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.

A little history

First epoch – can computers mimic the brain?

- 40's-50's the first mathematical models of the neuron (McCullaugh & Pitts, Hebb)
- '58 the perceptron (Rosenblatt)
- First winter '69 "Perceptrons" (Minsky and Papert) – they can only classify linearly separable data (and von Neumann computers steal the show)

Revival in the 80's-'95 – let's use layers!

- '82 Hopfield associative net (contents adressable memory)
- '84 Boltzmann Machine (Ackley, Hinton, Sejnowski)
- autoencoders
- '86 backpropagation (Sejnowski, Rosenberg)
- Reinforcement learning (Shannon, Samuel, Barton, Sutton, Watkins, Tesauro)
- Second winter cca '95 Statistical learning takes the stage, especially SVM

Current epoch cca 2005 – more data, more layers

- Deep learning
- Generative models
- Attention
- LLMs

Brains vs. Computers

Computer (von Neumann)

- Electrical binary signals directed by gates
- Wiring is fixed
- Sequential and parallel computation
- Memory is retrieved by address
- **Fragile** (if a gate stops working, computer crashes). Also, "no friction" – a single instruction can stop the entire machine/program.

- **Brain**
- Electrical signals, units=neurons
- Wiring is dynamic, changes with brain development, experiences, learning
- Parallel (and some) sequential computation
- Memory is distributed
- **Robust** (when neuron/region dies, brain rewires itself to compensate). No On/Off master switch

Brains vs. Computers

Computer (von Neumann)

- Electrical binary signals directed by gates
- Wiring is fixed
- Sequential and parallel computation
- Memory is retrieved by address
- **Fragile** (if a gate stops working, computer crashes). Also, "no friction" – a single instruction can stop the entire machine/program.

- **Brain**
- Electrical signals, units=neurons
- Wiring is dynamic, changes with brain development, experiences, learning
- Parallel (and some) sequential computation
- Memory is distributed
- **Robust** (when neuron/region dies, brain rewires itself to compensate). No On/Off master switch

- **Neural network**
- Signals are numbers passed between units
- Network structure is fixed (and dense) but the learned weights allow "rewiring" during training
- Parallel (in layer) and sequential (feed forward/backward) computation
- Memory is distributed (in the weights)
- **Redundant/robust** (no single neuron can influence the output much

(Artificial) Neural Network (nn) unit

- For each **unit** i

$$y_i \equiv f_i(x) = \phi(\sum_j w_{ij} x_j + w_{i0}) \quad (1)$$

- **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]

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

(Artificial) Neural Network (nn) unit

- For each **unit** i

$$y_i \equiv f_i(x) = \phi(\sum_j w_{ij} x_j + w_{i0}) \quad (1)$$

- **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)

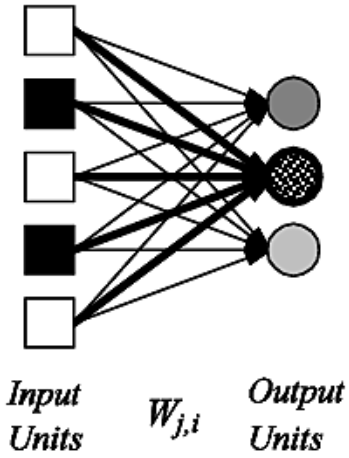
$$\textbf{logistic or sigmoid function} \quad \phi(u) = \frac{1}{1+e^{-u}} \quad (2)$$

- 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

- Single layer feed-forward network



Perceptron weights for OR, AND, XOR

Two-layer Neural Networks

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

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

  In other words, the neural network implements the function

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

  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

For the hidden layer, we have to choose

- number of units m
- activation function

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

Output 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 $\phi_{\text{out}\,k} = \phi_k(\beta_k^T u)$	$\in [0,1]^r$

- 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.

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

- 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)

The softmax function

- logistic $\phi(z) = \frac{e^z}{1+e^z}$
  - represents the probability that $y = 1$ in binary classification
  - in a nn, $f(x) = \phi(\beta^T u)$, with $u \in \mathbb{R}^m$ the activations of the hidden layer
- The softmax function generalizes the logistic to r classes
  $\phi(z) : \mathbb{R}^r \rightarrow (0,1)^r$

$$\phi_k(z) = \frac{e^{z_k}}{\sum_{j=1}^m e^{z_j}}, \text{ for } k = 1 : r \tag{5}$$

$$\phi(z) = [\phi_1(z) \ldots \phi_r(z)] \text{(overload of } \phi) \tag{6}$$

- Properties of softmax
  - $\sum_{j=1}^m \phi_j(z) = 1$ for all z
  - for $z_k \gg z_j$, $\phi_k(z) \rightarrow 1$.
  - derivatives $\frac{\partial \phi_j}{\partial z_k} = \phi_k \delta_{jk} - \phi_j \phi_k$
- in a nn, with output activations $u \in \mathbb{R}^m$, we train r weight vectors $\beta_{1:r}$, and

$$f_k(x) = \phi_k(\beta_k^T u(x)), \quad \text{for } k = 1 : r \tag{7}$$

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.

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* (3) *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**