CSE 480/680

2/5/26

Lecture 9

HW3 due 2/4
Sol 1, Sol 2
available
[HW4 : TB posted]

OH tomorrow
9:30, 10:30

L11 corrections

Lecture Notes IV – Neural Networks, Part 1

Marina Meilă
mmp@uwaterloo.ca

With Thanks to Pascal Poupart & Gautam Kamath
Cheriton School of Computer Science
University of Waterloo

February 2, 2026

A little history ✓

The single "neuron" ✓

Two-layer Neural Networks
- Hidden layer options
- Output layer options

Multi-layer neural networks

Training a neural network by backpropagation

Reading HTF Ch.: 11.3 Neural networks, Murphy Ch.: (16.5 neural nets), ~~Bach Ch.: –,~~ 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.

(Artificial) Neural Network (nn) unit

- For each **unit** i
$$y_i \equiv f_i(x) = \phi(\sum_j w_{ij}x_j + w_{i0}) \tag{1}$$
- **Weigth vector** w_i
  - w_{ij} = strength of the link from unit i to input j
  - $w_{ij} = 0$: no link
  - w_{ij} can be positive or negative
  - Sometimes we call the input vector $x = [x_{1:d}]$ input units
- **activation function** $\phi()$
  - must be non-linear (otherwise the unit is a linear transformation)
  - wanted: monotonically increasing, differentiable, gradient non-zero [1]

Notation ϕ is overloaded

- When we talk about nn in general: ϕ is any activation function
- When we do calculations with nn: ϕ is by default the logistic function (unless specified otherwise)
$$\textbf{logistic or sigmoid function} \quad \phi(u) = \frac{1}{1+e^{-u}} \tag{2}$$
- When we do statistics or ML (but not nn): ϕ is the logistic function
- Exercise: compare $f_i(x)$ from (1) with $p(x) = Pr[y = 1 \mid x]$ from logistic regression.

[1] More technically: ϕ can be any continuous, bounded and strictly increasing function on $\mathbb{R}$.

Perceptron

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

- Single layer feed-forward network $w_i \in \mathbb{R}^d$

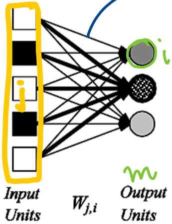
w_i = weights = row vector

$w_i^T x = z_i$

$u_i = \varphi(z_i)$

activation fction

x_j



m

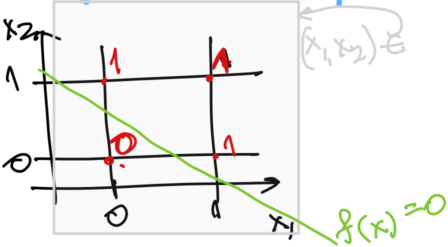
$\begin{bmatrix} u_i \\ \\ \end{bmatrix} = u \in \mathbb{R}^m$

UNIVERSITY OF WATERLOO

$$W = \begin{bmatrix} w_1 \\ w_2 \\ \cdots \\ w_m \end{bmatrix} \in \mathbb{R}^{m \times d}$$

Perceptron weights for OR, AND, XOR

1 unit implementation of OR



$$\varphi = \frac{\operatorname{sgn}(z)+1}{2}$$

$$w_1 = w_2 = 1$$

$$w_0 = -\frac{1}{2}$$

$$f^{true} = x_1 \text{ OR } x_2 \qquad x_{1,2} \in \{0,1\}$$

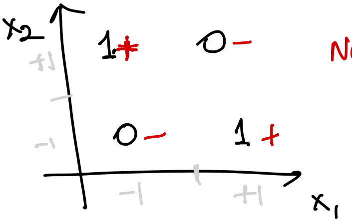
activation

$$f(x_1,x_2) = \varphi\big(\underbrace{w_1x_1 + w_2x_2 + w_0}_{z}\big)$$

Intercept as slope "trick"

$$\tilde{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \quad \tilde{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \Rightarrow z = \tilde{w}^T\tilde{x}$$

Perceptron weights for OR, AND, $\boxed{\text{XOR}}$



NOT linearly separable!

$f^{true} = x_1 \text{ XOR } x_2$

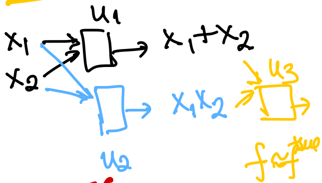
$f(x_1, x_2) = \varphi(w_1 x_1 + w_2 x_2 + w_0)$

monotonic ↗

$\Rightarrow \nexists\, w_{0,1,2}$ so that $f = f^{true}$ on $\{0,1\}^2$

Neural Nets

2) add layers



Ways around:

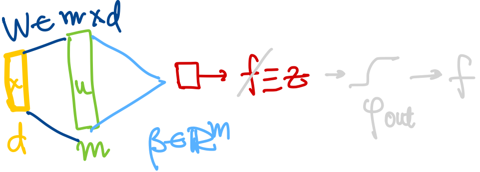
1) add "features"

$x_3 = x_1 x_2$

$\tilde{x} = [1 \;\; x_1 \;\; x_2 \;\; x_1 \cdot x_2]$

$\tilde{w} = [w_0 \;\; w_1 \;\; w_2 \;\; w_3]$ → SVM

Two-layer Neural Networks



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

Inputs	x_j	$j = 1:d$
Hidden layer — Bottom layer[2]	$u_i = \phi(\underbrace{w_i^T x}_{z})$	$i = 1:m,\ w_i \in \mathbb{R}^d$
Top layer	$f = \beta^T u$	$\beta \in \mathbb{R}^m$
Output	f	$\in \mathbb{R}$

z

In other words, the neural network implements the function

$$f(x) = \sum_{i=1}^{m} \beta_i u_i = \sum_{i=1}^{m} \beta_i \phi\Big(\sum_{j=1}^{d} w_{ij} x_j\Big) \in (-\infty, \infty) \tag{3}$$

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

- As we will see shortly, in general, $f(x)$ can also be non-linear

[2] In neural net terminology, each variable u_i 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, u, y variable.

Activation functions for the hidden layer

THEORY φ continuous
φ monotonic ↗
φ bounded
PRACTICAL φ' > 0
φ' computed fast

For the hidden layer, we have to choose

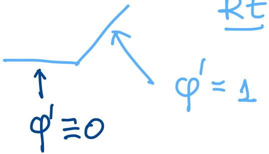
- number of units m
- activation function φ

Common activation functions

- **Functions that approximate a step function**
  - threshold function (or step function) 1 for $u \geq 0$, and 0 otherwise (not used)
  - logistic ϕ
  - hyperbolic tangent *tanh*, arctangent $\tan^{-1}$

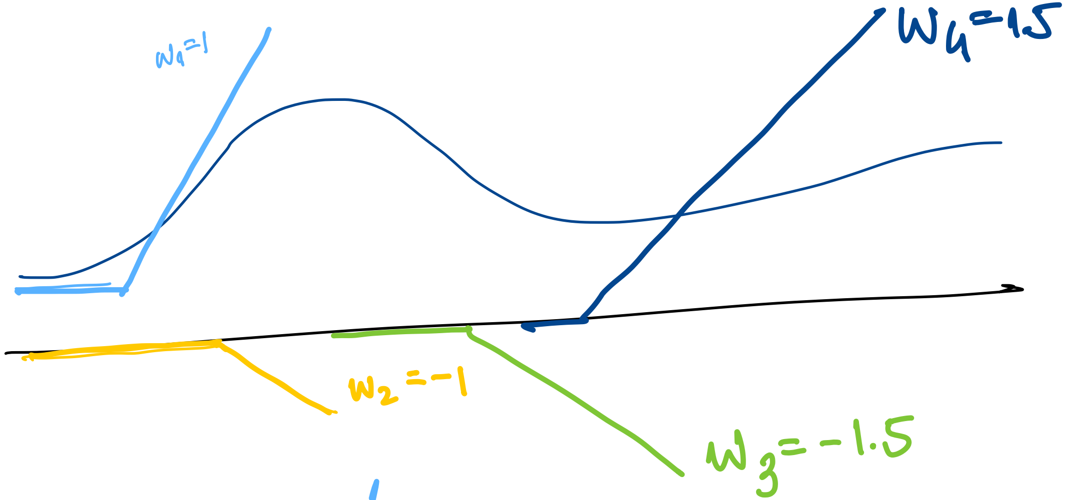
- **Hinge** functions
  - RELU $= max(u, 0)$
  - softplus $= \ln(1 + e^u)$
    in practical implementations, these unbounded functions are bounded at a large value M
  - Why hinge functions? Gradient is 1 or 0 (approximately), faster computation!!, and no saturation

REctified Linear Unit

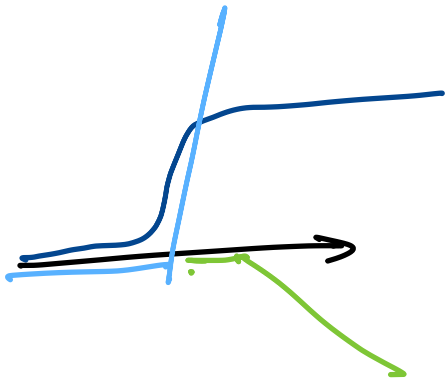


$$\varphi'(u) = \begin{cases} 1 & \text{if } u > 0 \\ 0 & u \leq 0 \end{cases}$$

$\rightarrow u_i \rightarrow$ $w^T u = \sum w_i u_i = \sum_{u_i > 0} w_i$



$w_1=1$
$w_4=1.5$
$w_2=-1$
$w_3=-1.5$



Output layer

$\mathbb{R}^m$ hidden layer $\in \mathbb{R}^m$ $z \in \mathbb{R}^r$ $y \in \{1, 2, \ldots r\}$ Multi → Layer

- Let $\boxed{z = \beta^T u}$ be the linear output of the nn.
- For problems other than regression, it unifies the analysis to apply a non-linear function ϕ_{out} to z. This is why, theoretically but not in practice, we will write $f(z(x)) = \phi_{\text{out}}(z(x))$
- Why? $\phi_{\text{out}}(z)$ matches the loss $\mathcal{L}$, which in turn matches the prediction problem

Prediction problem	Predict	Output layer ϕ_{out}	Range of f
regression	$\hat{y} = z$	linear $\phi_{\text{out}} = z$	$\in \mathbb{R}$
binary classification	$\hat{y} = \text{sgn} z$	logistic $\phi_{\text{out}} = \phi(\beta^T u)$	$\in [0, 1]$
multiway classification	$\hat{y} = \underset{1:r}{\text{argmax}}\, z_k$	softmax $\phi_{\text{out}\,k} = \phi_k(\beta_k^T u)$	$\in [0, 1]^r$

USE n.n. for prediction TRAINING

- Regression: **linear** layer as in (3) $f = \sum_i \beta_i u_i$
- Classification (binary): **logistic** layer $f(x) \in [0, 1]$ is interpreted as the probability of the $+$ class.

$$f(x) = \phi\left(\sum_{j=1}^{m} \beta_i u_i\right) = \phi\left(\sum_{i=1}^{m} \beta_j \phi(\sum_j w_{ij} x_j)\right) \tag{4}$$

- Multiway classification with r classes
  - Output is vector of r functions $f_1, \ldots f_r$
  - f_k is the probability of $y = k$
  - (sometimes f_k can be a "confidence")
  - This is done with a **softmax** layer (next page)

OPTIONAL - 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
$$P_\theta(y) \;=\; e^{\theta y - \psi(\theta)} \tag{8}$$
- the parameter θ is a linear function of $x \in \mathbb{R}^d$
$$\theta \;=\; \beta^T x \tag{9}$$
- 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

$$l(\beta) \;=\; \ln P_\theta(y|x) \;=\; \theta y - \psi(\theta) \quad \text{where } \theta = \beta^T x \tag{10}$$

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

$$\nabla_\beta l \;=\; \nabla_\theta l \nabla_\beta(\beta^T x) \;=\; (y - \mu)x \tag{11}$$

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

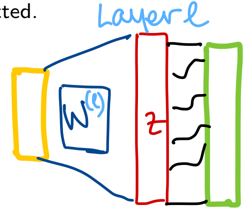
The construction can be generalized recursively to arbitrary numbers of layers.
Each layer is a linear combination of the outputs from a previous layer (a multivariate operation), followed by a non-linear transformation via the logistic function ϕ. Let $x \equiv x^{(0)}, y \equiv x^{(L)}, m_0 = d, m_L = \dim y$ (typically 1) and define the recursion:

$$x_j^{(l)} = \phi\left(\underbrace{(w_j^{(l)})^T x^{(l-l)}}_{z^{(l)}}\right), \text{ for } j = 1:m_l,\ l = 1:L \tag{12}$$

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.

$l = 1:L$

$l=0$
$x^{(0)} = x$
$m_0 = d$
input layer



$x \in \mathbb{R}^d$ → $x^{(l-1)} \in \mathbb{R}^{m_{l-1}}$

$l=L$
$m_L = 1$ for example
$\varphi_{out}(z^{(L)}) = z^{(L)}$ for ex

$u \in \mathbb{R}^m$ → $x^{(l)} \in \mathbb{R}^{m_l}$

$z = W^T x$ → $z^{(l)} \in \mathbb{R}^{m_l}$

$u = \varphi(z) \Leftrightarrow u_i = \varphi(z_i) \quad i = 1:m$

Are multiple layers necessary?

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

- A theoretical result

Theorem (Cybenko, $\approx$1986)

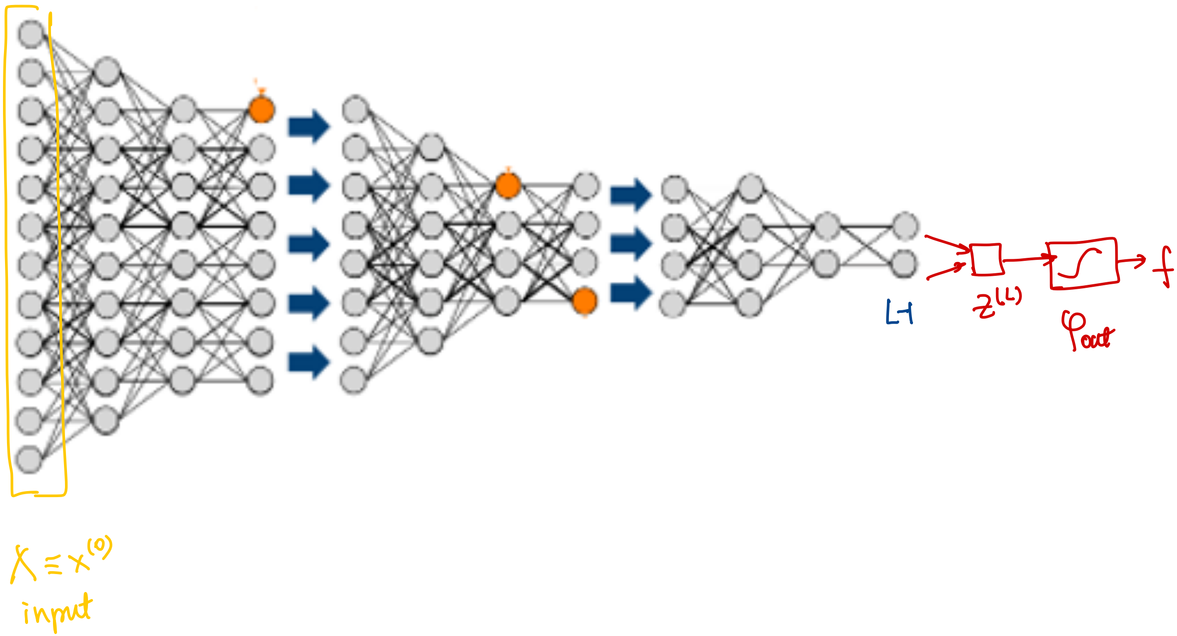
Any continuous function from $[0,1]^d$ to $\mathbb{R}$ can be approximated arbitrarily closely by a linear output, two layer neural network defined in (3) *with a sufficiently large number of hidden units m.*

- 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**



Parameters

1 unit $\quad W = \{w\} \quad d+1$ params

1 hidden layer $\quad W = \{W \in \mathbb{R}^{m\times d}, b \in \mathbb{R}^{m}\} \quad m + md$ param

L layers $\quad W = \{W^{(1)}, \dots W^{(L)}\} \quad \sum_{\ell=1}^{L} m^{(\ell-1)} m^{(\ell)}$ params