Neuronale Netze (SS 2002), 6.5.

Feedforward neural networks:

- A **neuron** is characterized by
  - the input dimension n,
  - weights $w_i \in \mathbb{R}$, $i = 1, \ldots, n$, bias $\theta \in \mathbb{R}$,
  - activation function $f : \mathbb{R} \to \mathbb{R}$ (e.g. H, sgd$(x) = (1 + \exp(-x))^{-1}$),

  Given $\vec{x} \in \mathbb{R}^n$, it computes the **nettoinput** $net = \sum_i w_i x_i - \theta$ and the **output** $o = f(net)$.

- A **neural network** is characterized by
  - N set of neurons,
  - a connection structure of the neurons, denoted by $i \to j$,
  - a weight w_{ij} for each connection $i \to j$,
  - a bias θ_i for each neuron,
  - an activation function f_i for each neuron, (often $f_i =$ sgd),
  - I the input neurons, O the output neurons

  A **neural architecture** does not specify the weights and biases, only the connection structure.

- A **feedforward** network has an acyclic graph $\to$, I are the neurons without predecessor, O the neurons without successor.

  It computes iteratively $f : \mathbb{R}^{|I|} \to \mathbb{R}^{|O|}$, given some input $\vec{x}$:

  $$net_i = o_i = x_i \text{ for input neurons } i$$

  $$net_i = \sum_{j \to i} w_{ji} o_j - \theta_i, o_i = f_i(net_i) \text{ otherwise}$$

  The output of the function can be found at the output neurons.

- **layered** network: the neurons decompose into groups s.t. connections can only be found to following layers, we find input layer, output layer, one, several, or none hidden layer.

  fully connected multilayer network: all connections between consecutive layers.

  fully connected multilayer network with shortcuts: all connections to all following layers.

  Short notation of the architecture via the number of neurons in the layers: $(1:2:3:2)$ or $(1:2:3:2)$s

- A **training set** is a set of desired input/output pairs:
  $\{(\vec{x}^p, \vec{y}^p) \in \mathbb{R}^I \times \mathbb{R}^O \mid p = 1, \ldots, P\}$

- ' Training' means, given an architecture and a training set, find weights and biases such that the inputs $\vec{x}^p$ are approximately mapped to the corresponding outputs $\vec{y}^p$, i.e.

  $$o_i(\vec{x}^p) \approx y_i^p \quad \forall p, \text{output neurons } i$$

  ... we'll see later how to determine the architecture and all this stuff ...

- **Quadratic error** $E = 0.5 \cdot \sum_p \sum_{\text{outputs i}} (o_i(\vec{x}^p) - y_i^p)^2$

- Training of a network with differentiable functions (e.g. sgd)
  minimize the function $\vec{w}, \vec{\theta} \mapsto E$ with a **gradient descent** method, i.e.:

  init w_{ij}, θ_i with small random numbers
  repeat
  $(\vec{w}, \vec{\theta}) := (\vec{w}, \vec{\theta}) - \eta \cdot gradientE$

  where $\eta > 0$ is the learning rate,
  $gradientE$ collects the partial derivatives of E w.r.t. the weights, biases,
  it constitutes the direction of the steepest descent!

- Thereby: partial derivative $=$ derivative with respect to one specified argument

  $$\frac{\partial f(x,y,z)}{\partial x} = f'_{y,z}(x)$$

where in $f_{y,z}$ the values y and z are regarded as constants.

- There exists unfortunately one ugly formula which we'll use:
$$\frac{\partial f(g_1(\vec{x}),\ldots,g_n(\vec{x}))}{\partial x_i} =$$
$$\frac{\partial f(g_1(\vec{x}),\ldots,g_n(\vec{x}))}{\partial g_1(\vec{x})}\frac{\partial g_1(\vec{x})}{\partial x_i} + \ldots + \frac{\partial f(g_1(\vec{x}),\ldots,g_n(\vec{x}))}{\partial g_n(\vec{x})}\frac{\partial g_n(\vec{x})}{\partial x_i}$$