4. Neural Networks

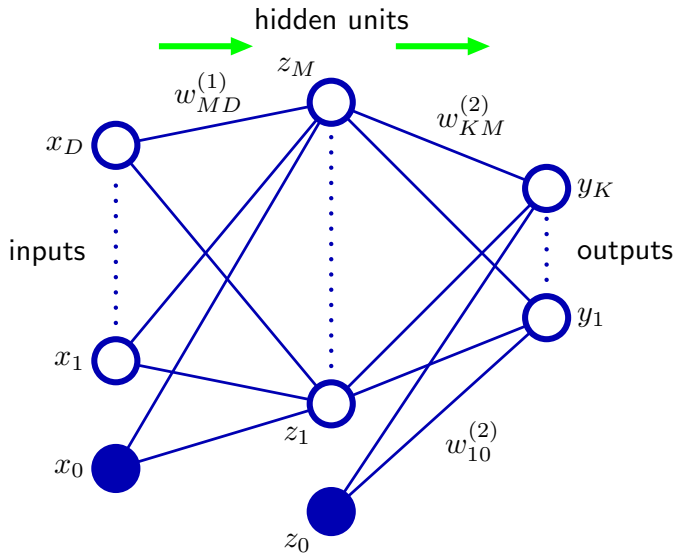
Robert Legenstein

Institute for Theoretical Computer Science

November, 2014

Overview

4.1 Neural Networks

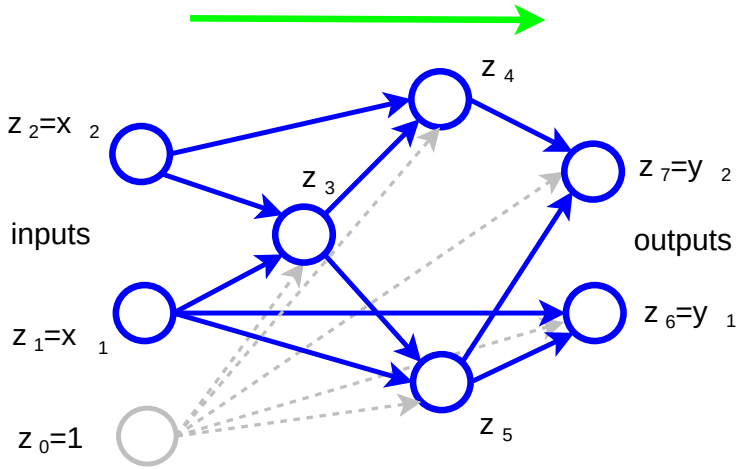


Linear models for regression and classification were of the form

$$y(\mathbf{x}) = h\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right)$$

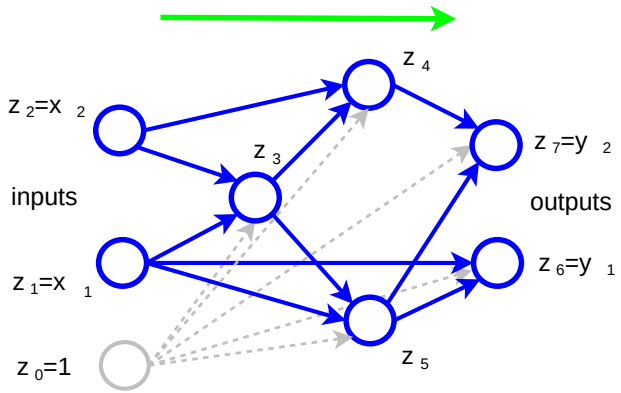
where the *activation function* $h(\cdot)$ is some nonlinearity.
One can view neural networks as models with adaptable basis functions $\phi_j(\mathbf{x})$.

General Network Description



A network can be described by

- a directed graph $G = (V, E)$ with
  - nodes (neurons and inputs) V and
  - edges (connections) E.
- a weight w_{ji} for each edge $(i, j) \in E$,
- an activation function h_i for each non-input node (differntiable), and
- a list of K output nodes $OUT = \langle out_1, \ldots, out_K \rangle$.



The output of node i is given by

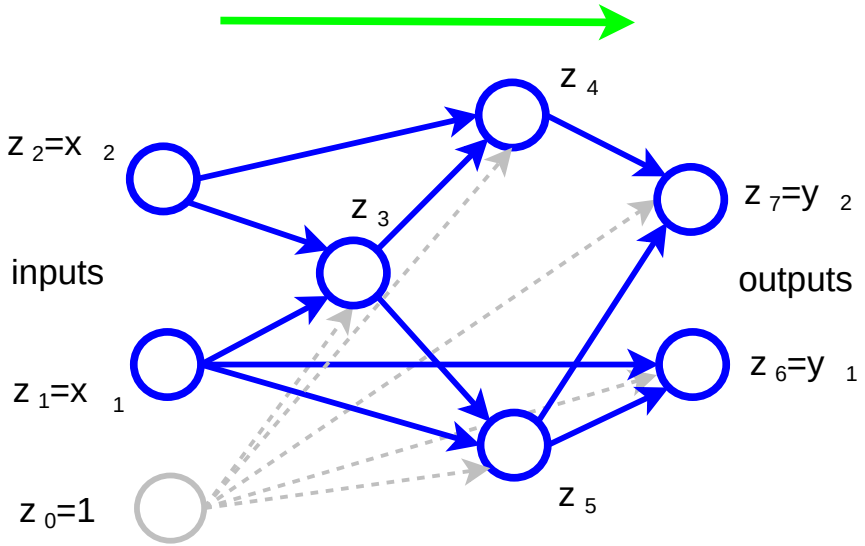
$$z_i = \begin{cases} x_i & , \quad \text{if } i \in \{0, \ldots, D\} \\ h_i(a_i) & , \quad \text{otherwise} \end{cases}$$

with **activations** $a_i = \sum_{j \in pre(i)} w_{ij} z_j$ and mappings

pre: $pre(i) = \{j | (j,i) \in E\}$ is the set of neurons which *connect to* neuron i.

post: $post(i) = \{j | (i,j) \in E\}$ is the set of neurons which i *connects to*.

Then, the kth output of the network is given by $y_k = z_{out_k}$.



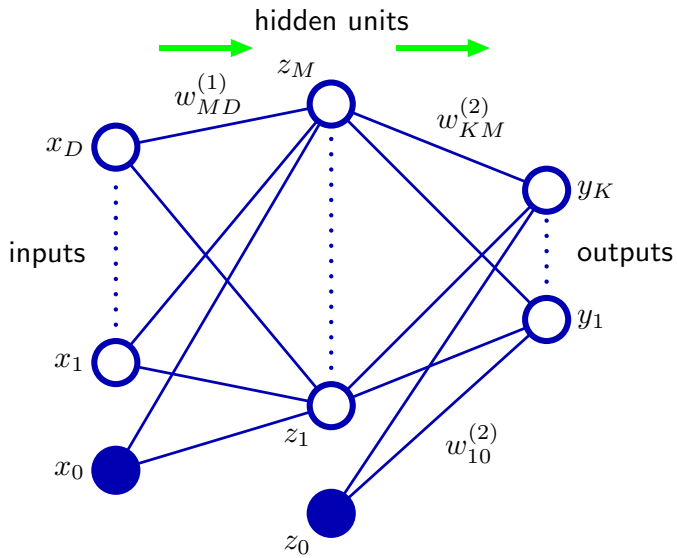
Any network can be described in this way.

The network function $\mathbf{y}(\mathbf{x}, \mathbf{w})$ can be defined recursively.

We consider only network graphs without loops $\longrightarrow$ **Feedforward Networks**.

Such networks are often called **Multilayer Perceptrons** although the neural units are usually **not** perceptrons.

Layered Networks



In a **layered network**, the nodes can be divided into layers, such that

- Input nodes are in layer 0 (input layer).
- A node in layer i has outgoing edges only to nodes in layer $i+1$ for all i.

We count the number of layers of a network excluding the input layer.

Hidden units: Neurons which are neither input nor output units.

Hidden layers: Layers containing hidden units (for layered network).

The choice of *output* activation functions depends on the nature of the problem:

- Linear for regression problems
- Sigmoid for (possibly multiple) binary classification problems.
- Softmax for multiclass classification problems.

$$\text{linear:} \qquad h_{out_k}(a) = a$$

$$\text{logsig:} \qquad h_{out_k}(a) = \frac{1}{1 + e^{-a}}$$

$$\text{softmax:} \qquad h_{out_k}(a_1, \ldots, a_K) = \frac{e^{a_k}}{\sum_j e^{a_j}}$$

For hidden units, one uses typically sigmoidal activation functions, in particular often the hyperbolic tangens:

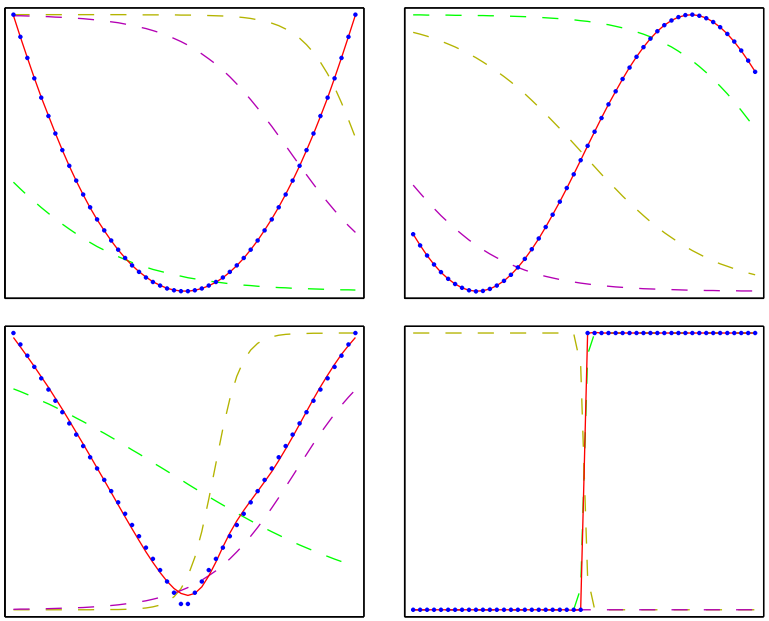
$$\text{tanh: } h_i(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$$

Approximation properties

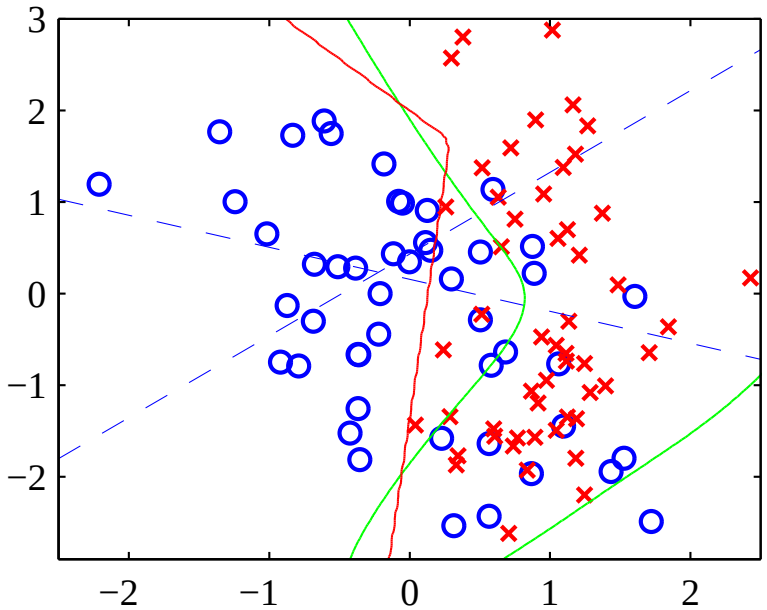
Neural networks define a very general class of functions.

Neural Networks are universal approximators

A two-layer neural network with linear output unit and sigmoidal hidden units can approximate any continuous function on a compact input domain to arbitrary accuracy provided a sufficiently large number of hidden units.



Furthermore, neural networks can define arbitrarily complex decision regions.



4.2 Network Training

The error functions for different types of problems can be derived from maximum likelihood and assumptions about data distributions.
The corresponding activation functions arise naturally such that the network output can be interpreted as the posterior distribution.

K-dimensional regression: (activation function is the identity)

$$\begin{aligned} p(\mathbf{t}|\mathbf{x},\mathbf{w}) &= \mathcal{N}(\mathbf{t}|\mathbf{y}(\mathbf{x},\mathbf{w}),\beta^{-1}I) \\ E(\mathbf{w}) &= \frac{1}{2}\sum_n ||\mathbf{y}(\mathbf{x}_n,\mathbf{w})-\mathbf{t}_n||^2 \\ &= \frac{1}{2}\sum_n\sum_k (y_k(\mathbf{x}_n,\mathbf{w})-t_{nk})^2 \end{aligned}$$

t_{nk} ... k-th component of the target vector for the n-th training example.
Compare to the 1D case:

$$E(\mathbf{w}) = \frac{1}{2}\sum_n (y(\mathbf{x}_n,\mathbf{w})-t_n)^2$$

Single binary classification: (activation function is logsig)

$$\begin{aligned} p(t|\mathbf{x},\mathbf{w}) &= y(\mathbf{x},\mathbf{w})^t[1-y(\mathbf{x},\mathbf{w})]^{1-t} \\ E(\mathbf{w}) &= -\sum_n t_n \ln y(\mathbf{x}_n) + (1-t_n)\ln(1-y(\mathbf{x}_n)) \end{aligned}$$

K-class classification: (activation function is softmax $\frac{\exp(a_k(\mathbf{x},\mathbf{w}))}{\sum_j \exp(a_j(\mathbf{x},\mathbf{w}))}$)

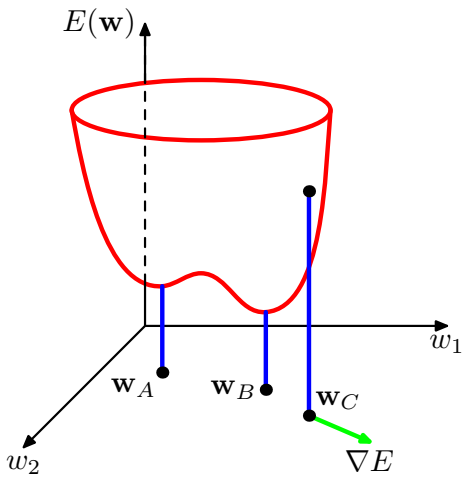
$$\begin{aligned} p(\mathbf{t}|\mathbf{x},\mathbf{w}) &= \prod_k y_k(\mathbf{x},\mathbf{w})^{t_k} \\ E(\mathbf{w}) &= -\sum_n \sum_k t_{nk} \ln y_k(\mathbf{x}_n) \end{aligned}$$

In any case, we have

$$\frac{\partial E}{\partial a_{out_k}} = \sum_n (y_k(\mathbf{x}_n) - t_{nk}),$$

where $a_i = \sum_{j \in pre(i)} w_{ij} z_j$.

Parameter Optimization



The smallest value of E occurs at a point where $\nabla E(\mathbf{w}) = 0$.
$\rightarrow$ **Stationary points:**

- minima, maxima, saddle points

We classify minimia into

global minima: Points $\mathbf{w}^*$ with $E(\mathbf{w}^*) \leq E(\mathbf{w})$ for all $\mathbf{w}$.

local minima: Minima such that points with smaller error exist.

Our task: Find a good local minimum.

There is no hope to find an analytical solution to $\nabla E = 0$.
We therefore resort to iterative numerical procedures of the form

- choose $\mathbf{w}^{(0)}$,
- $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)}$

There are many algorithms to do that, and many of them use gradient information.

Gradient Descent Optimization

Simplest approach: Take a small step in the direction of the negative gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta(\tau)\nabla E(\mathbf{w}^{(\tau)}),$$

where $\eta(\tau) > 0$ is a small *learning rate*.
Note that the evaluation of the gradient needs the processing of the whole training set $\longrightarrow$ *batch mode*.

More efficient batch methods:

- conjugate gradient
- quasi-Newton methods.

Alternative:

- Stochastic Gradient Descent: converges faster and can escape local minima.

4.3 Error Backpropagation

Error backpropagation is an efficient technique to evaluate the gradient of the error function.
This is achieved by local message passing through the network:

1. $\mathbf{x} \rightarrow \mathbf{y}$: forward propagate information to compute the activations and outputs of all gates.
2. $(\mathbf{y}, \mathbf{t}) \rightarrow \nabla E$: Backward propagate the errors. This step is known as *error backpropagation* or simply *backprop*.

The nomenclature is somewhat confusing in the literature. We use the term "backprop" specifically to describe the evaluations of the derivatives. A second step (not part of backprop) is then to compute the weight adjustments.

1. Apply $\mathbf{x}_n$, forward propagate and compute activations and outputs of all units using $a_j = \sum_{i \in pre(j)} w_{ji} z_i$ and $z_j = h(a_j)$.
2. Evaluate $\delta_{\text{out}_k} = y_k(\mathbf{x}_n) - t_{kn}$ for all output units.
3. Backpropagate the δ's using

$$\delta_j = h'(a_j) \sum_{k \in post(j)} w_{kj} \delta_k$$

4. Evaluate the derivatives using

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

For batch mode, we can sum the derivatives $\frac{\partial E}{\partial w_{ji}} = \sum_{n=1}^{N} \frac{\partial E_n}{\partial w_{ji}}$.
For simple batch gradient descent, update the weights $w_{ji}^{(\tau+1)} = w_{ji}^{(\tau)} - \eta(\tau) \frac{\partial E}{\partial w_{ji}}$.