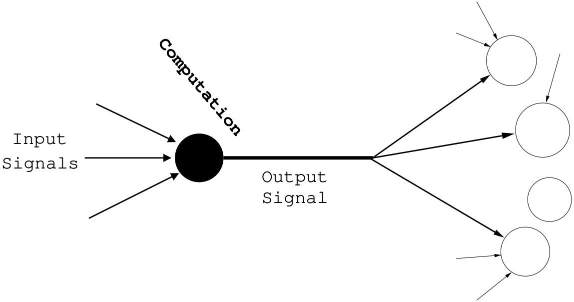
Basic definitions

Aapo Hyvärinen
for the NNDL course, 14 March 2025

- Neuron model
- Neural network as nonlinear function approximator
- Universal approximation capability
- Network architecture
- Statistical (probabilistic) objective function
- Learning as optimization of objective
- Deep learning system as:
  Statistical objective + network architecture + optimization algorithm

Literature: Prince's book chapters 2–4 (but it's only approximately the same)

Simple neuron model inspired by neuroscience



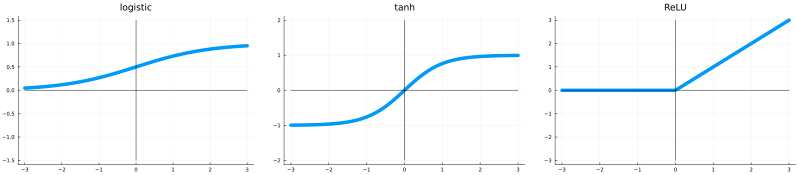
- Several inputs x_i collected, one scalar output y transmitted
- Rough model

$$y = \psi(\sum_i w_i x_i + b) \qquad (1)$$

  - w_i are the *weight* parameters (often collected in vector $\mathbf{w}$)
  - b is a *bias* parameter
  - ψ is *nonlinearity* or *activation function*, $\mathbb{R} \to \mathbb{R}$

Typical choices of activation function

$$y = \psi(\sum_i w_i x_i + b) \tag{2}$$



- logistic:

$$\psi(x) = \frac{1}{1 + \exp(-x)} \tag{3}$$

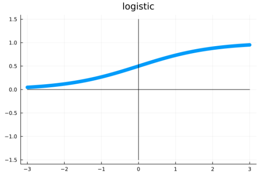
  soft thresholding, "is the linear sum above $-b$?"
- tanh: like logistic but rescaled to be odd-symmetric
- ReLU ("rectified linear unit"): $\psi(x) = \max(x, 0)$
  like thresholding but preserves the values above threshold
- Could be linear as well, $\psi(x) = x$, but rarely used

Interpretation of nonlinear activation function

- Consider two *non-negative* inputs, equal weights, logistic ψ:

$$y = \psi(x_1 + x_2 + b) \qquad (4)$$

- Logistic function is a "soft" thresholding at zero.



- We can have kind of AND operation with strongly negative b
  - Both x_1 and x_2 must be "active" (clearly > 0) to cross the threshold
- We can have kind of OR operation by mildly negative b
  - Just one active input, x_1 or x_2, is enough to cross the threshold
- Near zero (using very small weights) σ is close to linear: It can model linear functions if necessary
- But cannot do XOR... one reason why we need many layers.

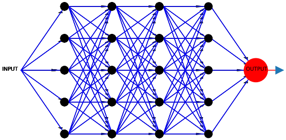
Considering bias as a hypothetical input

- For mathematical simplicity, define a hypothetical input $x_0 \equiv 1$.
- Denote bias as $w_0 := b$
- Then, we can denote

$$\sum_{i=1}^{n} w_i x_i + b = \sum_{i=0}^{n} w_i x_i =: \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} \tag{5}$$

  with a new, augmented $\tilde{\mathbf{w}} = [b, \mathbf{w}]$, $\tilde{\mathbf{x}} = [x_0, \ldots, x_n]$,
- Simplifies equations, assumed quite often
  - Can lead to some inconsistencies, but nobody usually cares....
- Tilde in notation above dropped

Neural network (most basic case)

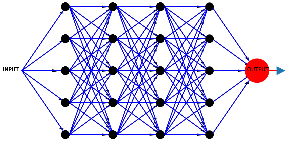


- Put many neurons (index: j) in parallel to create a *layer*
- Put K layers one after another, layer index k
  - $k = 0$ (or $k = 1$) is "input" layer; $k = K$ is "output" layer
  - Other k are *hidden* layers; there are perhaps $K - 1$ of them
  - Note that this terminology and indices is a bit variable
- Preceding layer $k - 1$ feeds into the next layer k
- Denote by $\mathbf{w}_{jk}$ weight vector of j-th neuron in k-th layer
- We define output of neuron j in k-th layer as

$$y_{jk} = \psi(\sum_i \mathbf{w}^i_{jk} y_{i,k-1}) \qquad (6)$$

where $y_{i,0}$, usually denoted by x_i, is original input to network.

Neural network in matrix notation



- Denote the matrix of weights in k-th layer as $\mathbf{W}_k$.
- Notational convention: $\psi(\mathbf{y}) = (\psi(y_1), \ldots, \psi(y_n))$
- We can write the vector of all k-th layer outputs as

$$\mathbf{y}_k = \psi(\mathbf{W}_k \mathbf{y}_{k-1}) \tag{7}$$

- The whole network becomes

$$\mathbf{y}_K = \psi(\mathbf{W}_K \psi(\mathbf{W}_{K-1} \psi(\mathbf{W}_{K-2} \ldots \psi(\mathbf{W}_2 \psi(\mathbf{W}_1 \mathbf{x}))))) \tag{8}$$

- Dimension of $\mathbf{y}_k$ can change arbitrarily (unlike in figure above)

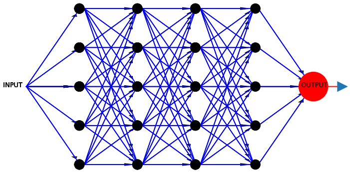
Universal approximation capability

- Fundamental theoretical results:
  - For certain nonlinearities (e.g. logistic, tanh, ReLU),
  - and in the limit of an infinite number of neurons (even with a single hidden layer)
  - $\Rightarrow$ a neural network can approximate any function!
- Even empirically, neural network seem to be the best known way of approximating arbitrary nonlinear functions in *high-dimensional* spaces
- Without nonlinearity, does not work. In fact, adding linear layers is useless:

$$\mathbf{y}_K = \mathbf{W}_K \mathbf{W}_{k-1} \dots \mathbf{W}_1 \mathbf{x} = \mathbf{M}\mathbf{x} \tag{9}$$

  Equivalent to a single-layer network.
  $\rightarrow$ Main reason for nonlinearity!
- *Deep* learning based on empirical claim that NN's with many (even 100's, 1000's) of layers are particularly good
  - as opposed to having 1000,000's of units in a couple of layers ("broad" but "shallow" network)

Neural network architectures



- Architecture means the detailed specification of the NN:
  - Number of layers in the network
  - Number of neurons in each layer
  - Activation function (not necessarily the same everywhere)
  - How the neurons are connected to each other, etc. etc.
- One fundamental distinction:
  Feedforward networks vs. recurrent networks (i.e. feedback)
- Many many proposals:
  Convolutional NN, residual NN, transformers etc.
- Essentially, different ways of approximating nonlinearities

Abstracting away the neural network

- NN is *just a method for approximating a function*
- Often, we abstract away the neurons and architecture; just consider the NN as a nonlinear function

$$\mathbf{y} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x}) \tag{10}$$

  - $\mathbf{x}$ is the n-dimensional vector of inputs x_i
  - $\mathbf{y}$ is the m-dimensional vector of outputs.
  - $\mathbf{g}$ is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$.
- Importantly, the function $\mathbf{g}$ depends on the weight vectors (incl. biases), collected in the vector $\boldsymbol{\theta}$.
- But *what* function should be approximated?
  - I.e. how to *set the weights*?
  - This is not answered by choosing the architecture

Learning task, learning principle, and objective function

- So, the whole NN is summarized as a function: $\mathbf{y} = \mathbf{g}_{\boldsymbol{\theta}}(\mathbf{x})$
- Crucial question:
  How to learn the $\boldsymbol{\theta}$, *given a learning task* (e.g. classification)
- Usually, this is by optimization of an *objective function* J

$$\max_{\boldsymbol{\theta}} J(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) \tag{11}$$

  given by some theoretical framework ("learning principle")
- Many terms used for this:
  objective function ≈ criterion ≈ loss function ≈ cost function
- Some objectives are the same as in the linear setting, just replace linear function by NN
  - Nonlinear regression possible by least squares objective
  - Classification by cross-entropy (as in logistic regression)
- Often based on probabilistic/statistical theory
  - independent of function approximation method
  - most of deep learning theory has little to do with NN's ! :=)

Optimization methods are crucial for deep learning

- Once objective function / loss (e.g. least-squares) is given, we need an algorithm to optimize it
- Optimization in real spaces, $\boldsymbol{\theta} \in \mathbb{R}^n$ with possibly huge n
- Classic topic independently of NNs: a lot of methods exist
- Gradient methods dominant in deep learning
- But we could use any optimization method, independently of objective function
  - E.g. evolutionary strategies

Summary of deep learning construction

- To solve a given task, we typically need three ingredients:
  1. Objective function J, based on a learning principle
     - Typically from statistical theory / probabilistic modelling
  2. NN as function approximator $(\mathbf{g})$ with specified architecture
     - Deep learning: some architecture with "many" layers
  3. Optimization algorithm applied on J
- For example: linear regression, the most basic method
  1. Learning principle: typically least-squares
  2. Function approximator: linear, architecture trivial
  3. Optimization method: e.g. solving a system of linear equations
- In principle, a combinatorial number of ensuing methods
- A lot of the theory not specific to deep learning:
  - Objective functions compatible with other function approximators
  - Note: A neural network in itself cannot learn anything !

About this course

- We need elements of different mathematical theories
  - Statistics (or probabilistic modelling) to formalize task and find objective function
  - Optimization, in particular in real spaces, to optimize it
  - Heuristics for choosing NN architecture; little theory exists :(
- In our course, emphasis is on
  - formalizing different tasks, typically in a probabilistic paradigm
  - deriving objective functions
- Optimization given some space as well
- Architectures given less emphasis
- All topics treated today will be expanded later