MLP – Notation

- X set of input neurons
- Y set of output neurons
- Z set of all neurons (tedy $X, Y \subseteq Z$)
- individual neurons are denoted by indices, e.g., i, j.
- ξ_j is the inner potential of the neuron j when the computation is finished.
- y_j is the output value of the neuron j when the computation is finished.

  (we formally assume $y_0 = 1$)
- w_{ji} is the weight of the arc **from** the neuron i **to** the neuron j.
- $j_{\leftarrow}$ is the set of all neurons from which there are edges to j
  (i.e. $j_{\leftarrow}$ is the layer directly below j)
- $j^{\rightarrow}$ is the set of all neurons with edges from j.
  (i.e. $j^{\rightarrow}$ is the layer directly above j)

MLP – Notation

- Inner potential of a neuron j:

$$\xi_j = \sum_{i \in j_{\leftarrow}} w_{ji} y_i$$

- A value of a non-input neuron $j \in Z \setminus X$ when the computation is finished is

$$y_j = \sigma_j(\xi_j)$$

  Here σ_j is an activation function of the neuron j.

  (y_j is determined by weights $\vec{w}$ and a given input $\vec{x}$, so it's sometimes written as $y_j[\vec{w}](\vec{x})$)

- Fixing weights of all neurons, the network computes a function $F[\vec{w}] : \mathbb{R}^{|X|} \rightarrow \mathbb{R}^{|Y|}$ as follows: Assign values of a given vector $\vec{x} \in \mathbb{R}^{|X|}$ to the input neurons, evaluate the network, then $F[\vec{w}](\vec{x})$ is the vector of values of the output neurons.

  Here, we implicitly assume a fixed ordering on input and output vectors.

MLP – Learning

- Given a set D of training examples:

$$D = \left\{ \left(\vec{x}_k, \vec{d}_k\right) \;\middle|\; k = 1, \ldots, p \right\}$$

  Here $\vec{x}_k \in \mathbb{R}^{|X|}$ and $\vec{d}_k \in \mathbb{R}^{|Y|}$. We write d_{kj} to denote the value in $\vec{d}_k$ corresponding to the output neuron j.

- **Error Function:** $E(\vec{w})$ where $\vec{w}$ is a vector of all weights in the network. The choice of E depends on the solved task (classification vs regression etc.).

  Example (Squared error):

$$E(\vec{w}) = \sum_{k=1}^{p} E_k(\vec{w})$$

  where

$$E_k(\vec{w}) = \frac{1}{2} \sum_{j \in Y} \left(y_j[\vec{w}](\vec{x}_k) - d_{kj}\right)^2$$

MLP – Batch Gradient Descent

The algorithm computes a sequence of weights $\vec{w}^{(0)}, \vec{w}^{(1)}, \ldots$.

- weights $\vec{w}^{(0)}$ are initialized randomly close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$) is $\vec{w}^{(t+1)}$ computed as follows:

$$w_{ji}^{(t+1)} = w_{ji}^{(t)} + \Delta w_{ji}^{(t)}$$

  where

$$\Delta w_{ji}^{(t)} = -\varepsilon(t) \cdot \frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$$

  is the weight change w_{ji} and $0 < \varepsilon(t) \leq 1$ is the learning rate in the step $t+1$.

Note that $\frac{\partial E}{\partial w_{ji}}(\vec{w}^{(t)})$ is a component of ∇E, i.e., the weight change in the step $t+1$ can be written as follows: $\vec{w}^{(t+1)} = \vec{w}^{(t)} - \varepsilon(t) \cdot \nabla E(\vec{w}^{(t)})$.