Introd. to Neural Networks

Dr. S. Recksiegel WS 2022/23

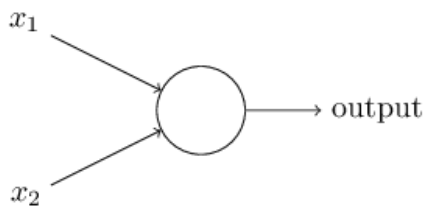
Exercise Sheet 6 L. Rickler, E. Eberhard, J. Mayer-Steudte, L. Wicke

November 23, 2022 Moodle: https://www.moodle.tum.de/course/view.php?id=79979

A very simple first Neural Network

We want to build a very simple neural network. If the basic principles of neural networks are not clear to you yet, 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 each. The input values are multiplied by weighting factors and added up with a bias, for neuron i,



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

The output of neuron i is then

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

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

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

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.

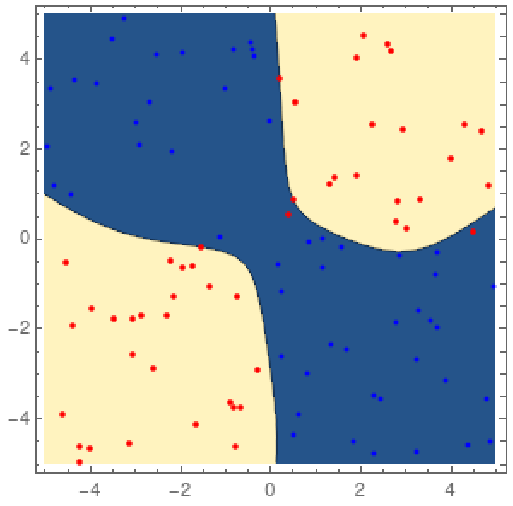
After assigning random starting values to all the weights and biases, we will train the network by repeated gradient descent,

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

(and respectively for $b(i)$ and $w_o(i)$), where in each round either only some, randomly selected variables (e.g. 20%) are updated or only a part of the training data are used. The parameter η should be small enough to avoid instabilities, e.g. $\eta = 0.01$.

Implement the neural network as described above, preferably in Python. Train the network for a few hundred rounds, and plot the output function of your trained network.

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



If you have time, you can try to improve your network:

1. 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.
2. Try to think of another cost function that would work in this example and implement it.

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.