Project 4: A 3-layer neural network

Prerequisites and necessary prior knowledge

A comparably short and comprehensible introduction to neural networks is given on the 3blue1brown channel [2]. For this project, you do not necessarily have to go through all the details of that lesson. However, you should be familiar with basic consepts of object oriented programming (in any programming language) before you start. You will need to define your own class with properties and class methods. Study the documentation on how to create a simple class in MATLAB [1].

Introduction

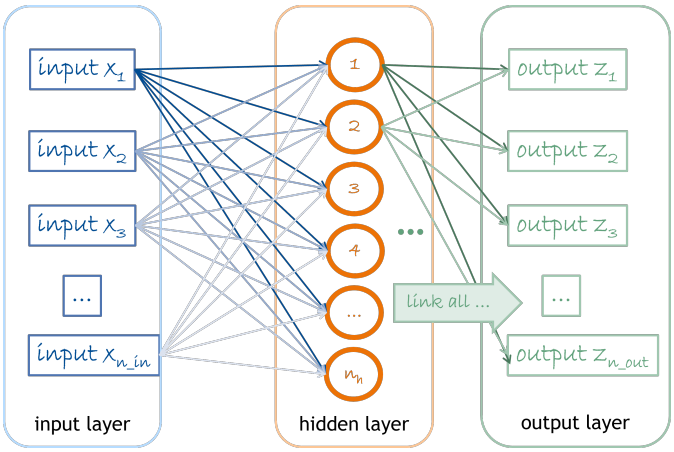

Figure 1: Schematic of a basic 3-layer neural network.

Neural networks are currently among the most popular topics in many scientific fields. This project guides you through the implementation of a basic 3-layer neural network.

A neuron recieves signals from various input paths (nodes) and transmits a signal to an output, if the sum of all input signals exceeds a certain threshold value. We implement this concept as described in the followin in a 3-layer neural network (cf. Fig. 1).

Transmitting signals through the network

We take a number n_{in} of input nodes and fill them with input values x_k where k is the number of the input node.

The middle layer, a "hidden" layer, whose number of nodes is n_h recieves input signals which are computed by

$$s_{inh}^{(j)} = \sum_{k=1}^{n_{in}} w_{kj} \cdot x_k^{(j)}$$

where w_{kj} is the link weight between the input node k and the hidden node j. If we arrange all the link weights in an $(n_h \times n_{in})$-matrix W_{inh}, we can compute the input signals into the hidden layer by a matrix multiplication:

$$\vec{s}_{inh} = W_{inh} \cdot \vec{x}.$$

The output signal $s_h^{(j)}$ from a hidden node j is computed by applying the so-called activation function (or threshold function) f_{act}:

$$s_h^{(j)} = f_{act}\left(s_{inh}^{(j)}\right)$$

Different types of functions can be used as the activation function. A popular one is the *sigmoid function* (or also called *logistic function*).

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

As an alternative the *Relu function* has made its way into artificial neural nets:

$$\text{relu}(x) = \max(0, x).$$

Both have their own (dis-)advantages for the training process of a neural network.

Summarizing, we can say that the following vectorized formula computes the output signals of the hidden layer from the input layer:

$$\vec{s}_h = f_{act}\left(W_{inh} \cdot \vec{x}\right).$$

Similarily, the signals of the third layer, i.e. the output layer here, are computed:

$$\vec{z} = f_{act}\left(W_{hout} \cdot \vec{s}_h\right),$$

where $\vec{z}$ contains the output signals from the output layer, W_{hout} is an $(n_{out} \times n_h)$-matrix holding the link weights between the nodes of the hidden layer and those of the output layer and n_{out} is the number of nodes in the final output layer.

Overall, a neural network can be regarded as a function $nn\left(\vec{x}\right)$ that calculates a vector $\vec{z}$ of output signals from a vector of input signals $\vec{x}$.

$$\vec{z} = nn\left(\vec{x}\right) = f_{act}\left(W_{hout} \cdot f_{act}\left(W_{inh} \cdot \vec{x}\right)\right).$$

Backpropagating errors

The link weights of a neural network for a given problem are obtained by *training* the network. Therefore, we need a set of *training data* which contains vectors of input signals together with the corresponding target outputs of the final layer.

Knowing the target output vector $\vec{t}$ from the training data we can compute an error vector $\vec{e}_{out}$:

$$\vec{e}_{out} = \vec{t} - \vec{z}.$$

These errors are *backpropagated* through the network with the help of the transposed matrices of link weights:

$$\vec{e}_h = W_{hout}^T \cdot \vec{e}_{out}$$
$$\vec{e}_{in} = W_{inh}^T \cdot \vec{e}_h = W_{inh}^T \cdot \left(W_{hout}^T \cdot \vec{e}_{out}\right)$$

($\vec{e}_h$ and $\vec{e}_{in}$: backpropagated errors in the hidden layer and the input layer, respectively).

Training the network

The weights that connect two layers are initialized by random values $0 < w_{jk} < 1$ and impoved in an iterative procedure. They are updated by:

$$w_{jk}^{new} = w_{jk}^{old} - \alpha \cdot \frac{\partial E}{\partial w_{jk}}$$

where $\alpha \in (0, 1)$ is the *learning rate* and

$$\frac{\partial E}{\partial w_{jk}} = -\left(e_k\right) \cdot \text{sigmoid}\left(\sum_j w_{jk} \cdot s_j\right)\left(1 - \text{sigmoid}\left(\sum_j w_{jk} \cdot s_j\right)\right) \cdot s_j.$$[1]

e_k being the error in the k-th layer and s_j the signals propagating from from the j-th to the k-th layer (which are at the same time the output signals from the j-th layer).

In vectorized notation this gives

$$\Delta W = \alpha \cdot [\vec{e}_k \; .* \; \vec{s}_k \; .* \; (1 - \vec{s}_k)] \cdot \vec{s}_j^T$$

Design of the 3-layer network

We set up our network with 784 input nodes. Each input node will hold the intensity value of one pixel of a 28x28 pixles image. The output layer has 10 nodes, one for each digit [0-9]. In this project we set the number of hidden nodes to 100. Please, feel free to test the performance of the network with other numbers of hidden nodes.

[1] The derivative of the sigmoid function is $\frac{\partial}{\partial x}\text{sigmoid}(x) = \text{sigmoid}(x)\ (1 - \text{sigmoid}(x))$,

Preparing the data

The moodle course contains 4 datasets of handwritten digits from the MNIST database: one small training data set of 100 entries together with a test data set of 10 entries and two larger sets for training and testing the network. Use the small data sets for testing purposes while developing your implementation in MATLAB/Octave. Finally, you should train and analyse your network with the larger data sets.

Each line contains 785 values: the first value represents the digit, the other 784 values are the intensities of 28x28 pixels which correspond to the image of the handwritten digit. The intesities are encoded by integer values from 0 to 255.

Neural nets work best, if the outputs are in the range of the values that the activation function can output. Additionally, values too close to zero (below the machine epsilon) should be avoided. Therefore, the intensity values v_k from the MNIST data have to be scaled and shifted as follows to obtain the signals for the input layer:

$$x_k = v_k/255.0 \cdot 0.99 + 0.01$$

The target vector $\vec{t}$ is produced from the label, which is the first number in each line of the data file. Its value is set to 0.99 for node which represents the corresponding digit. For all other nodes it is set to 0.1. For example, if the label is 0, the target vector should be `t = [ 0.99 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]`; if the label is 5, the vector is `t = [ 0.01 0.01 0.01 0.01 0.01 0.99 0.01 0.01 0.01 0.01]`.

Hint: Be careful with the indexing in Matlab compared to other programming languages.

Tasks

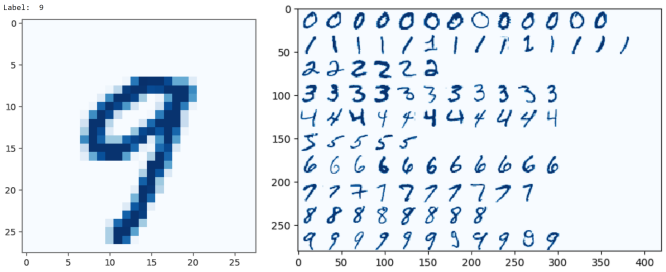

Figure 2: Handwritten digits from the MNIST data base.

1. Take a first look at the data by plotting 28x28 pixels in a 2d intensity plot similar to Fig. 2 (left). You may also want to see multiple examples of each digit 0-9 from the large training data set (as in Fig. 2, right).

2. Use the template from the CBEM moodle to implement a class for a basic 3-layer neural network. Hints for the implementation are included in the source file.
3. Use the large training data set to train a 3-layer network using your implementation.
4. Evaluate the performance of your netwokr. Compute how many cases of the large test data set in total your implementation gets correct. Compute the percentage of correct predictions separately for each digit. Which digits are best predicted? Which have the largest number of wrong predictions? Visualize your evaluation appropriately in a view graph.
5. Repeat the training and testing at least 3 times and observe the influence of different random initializations of the link weights. How does the situation change, when you initialize all link weights with the same value, for example $w_{jk} = 0.5$?
6. Optional: Produce a personal dataset with 28x28 pixels of digits with your own handwriting. Is the imple network able to identify these correctly?

Solution upload

Everybody must upload the following

- the .m-files which compute your solution;
- presentation slides (in landscape format) of your work (containing, in particular, the vizualization of your computation results), preferably in the `.pdf` file format, which you will use for your talk at the end of the semester or in your video;
- a list of your group colleagues.

If you prepare a video, it is sufficient if one person uploads the video file. The other group members should include a comment in their solution who is responsible for the video upload.

Groups: 3–5 students. Please register your group using the Moodle activity "List of group colleagues". If your group changes, follow the instructions given in that activity.

Deadline: Upload your project (.m files and pdf/presentation) by **July 18** to the Moodle assignment "MATLAB code and presentation". To account for technical or other problems which might occur, the upload remains open until July 31st.

Presentation: Each group has to give a presentation (max. 10 minutes). which can either be prepared as a video or given as a live presentation during the last two weeks of the lecture period. Do not necessarily include every single line of code on your slides. Rather show general schematics, point out interesting parts, discuss specific key lines or

findings which you might not have expected. If e.g. you have had the help of AI tools, let your audience know how you used them and in what way they helped you improve your solution. Within the CBEM lab it is also welcome to discuss obstacles that you encounter. Let your audience learn from your experience.

References

[1] Mathworks. *Creating a Simple Class*. URL: https://www.mathworks.com/help/matlab/matlab_oop/create-a-simple-class.html.

[2] Grant Sanderson. *3blue1brown: But what are neural networks?* URL: https://www.3blue1brown.com/lessons/neural-networks.