Perceptron Update Implementation
Assignment 4

Instructor	Iason Oikonomidis	oikonom@csd.uoc.gr
T.A.	Giorgos Karvounas	gkarv@csd.uoc.gr

Deadline **Thursday 23:59, 17th of May 2024**

Abstract

The goal of this assignment is some hands-on experience with the single-sample and batch Perceptron update rules. Specifically, you are called to implement two functions, one for each case. Then you are called to use them on the provided dataset, experiment with alternative strategies, and describe the resulting behavior.

Note: This is a personal assignment and should be pursued individually and without use of any computerized AI facilities. Note that there will be a face-to-face examination after the delivery. The assignment should be implemented entirely in Google Colaboratory following the instructions below. *Use library implementations only for linear algebra and plotting, unless explicitly stated otherwise*.

Question A: Single-sample Perceptron (55/100)

The scope of this exercise is to get you familiar with training a linear classifier with Gradient Descent.
Data: use the dataset in the dataset.csv file with `n_samples = 1000` 2-dimensional data samples and their labels.

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 (augmentation trick for the weight vector)
   (b) `y` : data + '1's (agreeing with the augmentation above)
   (c) `labels`
   (d) `n_iterations` : the number of iterations / updates to perform
   (e) `lr` : learning rate
   (f) `variable_lr` : boolean (reserved, default `False`, see step 7. below)
   and the function should return
   (a) `a` : trained model
   (b) `acc_history` : list of accuracy values
   The algorithm should be implemented according to the course slides, along with these additional functionalities and modifications:
   (a) compute and print the current model accuracy every `n_samples` (ie after an *epoch* in modern lingo), and
   (b) save and return the best model overall, **not** the one at the last iteration.
   Since training may be time-consuming, it would be helpful to display a progress bar using `tqdm` (not graded).

Release date: Wednesday, 24rth of April 2024
Deadline: Thursday 23:59, 17th of May 2024

3. Create a function called '`plot_model`' that takes as input the trained weights and bias, the data, and the labels, and displays a scatter plot with the decision boundaries of the model. You may find the `plt.contourf` function useful.
4. Train a linear model using the functions you've implemented. Use the following hyperparameters:
   (a) `n_iterations = 100000`
   (b) `lr = 1`
   (c) `variable_lr = False`
5. Plot the history of the accuracy during the training process.
6. Use the '`plot_model`' function you implemented to plot the data along with the trained model.
7. Let's now use a variable learning rate to retrain the model. Use this recursive rule to compute the learning rate:

$$\text{learning\_rate}_i = 1 / \text{sqrt}(k_i),\ k_i = k_{i-1} + (i \bmod n),$$

   with $k_0 = 0$, $i > 0$ the current iteration number, and n = `n_samples` (= 1000 for our dataset). Feel free to implement this in *any way you are more familiar with*. Here's a structured way: create a '`Scheduler`' class that implements a function '`get_next_lr`' which returns the next learning rate every time it is called. This function should implement the recursive rule above. Test your code by initializing the object and plotting the learning rate over 100 steps. Now configure the training function to use this object when '`variable_lr = True`'. A proper implementation would accept the object as a named argument, but for this assignment you can use a global object if you find it easier.
8. Retrain the model with a variable learning rate as implemented above. Plot the dataset and the trained model as before. Also plot the accuracy histories. What is the main difference compared to training with a fixed learning rate? What method do you think is better? Justify your answer.

Question B: Batch Perceptron (55/100)

Use the same data and variable learning rate as in the previous 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`' function in Question A
   (b) `theta` : the value for the theta criterion
   (c) `report_period`
   and the function should return:
   (a) the same as the '`train_single_sample`' function, and also
   (b) `error_history` : list of error values
   The algorithm should again be implemented according to the course slides. Additional functionalities and modifications include:
   (a) compute and print the model accuracy and error every `n_samples / report_period` iterations, and
   (b) save and return only the best model.
   The error here is the absolute sum of the update step, that is, the value we use to update the weights. Again, you may find it useful to employ the `tqdm` progress bar library.
2. Train a linear model using the function you've implemented. Use the following hyperparameters:
   (a) `n_iterations = 100000`
   (b) `theta = 0.01`
   (c) `report_period = 16`
   (b) `lr = 1`
   (c) `variable_lr = False`
3. Plot the history of the accuracy and the error during training. You may find the `plt.subplots` function useful.
4. Plot the data and the trained model. Use the '`plot_model`' function you implemented before.

Release date: Wednesday, 24rth of April 2024
Deadline: Thursday 23:59, 17th of May 2024

5. Retrain the model with a variable learning rate. You may reuse the Scheduler object if you implemented it in Question A. 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? Again, justify your answer.

Deliverable

This assignment should be implemented entirely in Google Colaboratory. Your deliverable should be a single .ipynb file (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). **Only use library functions for arithmetic operations, matrix operations and plots, unless explicitly stated.**

Release date: Wednesday, 24rth of April 2024
Deadline: Thursday 23:59, 17th of May 2024