Lecture Notes III – Neural Networks

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

October, 2021

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), Deep Learning Book (Goodfellow, Bengio, Courville) 6.1-4, ResNet 7.6, ConvNet 9., Autoencoders 14.1, 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**)

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

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]	$z_j = \phi(w_j^T x)$	$j = 1 : m,\ w_j \in \mathbb{R}^d$
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

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

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 $\phi(z) : \mathbb{R}^r \;\rightarrow\; (0,1)^r$

$$\phi_k(u) \;=\; \frac{e^{u_k}}{\sum_{j=1}^{m} e^{u_j}} \tag{4}$$

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

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

- the parameter θ is a linear function of $x \in \mathbb{R}^d$
$$\theta \;=\; \beta^T x \tag{6}$$

- 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

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

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

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

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.]

Hidden layer options

- sigmoidal functions ϕ, $tanh$
- hinge functions RELU $= max(u, 0)$, softplus $= \ln(1 + e^u)$

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 $x \equiv x^{(0)}, y \equiv x^{(L)}$, $m_0 = d, m_L = \dim y$ (typicall 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 $x^{(l)} \in \mathbb{R}^{m_l}$ 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 $[0,1]^d$ *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? $f_0(x) = x$

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

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

Generalization DenseNet

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

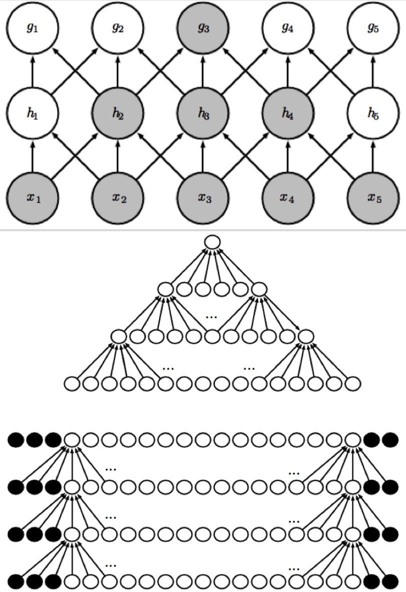
ConvNets – Convolutional Networks

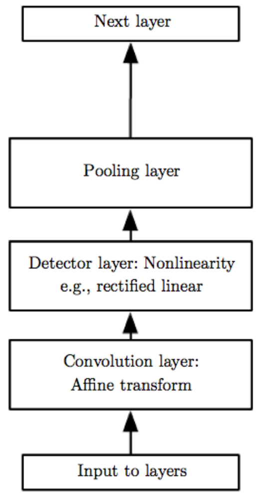
- **discrete convolution** let $f, g : \mathbb{Z} \to \mathbb{R}$
  $\mathbb{Z} =$ all integers
$$(f * g)(t) \;=\; \sum_{i \in \mathbb{Z}} f(t-i) g(i) \tag{11}$$

- convolution as **Toeplitz** matrix vector multiplication

- in ConvNets, $\mathbb{Z}$ is replaced by $1:m$, 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
$$s(t) = \sum_{i=t-s}^{t+s} w_i s(t-i) \tag{12}$$

- Pooling





from `www.deeplearningbook.org` Chapter 9

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

Variations

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

$$\tilde{x} \sim C(\,|x), \quad \min L(x, f(h(\tilde{x}))). \tag{14}$$

- Sparse autoencoder

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

  Ω is regularization that makes h sparse

-

Transformer networks: Why we need attention

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

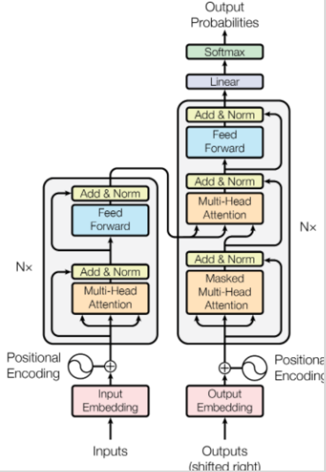
We propose a new simple network architecture, the Transformer,based solely on attention mechanisms, dispensing with recurrence and convolutionsentirely.

Wir schlagen eine neue einfache Netzwerkarchitektur, den Transformer, vor, die ausschließlich auf Aufmerksamkeitsmechanismen basiert und auf Wiederholung und Faltung vollständig verzichtet.

我们提出了一种新的简单的网络架构--Transformer，完全基于注意力机制，摒弃了递归和卷积。

- mapping sequences to sequences (structured prediction)
- both long and short range dependencies
- range depends on input sequence

Basic architecture



- inputs $x_1, x_2, \ldots$, outputs $y_1, y_2, \ldots$ from discrete set (e.g. words in English, Chinese)
- continuous internal representations
- **embedding** modules map input or output space to continuous representations (prelearned)

- recurrence/auto-regression y_t depends on $x_{1:t+k}$ and $y_{1:t-1}$

- **encoder, decoder, encoder-decoder** modules (which use attention)

How to implement attention

- **queries, keys** and **values**
- all **learned**
- Idea: query q matches key k results in selecting the corresponding value v
- q depends on current context, k depends on v
- $q, k \in \mathbb{R}^{d_k}$

- Q, K, V matrices of queries, keys, values

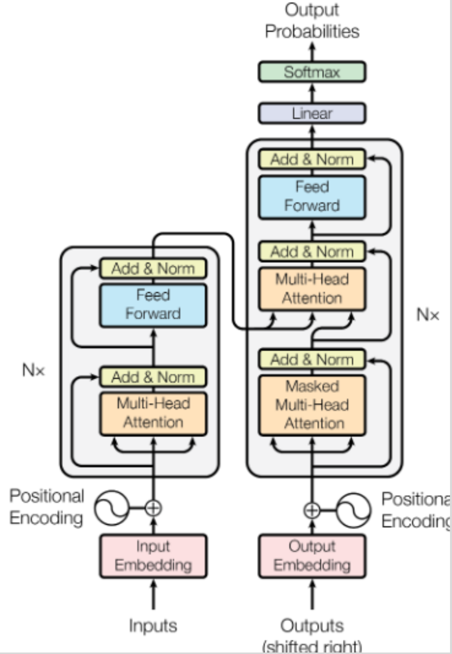
$$A(Q, K) = \text{softmax}(\frac{1}{\sqrt{d_k}} QK^T) \tag{16}$$

- $A_{q:}$ selects value v for the best matching key for each q

Transformer architecture

- D,E,DE modules: each have $N = 6$ layers of Attention $+$ Feed-forward (FFW) networks of same $d = 512$
- FFW, A are ResNets
- FFW is $W_2 \max(W_1 x, 0)$, $W_{1,2}$ with identical rows
- A is **multihead attention**
  - $h = 8$ parallel attention layers, concatenated
- advantages – implements long distance dependencies with fixed (small) number layers, and parallel computations

Attention mechanism in Transformer



- encoder-decoder
  - queries from previous decode layer
  - keys, values from current encoder output
- encoder – self-attention (=previous encoder layer)
- decoder
  - self-attention, **masked**
  - only from outputs before current step