Prof. Dr. Jörg Müller
Arthur Fleig

AI VIII – Serious Games

Exercises for **Programming, Data Analysis, and Deep Learning in Python** (SoSe 2021)

Exercise Sheet no. 11, *Deadline*: Monday, July 5, 10:15

Notes

- Pay attention to the notes on the previous sheet.

Exercise 31 Polynomial Regression with Least Squares (programming ex.) (12 points)

Download the corresponding Python script from E-Learning. The script contains data points (x_i, y_i), $i = 1, \ldots, m$. The goal is to approximate the given data set by a polynomial of degree $n < m$ using the Least Squares method. To this end, complete the following tasks:

1) For a given degree n, create either of the following two matrices (m rows, $n+1$ columns or $n+1$ rows, m columns) using the x-coordinate of the respective data point:

$$A := \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^n \end{pmatrix} \quad \text{or} \quad A^\top := \begin{pmatrix} 1 & \ldots & 1 \\ x_1 & \ldots & x_m \\ x_1^2 & \ldots & x_m^2 \\ \vdots & \ldots & \vdots \\ x_1^n & \ldots & x_m^n \end{pmatrix}.$$

2) Calculate the matrix $A^\top A$ and store the result in the variable `ATA`. From the y-coordinates of the data points, construct the vector $b := A^\top y$.

3) Solve the system of linear equations $A^\top Aa = b$ for a in two different ways:
   - directly, using `np.linalg.solve`, and
   - by calculating the inverse of $A^\top A$,

   and store the results in variables called `a_direct` and `a_via_inv`, respectively. (The vector $a = (a_0, a_1, ..., a_n)$ contains the coefficients of the approximating polynomial.)

4) Plot the data points (x_i, y_i) as single (unconnected) red dots and also display both approximating polynomials in that plot.
   Hint: To plot a polynomial $p(x)$, first, create a grid `x_fine` *with* `np.arange(...)` *for suitable parameters. Then evaluate the polynomial on that grid (you may use the function* `f_eval` *for that) and store the result in a list or array* `f_x`*. Finally, use* `x_fine` *and* `f_x` *in the* `plt.plot()` *command to plot the polynomial.*

5) Calculate the errors $\|Aa - y\|_2$ for a=`a_direct` and a=`a_via_inv`.

Try out different polynomial degrees $n = 1, ..., 15$. Does the error for `a_direct` always decrease with higher n? Briefly describe your observations regarding the error for `a_via_inv`.

Exercise 32 Predicting Diabetes (programming exercise) (12 points)

Download the associated file[1] from E-Learning. In this exercise we will try to predict whether a person has Diabetes using logistic regression.

a) Pre-process the data as follows. First, read the csv file, then divide the columns into two types of variables: The *target variable* (also called *dependent variable*) is the last column, which you should store in the variable `y`. The *feature variables* (also called *independent variables*) are all other columns. Store these in the variable `X`.
Next, as in the lecture, divide `X` and `y` into training data (`train_set_x, train_set_y`) and test data (`test_set_x, test_set_y`) using 75% of the data as training data and the other 25% as test data.

b) Copy all the functions that you need for logistic regression from the lecture notes (including `initialize_parameters(dim)`) and modify the following: In the `propagate` function, after the `cost` is calculated, check whether it is `NaN` and if so, change it to `np.inf`. In the `optimize` function, every 10000 steps (instead of every 100 steps), append the `cost` to the `costs` list and output the quadratic Euclidean norm of the gradient of the cost: `np.sum(grads['dw'] ** 2)+ np.sum(grads['db'] ** 2)`

c) As in the lecture notes, call the `model` function to run the logistic regression and then plot the costs.
Important: Set `np.random.seed(0)` before the pre-processing step. When calling the `model` function, set the number of steps to 1000001 and the learning rate to 0.00025.
Hints: When calling the `model` function, you might need to fix any errors regarding the dimension of arrays. You can do so by comparing the shapes of the respective arrays to the ones from the lecture.

d) Output the so-called *confusion matrix*. It displays the number of correct predictions on the diagonal and the incorrect predictions on the off-diagonal, similar to:

		Predicted	
		0	1
Actual	0	118	12
	1	26	36

To output it, proceed as follows: First, import `metrics` from the `sklearn` module: `from sklearn import metrics`. Then use `predict` to calculate the predictions on the test set. Next, call `cnf_matrix = metrics.confusion_matrix(arg1, arg2)`, where you should replace `arg1` by the test set of y and `arg2` by the predictions on that test set. Finally, `print(cnf_matrix)`.

e) Perform the tasks c) and d) with
   - a learning rate of 0.00025, once with 100001 steps and once with 10001 steps,
   - a learning rate of 0.0002, once with 1000001 steps, once with 100001 steps, and once with 10001 steps,

   and compare the corresponding confusion matrices.

[1] Source (2020-01-14): `https://github.com/plotly/datasets/blob/master/diabetes.csv`