KEN 4154 Assignment 1

Backpropagation

Adrian Sondermann, Abel de Wit

November 3, 2020

1 Introduction

In this assignment we are tasked to write a neural network from scratch using no external libraries. The task at hand is training the network such that it is able to represent the input that is passed in the output layer as well, while going trough a middle/hidden layer that has less nodes than the in- and output layers.

After this is implemented, we are asked to interpret the results of our network, including the convergence, the weights and activations of the network, to see whether we can understand how the network learned to still represent these 8 input neurons trough the 3 hidden neurons.

2 Implementation

The implementation strictly follows the algorithm described in the lectures slides and the help-file for backpropagation. A small change was made regarding the gradient descent: the weights get updated every time the Δ_{ij}^{l} are calculated, using a single training example. Therefore, the algorithm converged faster while testing, because the weights start to spread in different directions independently. Finally a learning curve containing the sum of squared output errors per iteration is returned.

The whole implementation is vectorised and in order to generate reproducible weights, the random generator of `numpy` is seeded with 0 initially.

3 Results

3.1 Tuning the parameters

Using the implementation of the network, we created a simple GridSearch over the three tunable parameters `learningRate` (between 0.1 and 0.9), `weightDecay` (between 0 and 0.5), and `iterations` (between 100 and 5000). One neural network was trained per unique parameter combination and the results were saved. All these combinations converge to a local minimum of the costs in the long run (possibly more than 5000 iterations needed!). Due to the independence of a single weight update on 7 training examples, the cumulative errors may fluctuate before finally converging to a local minimum.

From this GridSearch the best parameters were derived, which are given in Table 1 and result in a error of approximately 0.000019. The learning rates (errors) are visualized in Figure 1.

Another aspect to pay attention to is the choice of the `weightDecay` parameter, as nonzero values result in an training error rate always above 0. Because the training data is noise-free, using no `weightDecay` yields the lowest error.

3.2 Analysis

Using specific parameters, the program can be executed to analyse the network with respect to the weights. For the values from Table 1 the final weights in the hidden and output layer are presented in Figures 2 and 3. The x-axis contains the bias (0) and the weights which will be used to calculate the activations of the

parameter	value
learningRate	0.9
weightDecay	0
iterations	5000

Table 1: Optimal parameters by GridSearch

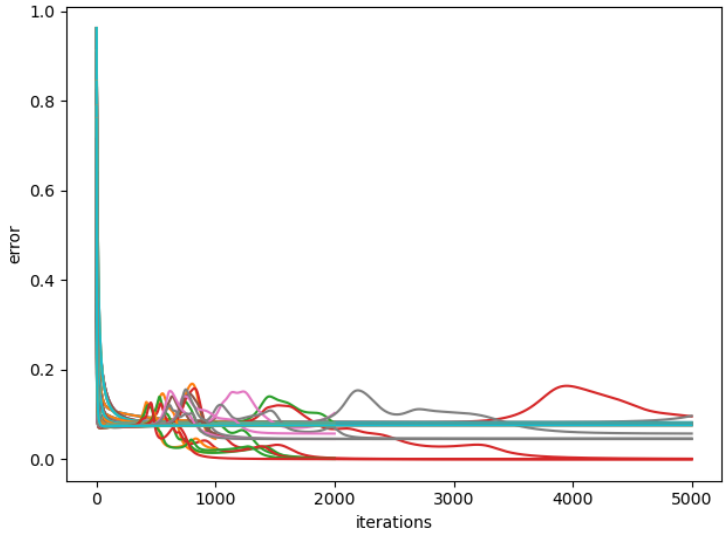

Figure 1: Learning curves of all models

neurons presented by the y-axis. Positive and negative weights are colored distinctively. According to those colors, every input is binary "encoded" by the hidden layer and "decoded" by the output layer. This results in weight matrices (without biases), which are transposed versions of another, if the values are seen as binary (smaller/bigger than 0). Because of this, the visualized network looks symmetrical along the hidden layer, as visualized in Figure 4.

Moreover, as the input and output can be viewed as binary data the following can be assumed: as

$$\#possibleValues^{\#hiddenNotes} = 2^3 = 8 = \#inputs = \#outputs \tag{1}$$

holds, the whole assignment could not be solved with 2 hidden nodes, if a low error rate is desired. If less output nodes are used the number of internal nodes could be reduced.

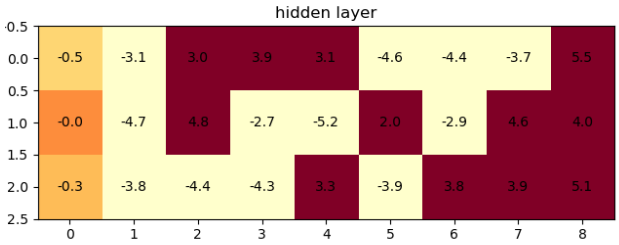

Figure 2: Weights in hidden layer

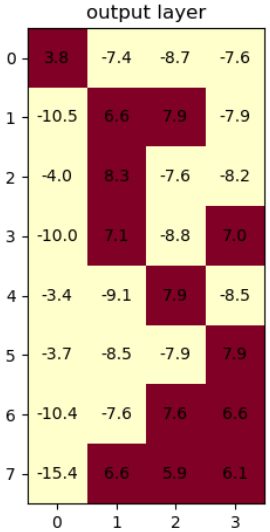

Figure 3: Weights in output layer

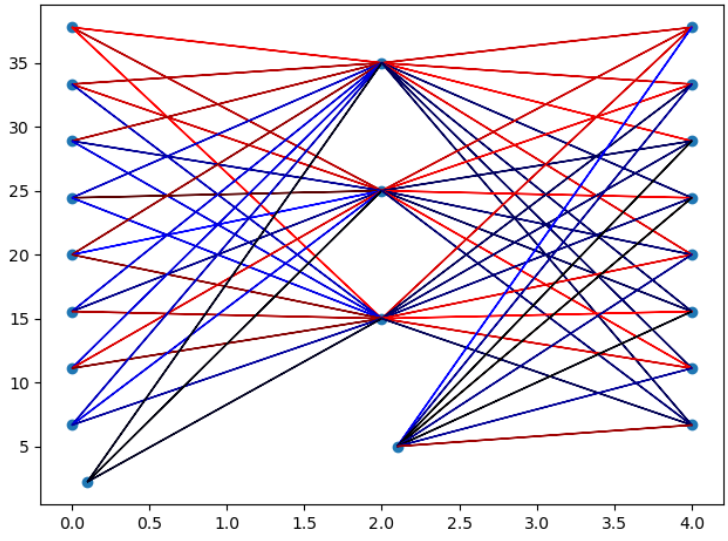

Figure 4: Visualization of the neural network