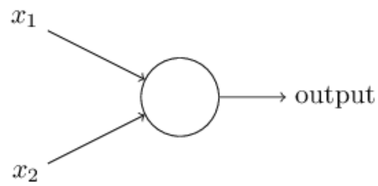
1. **Neural Networks**
We want to build a very simple neural network. If the basic principles of neural networks are not clear to you, have a look at Michael Nielsen's excellent introduction:
`http://neuralnetworksanddeeplearning.com/`.



Our simple network will consist of only one layer of four neurons that have only two inputs and one output. The input values are multiplied by a weighting function and added up with a bias,

$$z(i) = \sum_{j=1,2} w_j(i) \cdot x_j(i) - b(i); \tag{1}$$

the output of an individual neuron is then

$$\sigma(z(i)) = \frac{1}{1 + e^{z(i)}}. \tag{2}$$

(These are *sigmoidal neurons*, other functions, e.g. tanh or ReLU are possible.) The output of the neural network is the sum of the output of the neurons, again with a weighting function,

$$\text{output} = \sum_{i=1}^{4} w_o(i)\sigma(z(i)). \tag{3}$$

We want our network to learn to classify points in the x-y-plane according to the quadrants 1/3 or 2/4. Therefore, our inputs will be the coordinates x and y and we will train the network with 100 random points (x, y) and their quadrant.

The *cost function* or *loss function* is defined (very similarly to χ^2-fits) as

$$C(w_j(i), b(i), w_o(i)) = \sum_{\text{training data}} (\text{sign}(x \cdot y) - \text{output}(x, y))^2, \tag{4}$$

where $\text{sign}(x \cdot y)$ is the "correct" output for (x, y) that we want the network to learn.

After assigning random starting values to all the weights and biases, we will train the network by repeated gradient descent,

$$w_j(i) \to w_j(i) - \eta \frac{\partial C}{\partial w_j(i)} \tag{5}$$

(and respectively for $b(i)$ and $w_o(i)$), where in each round only some, randomly selected variables (e.g. 20%) are updated and η should be small enough to avoid instabilities, e.g. $\eta = 0.01$.

Try to solve the following problems:

(a) Implement the neural network as described above, train the network for a few hundred rounds, and plot the output function of your trained network.

(b) Try to speed up training by implementing a more effective method to minimize the cost function, e.g. Levenberg-Marquardt, and/or implement a different activation function for the neurons.

A working example of a neural network as described above can be studied at `https://playground.tensorflow.org` by selecting one hidden layer with four neurons, the *features* x_1 and x_2 and the top right of the four data sets.

Your output function could e.g. look like this (the points are the random points used to train the network):