_j^T x)$	$j = 1 : m,\ w_j \in \mathbb{R}^n$
Top layer	$f = \phi(\beta^T z)$	$\beta \in \mathbb{R}^m$
Output	f	$\in [0, 1]$

In other words, the neural network implements the function

$$f(x) = \sum_{j=1}^{m} \beta_j z_j = \sum_{j=1}^{m} \beta_j \phi\left(\sum_{k=1}^{m} w_{kj} x_k\right) \in (-\infty, \infty) \tag{2}$$

Note that this is just a linear combination of logistic functions.

[1] In neural net terminology, each variable z_j is a **unit**, the bottom layer is **hidden**, while top one is **visible**, and the units in this layer are called hidden/visible units as well. Sometimes the inputs are called **input units**; imagine neurons or individual circuits in place of each x, y, z variable.

Output layer options

- **linear** layer as in (2) $f = \sum_j \beta_j z_j$
- **logistic** layer: in classification $f(x) \in [0,1]$ is interpreted as the probability of the $+$ class.

$$f(x) \;=\; \phi\left(\sum_{j=1}^{m} \beta_j z_j\right) \;=\; \phi\left(\sum_{j=1}^{m} \beta_j \phi(\sum_j w_{kj} x_k)\right) \tag{3}$$

- **softmax** layer in multiway classification

  The softmax function $\phi(z) : \mathbb{R}^m \;\rightarrow\; (0,1)^m$

$$\phi_k(z) \;=\; \frac{e^{z_k}}{\sum_{j=1}^{m} e^{z_j}} \tag{4}$$

- Properties
  - $\sum_{j=1}^{m} \phi_j(z) = 1$ for all z
  - for $z_k \gg z_j$, $j \neq k$ $\phi_k(z) \rightarrow 1$.
  - derivatives $\frac{\partial \phi_j}{\partial z_k} \;=\; \phi_k \delta_{jk} - \phi_j \phi_k$

if 1)

Generalized Linear Models (GLM)

A GLM is a regression where the "noise" distribution is in the exponential fami ly.

- $y \in \mathbb{R}$, $y \sim P_\theta$ with

$$P_\theta(y) \;=\; e^{\theta y - \ln \psi(\theta)} \tag{5}$$

linear in x AND β

- the parameter θ is a linear function of $x \in \mathbb{R}^d$

$$\theta = \underbrace{\beta^T x}_{z(l)} \tag{6}$$

- We denote $E_\theta[y] = \mu$. The function $g(\mu) = \theta$ that relates the mean parameter to the natural parameter is called the **link function**.

The log-likelihood (w.r.t. β) is

true value

model prediction

$$2)\quad l(\beta) \;=\; \ln P_\theta(y|x) \;=\; \theta y - \psi(\theta) \quad \text{where } \theta = \beta^T x \tag{7}$$

and the gradient w.r.t. β is therefore

$$\frac{\partial l}{\partial \theta} = y - \mu(\theta)$$

$$\nabla_\beta l \;=\; \nabla_\theta l \nabla_\beta(\beta^T x) \;=\; (y - \mu)x \tag{8}$$

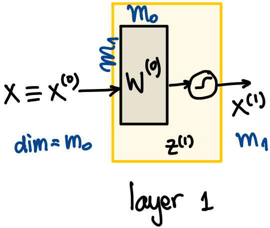
This simple expression for the gradient is the generalization of the gradient expression you obtained for the two layer neural network in the homework. [Exercise: This means that the sigmoid function is the *inverse link function* defined above. Find what is the link function that corresponds to the neural network.]

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 $x \equiv x^{(0)}, y \equiv x^{(L)}, m_0 = d, m_L = 1$ and define the recursion:

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

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.



$\beta^T x^{(1)} = y \equiv x^{(2)}$

Multi-layer/Deep neural networks

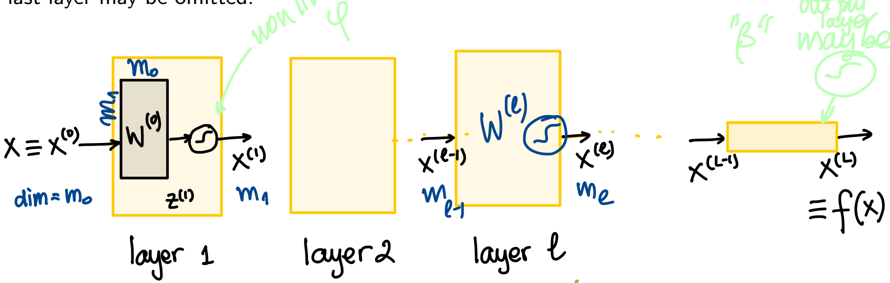
The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}, n_0 = n, n_L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

Recursion $\Rightarrow f(x)$

$$x_j^{(l)} = \phi\left(\overbrace{(w_j^{(l)})^T x^{(l-1)}}^{\text{linear}}\right), \text{ for } j = 1 : n_l \qquad l = 0, .., L \tag{9}$$

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 lin φ



Forward propagation

$X \longmapsto f(x)$

$X^{(0)} \longmapsto X^{(L)}$

computation $\sim m_0 m_1 + m_1 m_2 + \dots m_{L-1} m_L$, $m_L = 1$

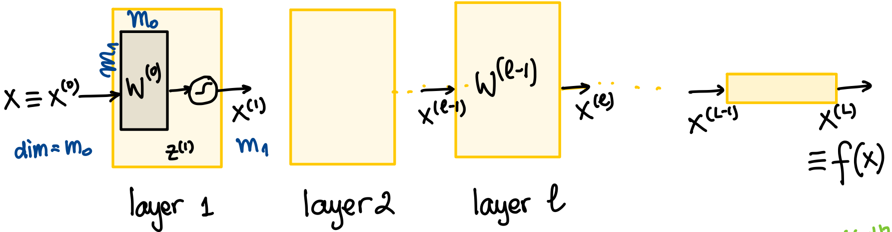
Multi-layer/Deep neural networks

Back-propagation

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:

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

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.



wanted

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

θ, L for given θ

$\theta = \{W^{(0)}, W^{(1)}, \ldots W^{(L-1)}\}$

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

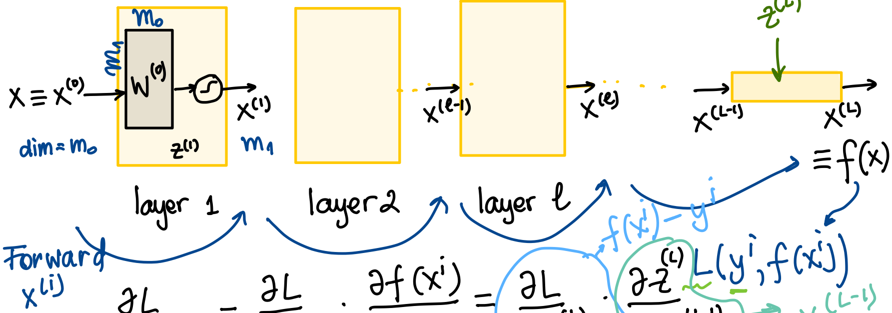
Multi-layer/Deep neural networks

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.

$\partial L/\partial W^{(L-1)}$



$f(x^i) - y^i$ $\quad L(y^i, f(x^i))$

$$\frac{\partial L}{\partial W^{(L-1)}} = \frac{\partial L}{\partial f(x^i)} \cdot \frac{\partial f(x^i)}{\partial W^{(L-1)}} = \frac{\partial L}{\partial z^{(L)}(x^i)} \cdot \frac{\partial z^{(L)}}{\partial W^{(L-1)}}$$

$x^{(L-1)}(x^i)$

GLiM

Multi-layer/Deep neural networks

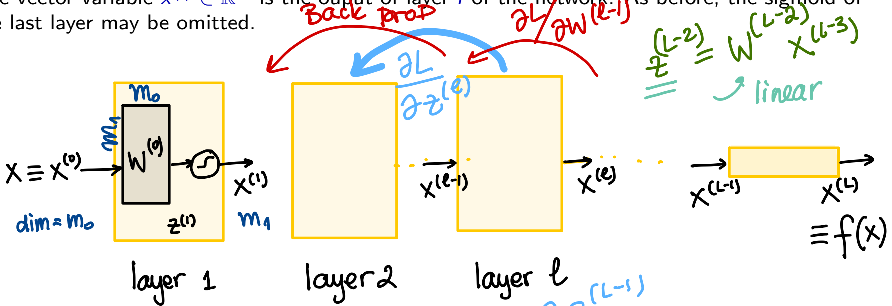
The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}, n_0 = n, n_L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

$$z^{(\ell)} = W^{(\ell-1)} \varphi\left(z^{(\ell-1)}\right)$$

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

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.

gradient at layer $\ell = L-2$



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

linear

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

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

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

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

$$W^{(L-2)} = \frac{\partial z^{(L-1)}}{\partial x^{(L-2)}} \quad \text{linear}$$

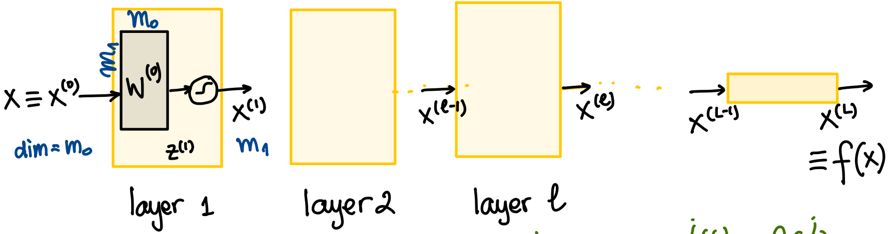
$$\varphi'\left(z^{(L-2)}\right) = \frac{\partial x^{(L-2)}}{\partial z^{(L-2)}}$$

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.



computation : forward $\quad x^{i(0)} \to x^{i(1)} \cdots \to x^{i(L)} = f(x^i)$

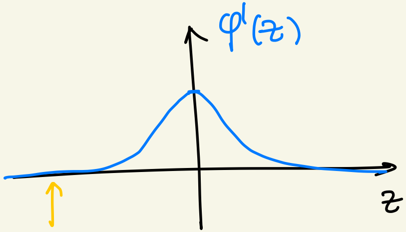
(data point i)

need φ'

+ matrix-vec multiplication

backward $\quad \frac{\partial L}{\partial z^{(L)}}(x^i), \frac{\partial L}{\partial W^{(L-1)}}, \frac{\partial L}{\partial z^{(L-1)}}, \frac{\partial L}{\partial W^{(L-2)}} \cdots$

Issues with deep back prop. with $\varphi =$ 



$\varphi' \approx 0$ for $|z|$ large

is Solving a "ONE secret" of Deep L.

Implementation

- already implemented packages PyTorch (scipy) TF MXNet
- moreover – autodifferentiation "compiler feature"

In : [python] code for $f(x, w)$

Out : [—"—] —"— —"— $f'(x)$ $df/dw(x)$

Other remarks – layers can implement "unrolled"

Some solutions

- "Normalization" of W's
- ReLU

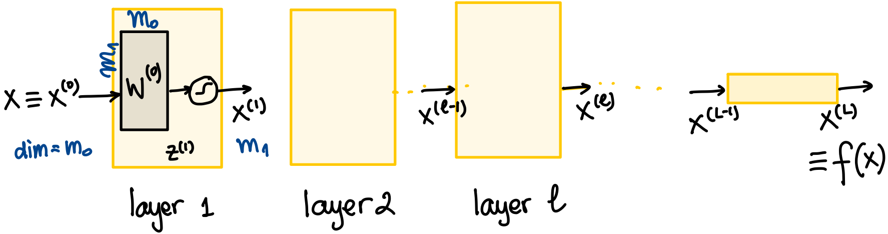
$\varphi' = 0$ $\varphi' = 1$

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.



Are multiple layers necessary?

- **1990's: NO**
- **2000's: YES**

- A theoretical result

Theorem (Cybenko,≈1986)

Any continuous function from $[0,1]^n$ *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.

m

- A practical result

MIT Technology Review

10 BREAKTHROUGH TECHNOLOGIES 2013

Deep Learning

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**

statistical

→ Many global minima

• Normalization

• large L, $m^{(l)}$ → non-parametric

Resnets – Residual networks

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

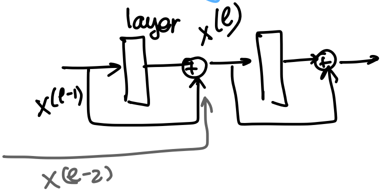
- Hence, a NN layer should learn the difference w.r.t. identity f_0

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

learn only non-linear

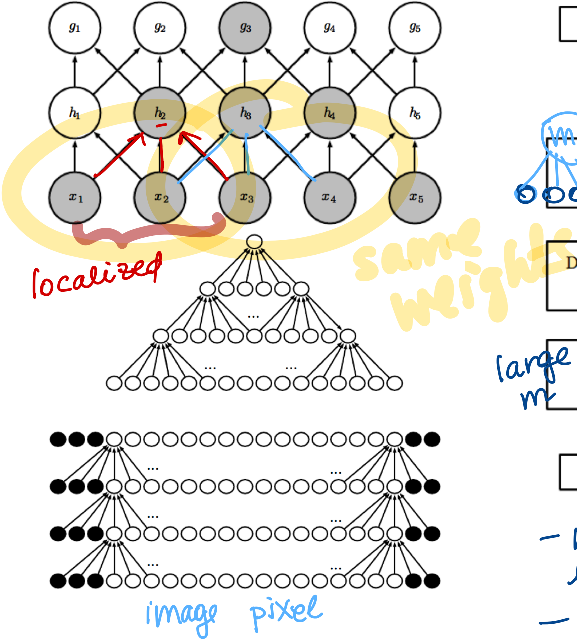
Generalization DenseNet

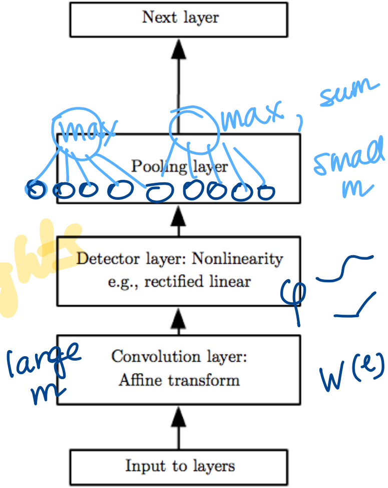
- Layer l gets inputs from $l-1, l-2, \ldots$



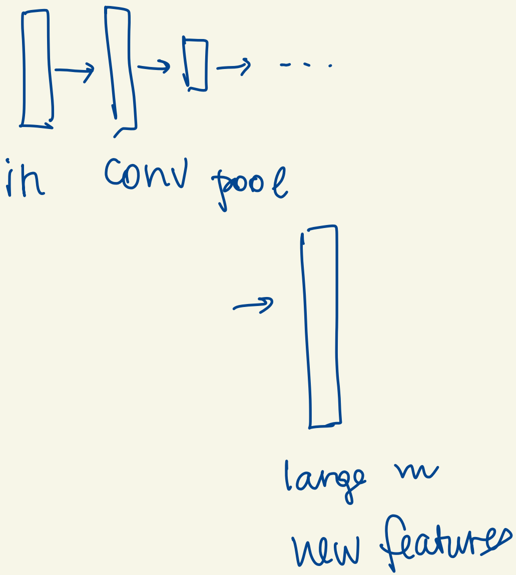
ConvNets – Convolutional Networks

- **discrete convolution** let $f, g : \mathbb{Z} \to \mathbb{R}$
  $\mathbb{Z} =$ all integers
  $$(f * g)(t) \;=\; \sum_{i \in \mathbb{Z}} f(t-i)g(i) \tag{11}$$
- convolution as **Toeplitz** matrix vector multiplication
- in ConvNets, $\mathbb{Z}$ is replaced by $1:n$, f is **padded with 0's**
  - g is a (smoothing) kernel
  - i.e. $g(i) = g(-i) > 0$ and $|\operatorname{supp} g| = 2m+1 \ll n$, $\sum_i g(i) = 1$
- Convolutional layer $f \leftarrow x$ input, $g \leftarrow w$ weights, s output
  $$s(t) = \sum_{i=t-m}^{t+m} w_i s(t-i) \tag{12}$$
- Pooling





- weights equal in layer ℓ
- weights localized


in
conv
pool
...
large m
new features

Autoencoders ← unsupervised

New architectures:
- generative "models"
  GAN
  VAE
  (Variational AE)
  Transformers

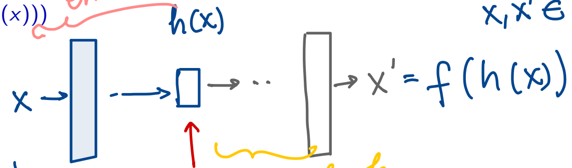
estion How to learn from data without outputs y?
This is unsupervised learning, not prediction

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

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

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

$x, x' \in \mathbb{R}^d$



low dimensional, OR sparse

decode h

Loss $(x - x')$

$f(x)$

⌊ discovers non-linear low dim representation

(OR) compresses X

Variations

- If f linear, L_{LS}, then we "learn" PCA
- Denoising autoencoder
  - Add noise to x input, predict true x

$$\tilde{x} \sim C(\,|x)$$

, $\min L(x, f(h(\tilde{x}))).(14)$

- Sparse autoencoder

$$\min L(x, f(h(x))) + \Omega(h) \tag{15}$$

Ω is regularization that makes h sparse

-