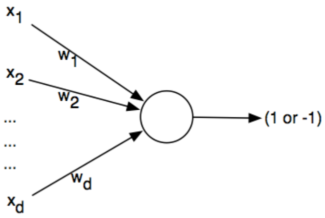
An Elementary Introduction to Neural Networks[1]

Stefan Riezler

Computational Linguistics, Heidelberg University, Germany

[1] Based on Kulkarni & Harman (2011). An Elementary Introduction to Statistical Learning Theory. Wiley

Perceptron as single neuron



- The perceptron algorithm can be seen as **neural network consisting of a single neuron**.
- Its inputs are the features of an example $x = (x_1, \ldots, x_d)$. Each input is connected to the single neuron by weights $w = (w_1, \ldots, w_d)$. The output $a = \mathrm{sign}(\langle x, w \rangle)$ is computed by the sign of the weighted combination of inputs, spanning a linear decision boundary.

Perceptron as single neuron

- Another useful concept is that of a **threshold**.
- The single neuron in the network in Figure 2 has a threshold of 0 since it produces the output $+1$ if $\langle x, w \rangle \geq 0$.
- This can be formalized by an extra input feature x_0 with constant value $+1$, yielding a threshold of $-w_0$:

$$\langle x, w \rangle + x_0 w_0 \geq 0 \text{ iff } \langle x, w \rangle \geq -w_0.$$

Perceptron as single neuron

- Example: A single-unit network formalizing the logical AND operation uses two inputs $x_1, x_2 \in \{0, 1\}$ with fixed weights $w_1 = w_2 = +1$ and a threshold $w_0 = -1.5$. Then

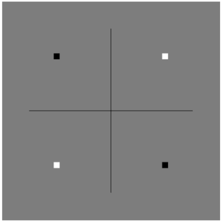
  $$((+1)(-1.5)+x_1(+1)+x_2(+1)) \geq 0 \text{ iff } x_1 = 1 \text{ AND } x_2 = 1.$$

- Using the output function a, we can rewrite the perceptron **update rule** as follows (where $y \in \{+1, -1\}$):

  $$w = w + \frac{1}{2}\eta^{(t)}(y - a)x \qquad (1)$$

Multilayer neural networks

- Since the decision boundary implemented by a perceptron is a hyperplane in $\mathbb{R}^d$, the perceptron can only classify correctly if the examples are **linearly separable**.
- A well-known example of a problem that is not linearly separable is the XOR problem. Suppose two input features x_1 and x_2. Classes "true" and "false" fall into opposite quadrants of the decision space and cannot be separated linearly by a hyperplane.



-1 XOR -1 = *false*

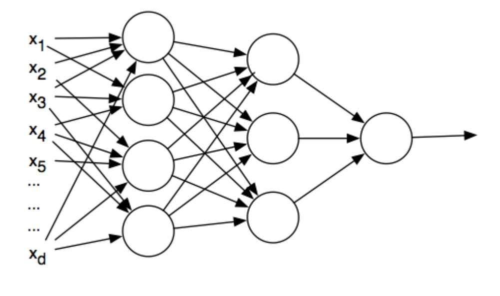
-1 XOR +1 = *true*

+1 XOR -1 = *true*

+1 XOR +1 = *false*

Multilayer neural networks

- **Multilayer networks** with just three layers and enough units in each layer can approximate any decision rule.

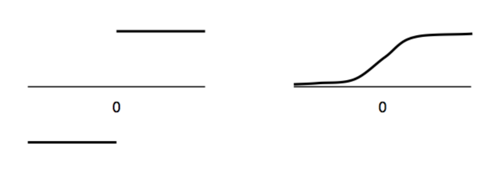


Multilayer neural networks

- The representation capabilities of multilayer networks sketched above allow to model **non-linear decision boundaries** such as the XOR problems shown above.
- This is done by a multilayer network where each unit in the first layer computes a linear decision boundary.
- The outputs of this layer are passed as inputs to a second layer, where each unit performs a logical AND that implements an intersection of the half-spaces computed in the first layer.
- The outputs of the second layer are passed through a final unit that performs a logical OR operation that implements a union of convex sets computed in the second layer.

Multilayer neural networks

- In difference to a single unit network, outputs are not fed as inputs into following layers. This makes the $\mathrm{sign}(\cdot)$ function less appropriate since it causes drastic changes in outputs and is discontinuous and not differentiable at 0.



- In multilayer networks we use a smooth function such as the **sigmoid function** that is continuously differentiable and varies between 0 and 1:

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

Multilayer neural networks

- A useful fact is to know its derivative which is

$$\sigma'(y) = \frac{d\sigma}{dy} = \sigma(y)(1 - \sigma(y)).$$

- The output of of a unit is now given by

$$a = \sigma(\langle x, w \rangle).$$

- During training, target outputs are taken to be $y \in \{0, 1\}$. At test time, the decision for class 1 is made by checking if $a \geq 1/2$, and decide for class 0 otherwise.

Training a multilayer network: Backpropagation

- **Training a multilayer network** requires a sequential implementation of gradient descent, called **backpropagation**.
- General form of weight adjustment δw in update rule is given by negative (stochastic) (sub)gradient, that includes a learning rate η, an error term, and an input term:

general form: $\Delta w = -\eta * \text{ error term } * \text{ input term}$

Backpropagation

single unit perceptron: $\Delta w_i = \eta(y_i - a_i)x_i$ (folding $\frac{1}{2}$ into η)

unit in layer l: $\Delta w_{ij}(l) = -\eta\delta_i(l)a_j(l-1)$ where

$$u_i(l) \text{ total input to unit } i \text{ in layer } l, \tag{2}$$

$$a_i(l) = \text{ output of unit } i \text{ in layer } l, \tag{3}$$

$$w_{ij}(l) = \text{ weight from unit } j \text{ in layer } l-1 \text{ to unit } i \text{ in layer } l, \tag{4}$$

$$\delta_i(l) = \sigma'(u_i(l)) \sum_k \delta_k(l+1) w_{ki}(l+1) \text{ for } l = 1, \ldots, L-1, \tag{5}$$

$$\delta_i(L) = \sigma'(u_i(L))(y_i - a_i(L)). \tag{6}$$

Backpropagation

- The term "backpropagation" comes from the following intuition:
- After choosing initial weights, for each training example, start at the input layer, and compute the total input $u_i(l)$ for each unit in each layer by feeding forward the computation to the output layer.
- Then compute the final output $a_i(L)$, compute the error by comparing it with the the target output, and propagate the unit errors $\delta_i(l)$ for each layer backward to the second layer.
- Adjust weights $w_{ij}(l)$ by adding $\Delta w_{ij}(l)$.

Derivation of backpropagation for single unit

Let the error at the *m*th example be

$$E_m = \frac{1}{2}(y - a)^2 = \frac{1}{2}(y - \sigma(u))^2. \tag{7}$$

The weight adjustment is defined by the negative (sub)gradient

$$\Delta w_i = -\eta \frac{\partial E_m}{\partial w_i}. \tag{8}$$

Using the chain rule, we can compute the variation in E_m by adjusting w_i directly through the input $u = \langle x, w \rangle$ as

$$\frac{\partial E_m}{\partial w_i} = \frac{\partial E_m}{\partial u} \frac{\partial u}{\partial w_i} \tag{9}$$

Derivation of backpropagation for single unit

The derivatives are

$$\frac{\partial u}{\partial w_i} = x_i, \tag{10}$$

and

$$\frac{\partial E_m}{\partial u} = -(y - \sigma(u))\frac{\partial \sigma(u)}{\partial u} = -(y - a)\sigma'(u). \tag{11}$$

Thus the final expression for Δw_i is

$$\Delta w_i = \eta(y - a)\sigma'(u)x_i. \tag{12}$$

Backpropagation for multilayer network

The input $u_i(l)$ to unit i in layer l depends on the output of the units in the previous layer $a_j(l-1)$ to which it is connected with weight w_{ij}:

$$u_i(l) = \sum_j w_{ij}(l) a_j(l-1). \tag{13}$$

The output of this unit is $u_i(l)$ passed through the sigmoid:

$$a_i(l) = \sigma(u_i(l)). \tag{14}$$

Again, the weight adjustment is defined by the negative (sub)gradient

$$\Delta w_{ij} = -\eta \frac{\partial E_m}{\partial w_{ij}}. \tag{15}$$

Backpropagation for multilayer network

The error gradient depends on $w_{ij}(l)$ through $u_i(l)$, thus using the chain rule, we get

$$\frac{\partial E_m}{\partial w_{ij}(l)} = \frac{\partial E_m}{\partial u_i(l)} \frac{\partial u_i(l)}{\partial w_{ij}(l)}. \tag{16}$$

For the second term, we get immediately

$$\frac{\partial u_i(l)}{\partial w_{ij}(l)} = a_j(l-1). \tag{17}$$

Let the first part be denoted by

$$\delta_i(l) = -\frac{\partial E_m}{\partial u_i(l)}. \tag{18}$$

Backpropagation for multilayer network

Backpropagation involves a recursive definition of the unit errors $\delta_i(l)$ at each layer using equations (5) and (6). At the output layer L, we can apply equation (11) to the single unit L, yielding

$$\delta_i(L) = -\frac{\partial E_m}{\partial u_i(L)} = (y - a(L))\sigma'(u_i(L)). \tag{19}$$

Backpropagation for multilayer network

For $l < L$, we note that E_m depends on $u_i(l)$ only through the output $a_i(l)$, thus using the chain rule, we get

$$\delta_i(l) = -\frac{\partial E_m}{\partial a_i(l)} \frac{\partial a_i(l)}{\partial u_i(l)}. \tag{20}$$

Since $a_i(l) = \sigma(u_i(l))$, we can compute the second term as

$$\frac{\partial a_i(l)}{\partial u_i(l)} = \sigma'(u_i(l)). \tag{21}$$

Backpropagation for multilayer network

To compute the first term, we note the E_m depends on $a_i(l)$ only through the effect the output has as to the inputs to all the units in layer $l+1$. Hence,

$$-\frac{\partial E_m}{\partial a_i(l)} = -\sum_k \frac{\partial E_m}{\partial u_k(l+1)} \frac{\partial u_k(l+1)}{\partial a_i(l)}. \tag{22}$$

Using equation (13), we get the derivative of the second term

$$\frac{\partial u_k(l+1)}{\partial a_i(l)} = \frac{\partial \sum_j w_{kj}(l+1)a_j(l)}{\partial a_i(l)} = w_{kj}(l+1). \tag{23}$$

Using the definition in (18), we get for the first term

$$-\frac{\partial E_m}{\partial u_k(l+1)} = \delta_k(l+1). \tag{24}$$

Backpropagation for multilayer network

Combining the last two equations, we get

$$-\frac{\partial E_m}{\partial a_i(l)} = \sum_k \delta_k(l+1) w_{kj}(l+1). \tag{25}$$

Substituting this into equation (20), we get

$$\delta_i(l) = \sigma'(u_i(l)) \sum_k \delta_k(l+1) w_{kj}(l+1). \tag{26}$$

Substituting this equation and (17) into (15) gives us the update rule

$$\Delta w_{ij}(l) = -\eta \delta_i(l) a_j(l-1).$$