Machine Learning, Spring 2018: Exercise Sheet 11

Please send your type-set solutions by email to our two TA's Xu He ("Owen") x.he@jacobs-university.de and Tianlin Liu t.liu@jacobs-university.de . Join into groups of two or three and submit a single solution sheet per group, indicating the group members' names on the sheet.

Deadline for submission is Wednesday May 16, 23:59 hrs (email sending timestamp). Submissions arriving later (even a second after midnight) will have their grade reduced by -10% per day of delay.

Outline. Another programming exercise: 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: $\{0, 1\}^2 \rightarrow \{0, 1\}$ given by XOR(0, 0) = XOR(1, 1) = 0; XOR(1, 0) = XOR(0, 1) = 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.

Historical remark. The XOR problem played a very big role in the history of neural networks and machine learning. I will say some more about that in the lecture, but here is the upshot:

- the original Perceptron of Rosenblatt was for several years believed (and hyped) to be the key to achieving artificial intelligence
- in 1969 Marvin Minsky and Seymore Papert published a book (*Perceptrons: an introduction to computational geometry*) where among many other things they pointed out that the original Perceptron cannot learn the XOR function.
- This completely deflated the first neural networks hype and sent the field into a hibernation phase that lasted until the early 1980s, when MLPs plus backprop appeared on stage.
- For many years, learnability of the XOR function remained a touchstone for machine learning systems – when some researcher wanted to advertise a new learning algorithm, s/he had to show it could cope with XOR.
- Today the bar is raised higher: when some researcher wants to advertise a new learning algorithm, s/he minimally has to show it can classify handwritten digits and characters (from the MNIST corpus, which is *much* more challenging than our course's digits dataset, see https://en.wikipedia.org/wiki/MNIST_database).

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: $[-1, 1]^2 \rightarrow \{0,1\}$, defined by $f(x, y) = \text{sign}(xy)$. 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: $\{0,1\}^n \rightarrow \{0,1\}$, $P(x) = 1$ if the number of 1's in the Boolean vector x is even, else = 0. 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.