MLP – architecture

Notation:

- Denote
  - X a set of *input* neurons
  - Y a set of *output* neurons
  - Z a set of *all* neurons ($X, Y \subseteq Z$)
- individual neurons denoted by indices i, j etc.
  - ξ_j is the inner potential of the neuron j *after the computation stops*
  - y_j is the output of the neuron j *after the computation stops*

  (define $y_0 = 1$ is the value of the formal unit input)
- w_{ji} is the weight of the connection **from** i **to** j

  (in particular, w_{j0} is the weight of the connection from the formal unit input, i.e. $w_{j0} = -b_j$ where b_j is the bias of the neuron j)
- $j_{\leftarrow}$ is a set of all i such that j is adjacent from i (i.e. there is an arc **to** j from i)
- $j^{\rightarrow}$ is a set of all i such that j is adjacent to i (i.e. there is an arc **from** j to i)

MLP – activity

Activity:

- inner potential of neuron j:

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

- activation function σ_j for neuron j (arbitrary differentiable)
- State of non-input neuron $j \in Z \setminus X$ after the computation stops:

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

  (y_j depends on the configuration $\vec{w}$ and the input $\vec{x}$, so we sometimes write $y_j(\vec{w}, \vec{x})$)
- The network computes a function $\mathbb{R}^{|X|}$ do $\mathbb{R}^{|Y|}$. Layer-wise computation: First, all input neurons are assigned values of the input. In the ℓ-th step, all neurons of the ℓ-th layer are evaluated.

MLP – learning

Learning:

- Given a **training dataset** $\mathcal{T}$ of the form

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

  Here, every $\vec{x}_k \in \mathbb{R}^{|X|}$ is an *input vector* end every $\vec{d}_k \in \mathbb{R}^{|Y|}$ is the desired network output. For every $j \in Y$, denote by d_{kj} the desired output of the neuron j for a given network input $\vec{x}_k$ (the vector $\vec{d}_k$ can be written as $\left(d_{kj}\right)_{j \in Y}$).

- **Error function:**

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

  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 – learning algorithm

Batch algorithm (gradient descent):

The algorithm computes a sequence of weight vectors $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$.

- weights in $\vec{w}^{(0)}$ are randomly initialized to values close to 0
- in the step $t+1$ (here $t = 0, 1, 2 \ldots$), weights $\vec{w}^{(t+1)}$ are 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 a *weight update* of w_{ji} in step $t+1$ and $0 < \varepsilon(t) \leq 1$ is a *learning rate* in step $t+1$.

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