Jacobs University

Machine Learning

Exercise Sheet 11

Solving the XOR function using an MLP

Authors:
Steven Abreu
Alex Maiereanu
Tudor Maiereanu

May 23, 2018

1 Task

Outline. Implement an MLP with a single hidden layer from scratch and train it on a classical historical challenge problem, the XOR function. You can earn valuable bonus points by going beyond this basic task, see end of this task sheet.

Data and learning task. The task is to train an MLP on the exclusive-OR function $XOR : \{0,1\}^2 \to \{0,1\}$ given by

$$XOR(x,y) = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{if } x \neq y \end{cases} \tag{1}$$

with an MLP that has 2 input units and one output unit. The training data consists of the four input-output pairs that define the XOR function. Note that this is a noise-free deterministic task and the function is completely covered by the training data. There are no "new" test data, hence training and test error coincide. This means that all the over/underfitting problems are a non-issue here – all you have to aim for is a low training error.

Suggested network structure. For this problem an MLP with a single hidden layer that has 2 units suffices. Also include bias units in the input and the hidden layer, as in Figure 25 in the lecture notes. I suggest you use the logistic sigmoid in the hidden layer, simply because its derivative is described in the lecture notes.

Learning goal. The purpose of this exercise is not so much to solve the XOR learning problem, but to become a close acquaintance of the most widely used learning algorithm im ML, backprop – and shake hands with history. You can basically just put the formulas 53, 55, 59, 61, 63 in the lecture notes into program code, which is not in itself difficult, but error-prone (you will presumably be hit by that sad fact), and on that way you have to inspect and understand each of those little formulas, which is what this exercise aims to achieve.

Detailed instructions. Implement an MLP with the structure outlined above, and implement backprop to compute the error gradients. I suggest to use the quadratic loss with no regularizer. Initialize your MLP with small random weights. After each epoch, compute the mean training squared error.
At any stage during training, your MLP will give output values that are real numbers, not clean 0 or 1 values. If you postprocess the MLP outputs by thresholding at 0.5 (postprocessed output = 1 if MLP output > 0.5, else = 0) your MLP turns into a Boolean function. After some training epochs this Boolean function will be the XOR. You may stop at that point and be satisfied, or you may continue

training to make the MLP output come closer and closer to the precise 0-1-values.

Deliverable. A typeset report which documents your learning set-up (initialization, learning rate) and shows a graphics of the learning curve, that is a plot of the mean training error vs. the epoch number. Target size: 1.5 pages, excluding graphics. Plus, your code. It should be a self-contained single script (Matlab or Python) which does the complete network creation, initialization, learning and plotting.

Grading. A nicely done report and functioning code will give 100%.

Bonus points (max 5) are awarded for work that goes beyond the deliverable sketched above. Bonus points will be added undiluted to the final course grade, that is a bonus of 5 pts will, for example, raise a course grade from 85% to 90%. Possible extensions for harvesting bonus points:

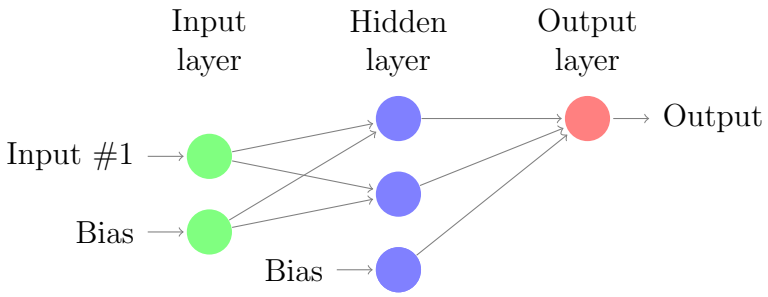
- Instead of the Boolean XOR, train a continuous version of the same problem, namely a function $f : [-1, 1]^2 \to 0, 1$, defined by $f(x, y) = sign(xy)$. Display the performance of your final MLP by a nice 2-dim surface graphic.
- Try to train more complex Boolean functions. A classical teaser is the parity function: $P : 0, 1^n \to 0, 1$ given by:

$$P(x) = \begin{cases} 1 & \text{if the number of 1's in the Boolean vector } x \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

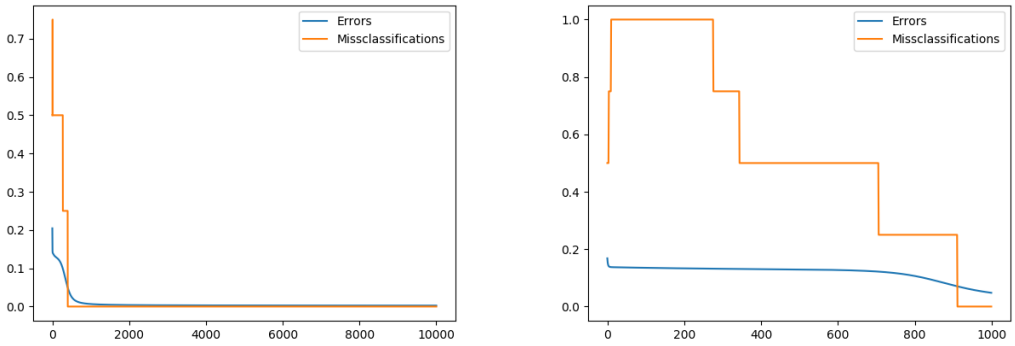
  When n grows, this quickly becomes a (very) difficult learning task.
- Train an MLP classifier on our digits dataset. This is a much more demanding task because now you have to fight with the bias-variance dilemma and find a way to prevent overfitting (suggestion: use a large enough MLP with 2 hidden layers and use early stopping for regularization). If you can push the final classification test error (on the standard test data consisting of the second 100 images per class) below 3%, you can feel very satisfied. The best ML methods reported in the literature achieve about 1.8% test error, but they employ combinations of several methods, not just a single MLP.

2 Solving the XOR function with an MLP

We used an MLP with one hidden layer that uses one neuron for the input layer (plus one bias neuron), two neurons for the hidden layer (plus one bias neuron) and one output neuron, as illustrated below:



Our MLP uses the Sigmoid function in the hidden layer.
For training, we used the backpropagation algorithm. For our errors, we used quadratic loss with no regularizer. Our training data is the same as our testing data, which is: $\{(0,0),(1,1),(1,0),(0,1)\}$ with the targets: $\{0,0,1,1\}$.
We initialized our neural network with small random weights, $w_{ij} \in [-1,1]$. In each epoch of the training procedure, we go through the training data once. We used the learning rate $\lambda = 0.1$.
For 10.000 epochs, we get the following error plot:



(a) 10.000 epochs

(b) 1.000 epochs

Figure 1: Plot of MSE and MISS for XOR MLP

From the graph, it is easy to see that after around 1.000 epochs, we have no more classifications, but we can continue training in order to further reduce the mean squared error.

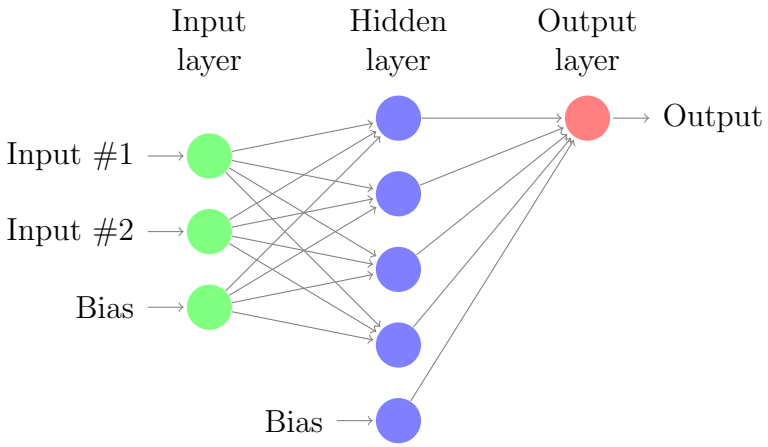
3 Training an MLP for the Sign function

We want to train our MLP to recognize the function:

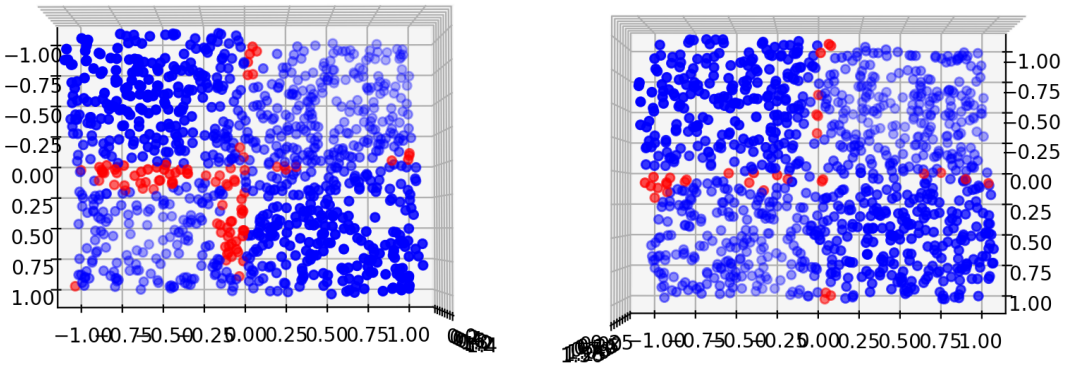
$$sign : [0,1]^2 \rightarrow \{0,1\} \tag{3}$$

$$sign(x,y) = \begin{cases} 0 & \text{if } xy < 0 \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

We used the following structure for our neural network:



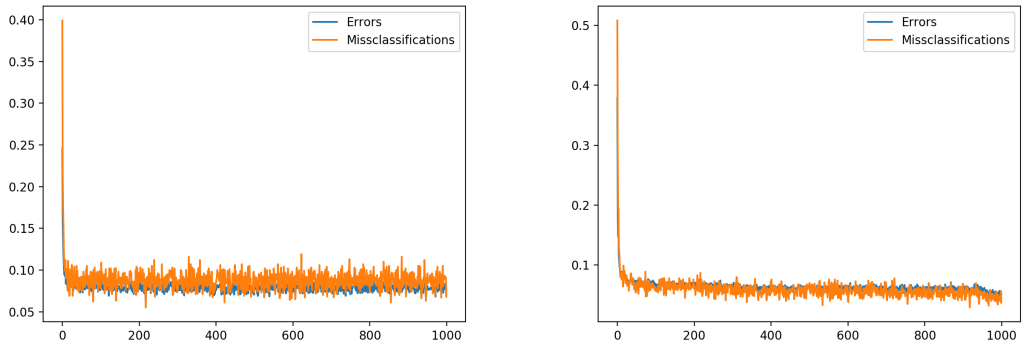
Aside from additional input and hidden units, we left the MLP setup unchanged from the XOR MLP. We generated training patterns randomly in every epoch and generated random patterns for testing as well.
Using a 3D scatter plot, we visualized the misclassifications by marking correctly classified patterns in blue and wrongly classified patterns in red. We decided to use 1.000 epochs (with 1.000 training patterns each, thus a total of 1.000.000 training patterns). The following figure compares the performance of our MLP using either 4 hidden neurons or 10.
It is very obvious that 10 hidden neurons give us a much clearer classification with fewer misclassifications.



(a) Using 4 hidden neurons

(b) Using 10 hidden neurons

Figure 2: Plot of MSE and MISS for XOR MLP



(a) Using 4 hidden neurons

(b) Using 10 hidden neurons

Figure 3: Error plot of MSE and MISS for XOR MLP

4 Training an MLP for Digit Classification