Assignment 4

Prof.	Panos Trahanias	trahania@csd.uoc.gr
T.A.	Emmanouil Sylligardos	sylligardos@csd.uoc.gr
T.A.	Despina - Ekaterini Argiropoulos	despargy@csd.uoc.gr
Deadline	**Thursday 23:59, 18th of May 2023**	

Note: This is a personal assignment and should be pursued individually and without the use of any computerized AI facilities. Note that this will be automatically checked for every delivered assignment. The assignment should be implemented entirely in Google Colaboratory, following the delivered instructions below.

Note: The pseudo-code and the theory for the following algorithms are written in the book's fifth chapter uploaded on the course's website. This assignment is very demanding and no code will be given (no coding tutorial, only theoretical). Make sure to read and understand the theory before any coding. All the algorithms presented in this assignment are the building blocks of Deep Learning and Neural Networks.

Question 1: Single-sample Perceptron (50/100)

The scope of this exercise is to get you familiar with training a linear classifier with Gradient Descent.

Data: For this exercise, you will use the dataset in the *dataset.csv* file that consists of 1000 2-dimensional data samples and their labels.

Equation:

$$learning\_rate_i = 1/\sqrt{k_i} \tag{1}$$

where $k_i = k_{i-1} + (i \mod n)$, $k_0 = 0, i >= 1$, i = iteration index, n = number of samples.

!Notice: You are not allowed to use library functions, except for matrix multiplication and plotting, everything else should be developed from scratch.

Questions:

1. Create a scatter plot of the dataset. Is the dataset linearly separable? Justify your answer in 1-2 sentences.
2. Create a function called 'train_single_sample' that implements the Fixed-Increment Single-Sample Perceptron algorithm. The arguments of the function should be:
   (a) a := weights + bias (bias trick)
   (b) y := data + '1's (bias trick)
   (c) labels
   (d) n_iterations := the number of iterations/updates
   (e) lr := learning rate
   (f) variable_lr := boolean

   and the function should return:

   (a) a := trained model
   (b) acc_history := list of accuracy values

   The algorithm should be implemented according to the notes. Extra functionalities include computing and printing the model's accuracy every *n_samples* iterations. Also, you should save and return only the best model. **Bonus 2.5%**: Use the tqdm library to print the progress of the model along with the accuracy during training.

3. Create a function called 'plot_model' that takes as input the trained weights (+ bias), the data, and the labels and returns a scatter plot with the decision boundaries of the model (**Hint**: Use contourf)
4. Train a linear model using the functions you have implemented. Use the following hyperparameters:
   (a) n_iterations = 100000
   (b) lr = 100000
   (c) variable_lr = False
5. Plot the history of the accuracy during training.
6. Plot the data along and the trained model. Use the 'plot_model' function you implemented before.
7. Now we are going to retrain our model but with a variable learning rate. Create a 'Scheduler' class that implements a function 'get_next_lr' that every time it is called returns the next learning rate. The object should work according to the function at the beginning of the assignment. Test your object by initializing it and plotting the learning rate over 100 steps. Now configure the training function to use this object when 'variable_lr = True'
8. Retrain the model with a variable learning rate. Plot the dataset and the trained model as before. What is the main difference compared to training with a fixed learning rate? What method do you think is better? Justify your answer.

Question 2: Batch Perceptron (50/100)

The **data**, the **equation**, and the **notice** from the previous question apply to this one too.

Question:

1. Create a function called 'train_batch' that implements the Batch Perceptron algorithm. The arguments of the function should be:
   (a) the same as the 'train_single_sample' in Question 1 +
   (b) theta := the value for the theta criterion
   (c) batch_size

   and the function should return:
   (a) the same as the 'train_single_sample' in Question 1 +
   (b) error_history := list of error values

   The algorithm should be implemented according to the notes. Extra functionalities include computing and printing the model's accuracy and the error every $\frac{n\_samples}{batch\_size}$ iterations. Also, you should save and return only the best model. The error here is the absolute sum of the updating step, that is the value we use to update the weights. **Bonus 2.5%**: Use the tqdm library to print the progress of the model along with the accuracy and the error during training.
2. Train a linear model using the function 'train_batch' you have implemented. Use the following hyperparameters:
   (a) n_iterations = 100000
   (b) theta = 0.01
   (c) batch_size = 16
   (d) lr = 100000
   (e) variable_lr = False
3. Plot the history of the accuracy and the error during training. Use the plt.subplots function.
4. Plot the data and the trained model. Use the 'plot_model' function you implemented before.
5. Retrain the model with a variable learning rate. Use the Scheduler you implemented in Question 1. Plot the dataset and the trained model as before. What do you notice now? Is the difference between training with a fixed and a variable learning rate the same as before? Justify your answer.

Bonus Question: Pythonic Implementation (+10/100)

Use only two for/while loops throughout the whole assignment.

Deliverable

This assignment should be implemented entirely in Google Colaboratory. Google's Notebook allows you to combine executable Python scripts with rich text in a single document. Your deliverable should be a single '.ipynb' file along with its corresponding .py file (both can be easily exported from Google Colaboratory). Every single question should be implemented in a single code block. Code blocks should be clearly and shortly explained (you may use the text boxes for that goal). Use library functions **only** for matrix operations and plots.

Submission instructions

- To ask questions, email hy473-list@csd.uoc.gr with the subject "[CS473]: Assignment 4 question".
- To submit your implementation, email sylligardos@csd.uoc.gr and despargy@csd.uoc.gr with the subject "[CS473]: Assignment 4 submission".
- You should submit only the files '.ipynb' and '.py' in a zipped folder (.zip) with the name *"hw4_<am>"* where *am* is your university identification number. The folder should contain nothing else than the two aforementioned files.
- The names of the submitted '.ipynb' and '.py' files should be *"hw4_<am>.ipynb"* and *"hw4_<am>.py"* respectively.
- The whole '.ipynb' file should run when selecting 'Runtime → Run all' without any problem. Meaning that any unnecessary code blocks should be removed before submitting. Code blocks that can not run will not be graded. If they prevent the smooth execution of the file, they will have a negative impact on the grade of the assignment.