Introduction to Deep Learning
Assignment 0

September 2020

Introduction

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 familiarize yourself with Python 3 and using Jupyter notebooks. Additionally, you will get a feel for working with matrices using NumPy and plotting using Matplotlib. The assignment consists of two parts: polynomial regression and a problem involving high-dimensional spaces.

You can work on this assignment during the practicums on Fridays and we will be there to answer your questions (either at Snellius lab rooms or online). Furthermore, you can post questions on the discussion board on BrightSpace.

Deliverables: You should form a group consisting of 2 people on BrightSpace and submit a single Jupyter notebook containing your solutions. **This assignment will not be graded!**

1 Polynomial curve fitting, regularization

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.

Note: You should check another file on BrightSpace (*A0_supplement*) which demonstrates some of the plots that you should generate in this task along with additional information on the problem. Consider the function:

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

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, ..., 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 only a small number of samples. 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 penalty for coefficients that have exceedingly large values.

Think, or search in the literature, how to minimize the error function including the regularization term.
You should also look up the *sklearn.linear_model.Ridge* function.

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

2 High dimensional spaces

This task will help you to develop some intuitions about shapes in high dimensional spaces. Some of the results are counter-intuitive and this is a valuable skill that will help you understand more advanced concepts later on in the course.

Consider a unit cube in n-dimensional space $U_n = [0,1]^n$, and an n-dimensional unit ball B_n that is included in U_n:

$$B_n = \{ (x_1, x_2, ..., x_n) \mid (x_1 - 0.5)^2 + (x_2 - 0.5)^2 + ... + (x_n - 0.5)^2 < 0.5^2 \} \quad (1)$$

for $n = 1, 2, ..., 100$:

1. Find the number of vertices ("corners") of U_n, **Corners(n)**
2. Calculate the length of the longest diagonal of U_n, **DiagU(n)**
3. What is the volume of B_n and how does it change as the number of dimensions increases? Estimating it is not entirely straightforward (you can look up the formula and implement it if you wish). However you can approximate this procedure by generating random points, uniformly distributed within U_n (points=rand(10^6,n)) and checking what percentage of them are contained within B_n.
4. Calculate the volume of the "0.01-skin" of U_n :
   VolumeS(n) $= 1^n - (1 - 2 * 0.01)^n$
5. For $n = 2, 4, 8, ..., 1024$ generate 1000 points in U_n, uniformly distributed, find distances between all pairs of these points, and produce a histogram of these distances.