JSS30, Summer School, COM5: Machine learning in inverse and ill-posed problems

Larisa Beilina*

Department of Mathematical Sciences, Chalmers University of Technology and Gothenburg University, SE-42196 Gothenburg, Sweden

https://www.jyu.fi/en/research/

Regularized and non-regularized neural networks
Lecture 5

Artificial neural networks

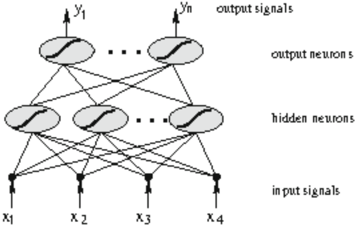


Figure: Example of neural network which contains two interconnected layers (M. Kurbat, *An Introduction to machine learning*, Springer, 2017.)

- In an artificial neural network simple units - neurons- are interconnected by weighted links into structures of high performance.
- Multilayer perceptrons and radial basis function networks will be discussed.

Neurons

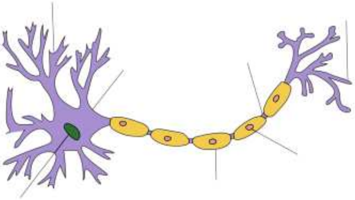


Figure: Structure of a typical neuron (Wikipedia).

- A neuron, also known as a nerve cell, is an electrically excitable cell that receives, processes, and transmits information through electrical and chemical signals. These signals between neurons occur via specialized connections called synapses.
- An artificial neuron is a mathematical function which presents a model of biological neurons, resulting in a neural network.

Artificial neurons

- Artificial neurons are elementary units in an artificial neural network. The artificial neuron receives one or more inputs and sums them to produce an output (or activation, representing a neuron's action potential which is transmitted along its axon).
- Each input is separately weighted by weights ω_{kj}, and the sum $\sum_k \omega_{kj} x_k$ is passed as an argument $\Sigma = \sum_k \omega_{kj} x_k$ through a non-linear function $f(\Sigma)$ which is called the activation function or transfer function.
- Assume that attributes x_k are normalized and belong to the interval $[-1, 1]$.

Artificial neurons

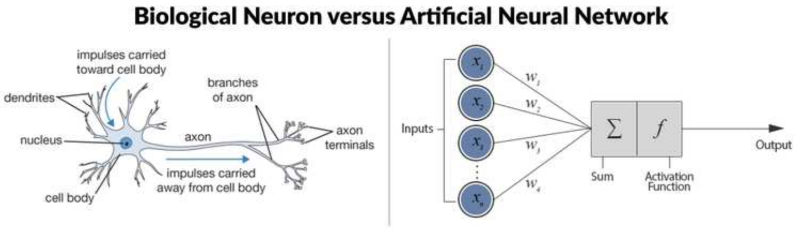


Figure: Perceptron neural network consisting of one neuron (source: DataCamp(datacamp.com)).

Each input is separately weighted by weights ω_{kj}, and the sum $\sum_k \omega_{kj} x_k$ is passed as an argument $\Sigma = \sum_k \omega_{kj} x_k$ through a non-linear function $f(\Sigma)$ which is called the activation function or transfer function.

Artificial neurons: transfer functions

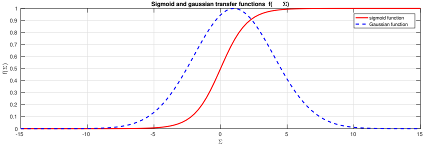


Figure: Sigmoid and Gaussian (for $b = 1, \sigma = 3$ in (2)) transfer functions.

- Different transfer (or activation) functions $f(\Sigma)$ with $\Sigma = \sum_k \omega_{kj} x_k$ are used. We will study sigmoid and gaussian functions.
- Sigmoid function:

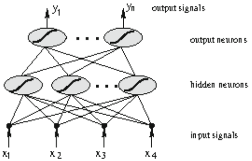
$$f(\Sigma) = \frac{1}{1 + e^{-\Sigma}} \tag{1}$$

- Gaussian function centered at b for a given variance σ^2

$$f(\Sigma) = \frac{e^{-(\Sigma - b)^2}}{2\sigma^2} \tag{2}$$

Forward propagation

Example of neural network called multilayer perceptron (one hidden layer of neurons and one output layer). (M. Kurbat, *An Introduction to machine learning*, Springer, 2017.)



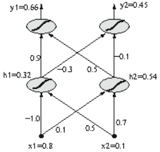
- Neurons in adjacent layer are fully interconnected.
- Forward propagation is implemented as

$$y_i = f(\Sigma_j \omega_{ji}^{(1)} x_j) = f(\Sigma_j \omega_{ji}^{(1)} \underbrace{f(\Sigma_k \omega_{kj}^{(2)} x_k)}_{x_j}), \tag{3}$$

where $\omega_{ji}^{(1)}$ and $\omega_{kj}^{(2)}$ are weights of the output and the hidden neurons, respectively, f is the transfer function.

Example of forward propagation through the network

Source: M. Kurbat, *An Introduction to machine learning*, Springer, 2017.



- Using inputs x_1, x_2 compute inputs of hidden-layer neurons:

$$x_1^{(2)} = 0.8 * (-1.0) + 0.1 * 0.5 = -0.75, \; x_2^{(2)} = 0.8 * 0.1 + 0.1 * 0.7 = 0.15$$

- Compute transfer function (sigmoid $f(\Sigma) = \frac{1}{1+e^{-\Sigma}}$ in our case):

$$h_1 = f(x_1^{(2)}) = 0.32, \; h_2 = f(x_2^{(2)}) = 0.54.$$

- Compute input of output-layer neurons

$$x_1^{(1)} = 0.32 * 0.9 + 0.54 * 0.5 = 0.56, \; x_2^{(1)} = 0.32 * (-0.3) + 0.54 * (-0.1) = -0.15.$$

- Compute outputs of output-layer neurons using transfer function (sigmoid in our case):

$$y_1 = f(x_1^{(1)}) = 0.66, \; y_2 = f(x_2^{(1)}) = 0.45.$$

Backpropagation of error through the network

Our goal is to find optimal weights $\omega_{ji}^{(1)}$ and $\omega_{kj}^{(2)}$ in forward propagation

$$y_i = f(\Sigma_j \omega_{ji}^{(1)} x_j) = f(\Sigma_j \omega_{ji}^{(1)} \underbrace{f(\Sigma_k \omega_{kj}^{(2)} x_k)}_{x_j}). \tag{4}$$

To do this we introduce functional

$$F(\omega_{ji}^{(1)}, \omega_{kj}^{(2)}) = \frac{1}{2} \| t_i - y_i \|^2 = \frac{1}{2} \sum_{i=1}^{m} (t_i - y_i)^2. \tag{5}$$

Here, $t = t(x)$ is the target vector which depends on the concrete example x. In the domain with m classes the target vector $t = (t_1(x), ..., t_m(x))$ consists of m binary numbers such that

$$t_i(x) = \begin{cases} 1, & \text{example } x \text{ belongs to } i\text{-th class,} \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Examples of target vector and mean square error

Let there exist three different classes c_1, c_2, c_3 and x belongs to the class c_2. Then the target vector is $t = (t_1, t_2, t_3) = (0, 1, 0)$.
The mean square error is defined as

$$E = \frac{1}{m}\|t_i - y_i\|^2 = \frac{1}{m}\sum_{i=1}^{m}(t_i - y_i)^2. \tag{7}$$

Let us assume that we have two different networks to choose from, every network with 3 output neurons corresponding to classes c_1, c_2, c_3. Let $t = (t_1, t_2, t_3) = (0, 1, 0)$ and for the example x the first network output is $y_1 = (0.5, 0.2, 0.9)$ and the second network output is $y_2 = (0.6, 0.6, 0.7)$.

$$E_1 = \frac{1}{3}\sum_{i=1}^{3}(t_i - y_i)^2 = \frac{1}{3}((0-0.5)^2 + (1-0.2)^2 + (0-0.9)^2)) = 0.57,$$

$$E_2 = \frac{1}{3}\sum_{i=1}^{3}(t_i - y_i)^2 = \frac{1}{3}((0-0.6)^2 + (1-0.6)^2 + (0-0.7)^2)) = 0.34.$$

Since $E_2 < E_1$ then the second network is less wrong on the example x than the first network.

Backpropagation of error through the network

To find minimum of the functional (29) $F(\omega)$ with $\omega = (\omega_{ji}^{(1)}, \omega_{kj}^{(2)})$, recall it below:

$$F(\omega) = F(\omega_{ji}^{(1)}, \omega_{kj}^{(2)}) = \frac{1}{2} \|t_i - y_i\|^2 = \frac{1}{2} \sum_{i=1}^{m} (t_i - y_i)^2, \tag{8}$$

we need to solve the minimization problem

$$\min_{\omega} F(\omega) \tag{9}$$

and find a stationary point of (8) with respect to ω such that

$$F'(\omega)(\bar{\omega}) = 0, \tag{10}$$

where $F'(\omega)$ is the Fréchet derivative such that

$$F'(\omega)(\bar{\omega}) = F'_{\omega_{ji}^{(1)}}(\omega)(\bar{\omega}_{ji}^{(1)}) + F'_{\omega_{kj}^{(2)}}(\omega)(\bar{\omega}_{kj}^{(2)}). \tag{11}$$

Backpropagation of error through the network

Recall now that y_i in the functional (8) is defined as

$$y_i = f(\sum_j \omega_{ji}^{(1)} x_j) = f(\sum_j \omega_{ji}^{(1)} \underbrace{f(\sum_k \omega_{kj}^{(2)} x_k)}_{x_j}). \quad (12)$$

Thus, if the transfer function f in (12) is sigmoid, then

$$\begin{aligned} F'_{\omega_{ji}^{(1)}}(\omega)(\bar{\omega}_{ji}^{(1)}) &= (t_i - y_i) \cdot y'_{i\,(\omega_{ji}^{(1)})}(\bar{\omega}_{ji}^{(1)}) \\ &= (t_i - y_i) \cdot x_j \cdot f(\sum_j \omega_{ji}^{(1)} x_j)(1 - f(\sum_j \omega_{ji}^{(1)} x_j))(\bar{\omega}_{ji}^{(1)}) \quad (13) \\ &= (t_i - y_i) \cdot x_j \cdot y_i(1 - y_i))(\bar{\omega}_{ji}^{(1)}), \end{aligned}$$

Backpropagation of error through the network

Here we have used that for the sigmoid function $f'(\Sigma) = f(\Sigma)(1 - f(\Sigma))$ since

$$\begin{aligned} f'(\Sigma) &= \left(\frac{1}{1+e^{-\Sigma}}\right)' = \frac{1+e^{-\Sigma}-1}{(1+e^{-\Sigma})^2} \\ &= f(\Sigma)\left[\frac{(1+e^{-\Sigma})-1}{1+e^{-\Sigma}}\right] = f(\Sigma)\left[\frac{(1+e^{-\Sigma})}{1+e^{-\Sigma}} - \frac{1}{1+e^{-\Sigma}}\right] \\ &= f(\Sigma)(1 - f(\Sigma)). \end{aligned} \tag{14}$$

Backpropagation of error through the network

Again, since

$$y_i = f(\sum_j \omega_{ji}^{(1)} x_j) = f(\sum_j \omega_{ji}^{(1)} \underbrace{f(\sum_k \omega_{kj}^{(2)} x_k)}_{x_j}). \tag{15}$$

for the sigmoid transfer function f we also get

$$\begin{aligned} F'_{\omega_{kj}^{(2)}}(\omega)(\bar{\omega}_{kj}^{(2)}) &= (t_i - y_i) \cdot y'_{i\,(\omega_{kj}^{(2)})}(\bar{\omega}_{kj}^{(2)}) \\ &= \left[\underbrace{h_j(1 - h_j)}_{f'(h_j)} \cdot \left[\sum_i \underbrace{y_i(1 - y_i)}_{f'(y_i)}(t_i - y_i)\omega_{ji}^{(1)} \right] \cdot x_k \right] (\bar{\omega}_{kj}^{(2)}), \end{aligned} \tag{16}$$

since for the sigmoid function f we have:
$f'(h_j) = f(h_j)(1 - f(h_j)), f'(y_i) = f(y_i)(1 - f(y_i))$ (prove this). Hint:
$h_j = f(\sum_k \omega_{kj}^{(2)} x_k), y_i = f(\sum_j \omega_{ji}^{(1)} x_j)$.

Backpropagation of error through the network

Usually, $F'_{\omega_{ji}^{(1)}}(\omega)/x_j, F'_{\omega_{kj}^{(2)}}(\omega)/x_k$ in (13), (16) are called responsibilities of output layer neurons and hidden-layer neurons $\delta_i^{(1)}, \delta_i^{(2)}$, respectively, and they are defined as

$$\begin{aligned} \delta_i^{(1)} &= (t_i - y_i) y_i (1 - y_i), \\ \delta_j^{(2)} &= h_j (1 - h_j) \cdot \sum_i \delta_i^{(1)} \omega_{ji}^{(1)}. \end{aligned} \tag{17}$$

By knowing responsibilities (17), weights can be updates using usual gradient update formulas:

$$\begin{aligned} \omega_{ji}^{(1)} &= \omega_{ji}^{(1)} + \eta \delta_i^{(1)} x_j, \\ \omega_{kj}^{(2)} &= \omega_{kj}^{(2)} + \eta \delta_j^{(2)} x_k. \end{aligned} \tag{18}$$

Here, η is the step size in the gradient update of weights and we use value of learning rate for it such that $\eta \in (0, 1)$.

Algorithm A1: backpropagation of error through the network with one hidden layer

- Step 0. Initialize weights.
- Step 1. Take example x in the input layer and perform forward propagation.
- Step 2. Let $y = (y_1, ..., y_m)$ be the output layer and let $t = (t_1, ..., t_m)$ be the target vector.
- Step 3. For every output neuron $y_i, i = 1, ..., m$ calculate its responsibility δ_i^1 as

$$\delta_i^{(1)} = (t_i - y_i)y_i(1 - y_i). \tag{19}$$

- Step 4. For every hidden neuron compute responsibility $\delta_j^{(2)}$ for the network's error as

$$\delta_j^{(2)} = h_j(1 - h_j) \cdot \sum_i \delta_i^{(1)} (\omega_{ji})^1, \tag{20}$$

  where $\delta_i^{(1)}$ are computed using (30).
- Step 5. Update weights with learning rate $\eta \in (0, 1)$ as

$$\begin{aligned} \omega_{ji}^{(1)} &= \omega_{ji}^{(1)} + \eta(\delta_i^{(1)})x_j, \\ \omega_{kj}^{(2)} &= \omega_{kj}^{(2)} + \eta(\delta_j^{(2)})x_k. \end{aligned} \tag{21}$$

Algorithm A2: backpropagation of error through the network with l hidden layers

- Step 0. Initialize weights and take $l = 1$.
- Step 1. Take example x^l in the input layer and perform forward propagation.
- Step 2. Let $y^l = (y_1^l, ..., y_m^l)$ be the output layer and let $t^l = (t_1^l, ..., t_m^l)$ be the target vector.
- Step 3. For every output neuron $y_i^l, i = 1, ..., m$ calculate its responsibility $(\delta_i^{(1)})^l$ as

$$(\delta_i^{(1)})^l = (t_i^l - y_i^l) y_i^l (1 - y_i^l). \tag{22}$$

- Step 4. For every hidden neuron compute responsibility $(\delta_j^{(2)})^l$ for the network's error as

$$(\delta_j^{(2)})^l = h_j^l (1 - h_j^l) \cdot \sum_i (\delta_i^{(1)})^l (\omega_{ji}^{(1)})^l, \tag{23}$$

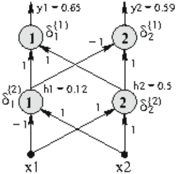
  where $(\delta_i^{(1)})^l$ are computed using (30).
- Step 5. Update weights with learning rate $\eta^l \in (0, 1)$ as

$$\begin{aligned} (\omega_{ji}^{(1)})^{l+1} &= (\omega_{ji}^{(1)})^l + \eta^l (\delta_i^{(1)})^l x_j^l, \\ (\omega_{kj}^{(2)})^{l+1} &= (\omega_{kj}^{(2)})^l + \eta^l (\delta_j^{(2)})^l x_k^l. \end{aligned} \tag{24}$$

- Step 6. If the mean square error less than tolerance, or $\|(\omega_{ji}^{(1)})^{l+1} - (\omega_{ji}^{(1)})^l\| < \epsilon_1$ and $\|(\omega_{kj}^{(2)})^{l+1} - (\omega_{kj}^{(2)})^l\| < \epsilon_2$ stop, otherwise go to the next layer $l = l + 1$, assign $x^l = x^{l+1}$ and return to the step 1. Here, ϵ_1, ϵ_2 are tolerances chosen by the user.

Example of backpropagation of error through the network

Source: M. Kurbat, *An Introduction to machine learning*, Springer, 2017.



- Assume that after forward propagation with sigmoid transfer function we have

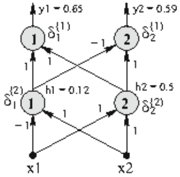
$$h_1 = f(x_1^{(2)}) = 0.12,\ h_2 = f(x_2^{(2)}) = 0.5,$$
$$y_1 = f(x_1^{(1)}) = 0.65,\ y_2 = f(x_2^{(1)}) = 0.59.$$

- Let the target vector be $t(x) = (1, 0)$ for the output vector $y = (0.65, 0.59)$.
- Compute responsibility for the output neurons:

$$\sigma_1^{(1)} = y_1 * (1 - y_1)(t_1 - y_1) = 0.65(1 - 0.65)(1 - 0.65) = 0.0796,$$
$$\sigma_2^{(1)} = y_2 * (1 - y_2)(t_2 - y_2) = 0.59(1 - 0.59)(0 - 0.59) = -0.1427$$

Example of backpropagation of error through the network

Source: M. Kurbat, *An Introduction to machine learning*, Springer, 2017.



- Compute the weighted sum for every hidden neuron

$$\delta_1 = \sigma_1^{(1)} w_{11}^{(1)} + \sigma_2^{(1)} w_{12}^{(1)} = 0.0796 * 1 + (-0.1427) * (-1) = 0.2223,$$
$$\delta_2 = \sigma_1^{(1)} w_{21}^{(1)} + \sigma_2^{(1)} w_{22}^{(1)} = 0.0796 * 1 + (-0.1427) * 1 = -0.0631.$$

- Compute responsibility for the hidden neurons for above computed δ_1, δ_2:

$$\sigma_1^{(2)} = h_1(1 - h_1)\delta_1 = -0.0235, \; \sigma_2^{(2)} = h_2(1 - h_2)\delta_2 = 0.0158.$$

Example of backpropagation of error through the network

- Compute new weights $\omega_{ji}^{(1)}$ for output layer with learning rate $\eta = 0.1$ as:

$$
\begin{aligned}
\omega_{11}^{(1)} &= \omega_{11}^{(1)} + \eta \sigma_1^{(1)} h_1 = 1 + 0.1 * 0.0796 * 0.12 = 1.00096, \\
\omega_{21}^{(1)} &= \omega_{21}^{(1)} + \eta \sigma_1^{(1)} h_2 = 1 + 0.1 * 0.0796 * 0.5 = 1.00398, \\
\omega_{12}^{(1)} &= \omega_{12}^{(1)} + \eta \sigma_2^{(1)} h_1 = -1 + 0.1 * (-0.1427) * 0.12 = -1.0017, \\
\omega_{22}^{(1)} &= \omega_{22}^{(1)} + \eta \sigma_2^{(1)} h_2 = 1 + 0.1 * (-0.1427) * 0.5 = 0.9929.
\end{aligned}
$$

- Compute new weights $\omega_{kj}^{(2)}$ for hidden layer with learning rate $\eta = 0.1$ as:

$$
\begin{aligned}
\omega_{11}^{(2)} &= \omega_{11}^{(2)} + \eta \sigma_1^{(2)} x_1 = -1 + 0.1 * (-0.0235) * 1 = -1.0024, \\
\omega_{21}^{(2)} &= \omega_{21}^{(2)} + \eta \sigma_1^{(2)} x_2 = 1 + 0.1 * (-0.0235) * 1 = 1.0024, \\
\omega_{12}^{(2)} &= \omega_{12}^{(2)} + \eta \sigma_2^{(2)} x_1 = 1 + 0.1 * 0.0158 * 1 = 1.0016, \\
\omega_{22}^{(2)} &= \omega_{22}^{(2)} + \eta \sigma_2^{(2)} x_2 = 1 + 0.1 * 0.0158 * (-1) = 0.9984.
\end{aligned}
$$

- Using computed weights for hidden and output layers, one can test a neural network for a new example.

Perceptron non-regularized neural network

- Step 0. Initialize weights ω_i to small random numbers.
- Step 1. If $\sum_{i=0}^{n} \omega_i x_i > 0$ we will say that the example is positive and $h(x) = 1$.
- Step 2. If $\sum_{i=0}^{n} \omega_i x_i < 0$ we will say the the example is negative and $h(x) = 0$.
- Step 3. Update every weight ω_i using algorithm of backpropagation of error through the network (perform steps 3-5 of A1 or A2)
- Step 4. If $c(\mathbf{x}) = h(\mathbf{x})$ for all learning examples - stop. Otherwise return to step 1.

Here, $\eta \in (0, 1]$ is called the learning rate.

Non-regularized and regularized neural network

Our goal is to find optimal weights $\omega_{ji}^{(1)}$ and $\omega_{kj}^{(2)}$ in forward propagation

$$y_i = f(\Sigma_j \omega_{ji}^{(1)} x_j) = f(\Sigma_j \omega_{ji}^{(1)} \underbrace{f(\Sigma_k \omega_{kj}^{(2)} x_k)}_{x_j}). \tag{25}$$

To do this we introduce functional

$$F(\omega_{ji}^{(1)}, \omega_{kj}^{(2)}) = \frac{1}{2}\|t_i - y_i\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i)^2. \tag{26}$$

Here, $t = t(x)$ is the target vector which depends on the concrete example x. In the domain with m classes the target vector $t = (t_1(x), ..., t_m(x))$ consists of m binary numbers such that

$$t_i(x) = \begin{cases} 1, & \text{example } x \text{ belongs to } i\text{-th class,} \\ 0, & \text{otherwise.} \end{cases} \tag{27}$$

- Non-regularized neural network

$$F(w) = \frac{1}{2}\|t_i - y_i(w)\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i(w))^2. \tag{28}$$

- Regularized neural network

$$F(w) = \frac{1}{2}\|t_i - y_i(w)\|^2 + \frac{1}{2}\gamma\|w\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i(w))^2 + \frac{1}{2}\gamma\sum_{j=1}^{M}|w_j|^2 \tag{29}$$

Here, γ is reg.parameter, $\|w\|^2 = w^T w = w_1^2 + ... + w_M^2$, M is number of weights.

Algorithm: backpropagation of error through the regularized network with one hidden layer

- Step 0. Initialize weights.
- Step 1. Take example x in the input layer and perform forward propagation.
- Step 2. Let $y = (y_1, ..., y_m)$ be the output layer and let $t = (t_1, ..., t_m)$ be the target vector.
- Step 3. For every output neuron $y_i, i = 1, ..., m$ calculate its responsibility δ_i^1 as

$$\delta_i^{(1)} = (t_i - y_i) y_i (1 - y_i). \tag{30}$$

- Step 4. For every hidden neuron compute responsibility $\delta_j^{(2)}$ for the network's error as

$$\delta_j^{(2)} = h_j (1 - h_j) \cdot \sum_i \delta_i^{(1)} (\omega_{ji})^1, \tag{31}$$

  where $\delta_i^{(1)}$ are computed using (30).
- Step 5. Update weights with learning rate $\eta \in (0, 1)$ and regularization parameters $\gamma_1, \gamma_2 \in (0, 1)$ as

$$\begin{aligned} \omega_{ji}^{(1)} &= \omega_{ji}^{(1)} + \eta(\delta_i^{(1)}) x_j + \gamma_1 \omega_{ji}^{(1)}, \\ \omega_{kj}^{(2)} &= \omega_{kj}^{(2)} + \eta(\delta_j^{(2)}) x_k + \gamma_2 \omega_{kj}^{(1)}. \end{aligned} \tag{32}$$