Neural Networks 2018: Assignment 0

February 7, 2018

The purpose of this assignment is to introduce you to some of the important concepts that you will get to explore during this course. It will also help you to master Python 3. We will start with a classification problem and secondly look at (polynomial) regression. You will get a feel for working with matrices using NumPy and plotting using Matplotlib.
This assignment is not graded. You can work on this assignment during the practicum on the 14th of February (but feel free to start earlier) and we will be there to answer your questions. Furthermore, a blackboard discussion forum has been opened where you can post your questions (anonymously if you want).

1 Classifying boolean functions

Consider the functions AND and XOR with the input-output table below:

x_1	x_2	AND	XOR
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

1. Create, using NumPy, a matrix X containing x_1 and x_2, and 2 separate vectors to hold the values for the AND and XOR function results. We will create an image of the *feature space*: use matplotlib to create two figures in which you plot the 4 points from X in $\mathbb{R}^2$, labeling them to reflect the function values (0 or 1). Make one figure for the XOR function and one for the AND function.

2. Initialize a vector w for *weights* and a scalar b as *bias* at random using **numpy.random.randn** and calculate $\hat{y} = w^T X + b$.

   Then, for each element in $\hat{y}$, calculate the Heaviside step function:

   $$H(n) = \begin{cases} 0, & n \leq 0 \\ 1, & n > 0 \end{cases}$$

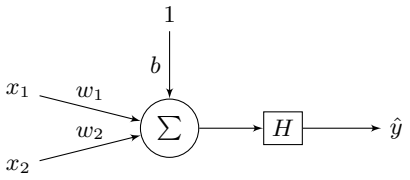
   You can use **numpy.heaviside(x, 0)**.

   Play with the values of w and b until you correctly predict the AND function. Can you do the same for XOR? Why, or why not?

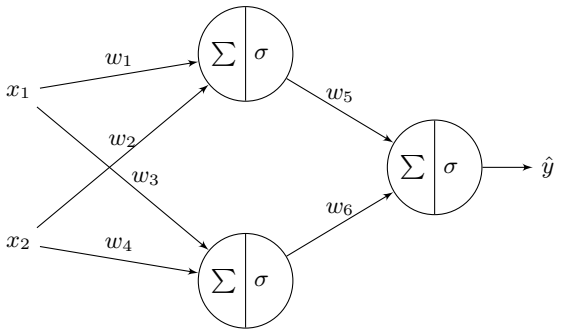
3. Try to plot the line that separates the 1's from 0's, only for the figure containing the outcomes of the AND function. We will call this line the *decision boundary*. Start by creating a meshgrid with the **numpy.meshgrid** function. Then, assign a label to each point on the grid using your function, forming a prediction matrix. The prediction matrix should thus have the same size as the grid. Then use the **matplotlib.pyplot.contour** function to plot the decision boundary over your figure, where the first two arguments should be the x and y of the meshgrid, and the third argument the prediction matrix.

   Observing the plots you see that such a line can not be drawn for the XOR figure, what does this mean?

4. The function in 1.2 can be graphically represented as a graph:



   This representation helps us to think about it in a more abstract way. We can also represent more complex functions using this graph-like structure. We will now try to find a correct prediction for the XOR function. For this we need a slightly more complex *model*:



   Here we use the outputs of the first two functions as the input for the last function. This way we create a more complicated function. Instead of the Heaviside function, we will now use the logistic function:

$$\sigma(n) = \frac{1}{1 + e^{-n}}$$

   This has several advantages, which will be discussed in a later lecture. The logistic function (or *activation function*) is drawn inside the node, which gives us a more compact representation. For now we omit the scalar b.

   Use the **numpy.random.randn** function to set the values for $w_1, w_2 \ldots, w_6$ and calculate $\hat{y}$ as illustrated in the above figure. Similar to the Heaviside

function, use a threshold of 0.5 after the last logistic function to predict whether a value should be 0 or 1. Repeat this process 10^6 times. How many times can you correctly predict XOR? What does this mean for more complex classification tasks?

5. Can you think of a way to find a correct prediction for the XOR function using only one function as in 1.1?

2 Polynomial curve fitting

We will fit a polynomial to training data and validate our results on a test set. This is a common approach in machine learning to test if your model generalizes well. You will get an idea of what overfitting is and how the size of the training set influences this phenomenon. If you need some more intuition for the problem, you can already take a look at the slides of lecture 2.

Consider the function $y(x) = 0.5 + 0.4\sin(2\pi x),$ for x in $[0, 1]$.

1. Use this function to generate two noisy sets of n points (train and test) that will be used for modeling y, for $n = 9, 15, 100$. The x-coordinates should be uniformly (at random) distributed over $[0, 1]$, y-coordinates should be contaminated with Gaussian noise with $\mu = 0, \sigma = 0.05$.

   Hint: You can use the **numpy.random.uniform** and **numpy.random.normal** functions.

2. Find the best polynomial approximation of degree d ($d = 0, 1, \ldots, 9$) of the training set and plot the results.

   Hint: check the **numpy.polyfit** and **numpy.polyval** functions

3. Generate 3 plots (one for each value of n) that demonstrate the approximation error (MSE) on the train and test sets as a function of the polynomial degree. Generates all these 3*10 + 3 figures.

4. You should see that higher degree polynomials can fit well to the training data, but can have problems to generalize to the test data when there is not enough data. This is called overfitting. One way to avoid this is to enforce the absolute values of our model parameters, the coefficients of the polynomial, to be relatively small. One option is to add the sum of squared coefficients $\lambda \sum_{i=0}^{M} w^2$ to the error function. Here λ is a tunable parameter that controls the size "punishment" for too big values of coefficients.

   Think, or search in the literature, how to minimize the error function including the regularization term.

   Hint: the error function is a quadratic function of weights, so its gradient is given by linear functions of weights. Therefore, finding the minimum of the error function reduces to solving a system of linear equations.

5. Now set $d = 9$. Play with different values for λ and plot the error on the train and test set as a function of λ for each value of n (3 figures).