CMPUT 328 Fall 2019
Assignment 3

Logistic Regression [Worth 5.7% of the total weight]:

Implement Logistic Regression in PyTorch.

a) Implement Logistic Regression on the **MNIST** dataset and **CIFAR10** dataset.
b) Add a regularization term to improve your model (**L1 or L2 regularization, whichever gives better accuracy**)
c) Find out hyperparameters value for regularization using **the validation-set**

- For this assignment, the validation set will be the last 12000 samples of the train set for both MNIST and CIFAR10. You should do hyperparameters search using the accuracy on the validation set and select the best configurations for the two datasets.
- **You will define your own data using PyTorch data-loaders.**
- You should not touch the test set during this process. You need to implement the above algorithms for both **MNIST** and **CIFAR10** datasets. Note that MNIST images are grayscale, while CIFAR10 images are RGB (color). Also, note that MNIST and CIFAR10 images have different spatial dimensions (height and width).

NOTE: A correct implementation and somewhat well-tuned version of this algorithm will have an accuracy of 92-94% on MNIST and 38-40% on CIFAR10 for both test and validation sets.

DUE DATE: The due date is Friday, October 4th at 11:55 pm. The assignment is to be submitted online on eclass. For late submissions' rules please, check the course information on eclass

SUBMISSION: You need to submit one file of code: logistic_regression.py that contain the logistic regression function similar to the Assignment 1, and two other files .txt that will be generated by your code where it contains the predictions over the test_set of Mnist and Cifar-10 each on their own. So they can be used for the final marking. Also, import any additional libraries you need in the file as well.

Skeleton Code: The first file named assignment_3_notebook.ipynb is a Jupyter notebook for you to work on. Upload this file to Google Colab to use it. This file contains the template code for this assignment and a function with signature *def logistic_regression(dataset_name, x_train,*

y_train, x_valid, y_valid, x_test) which you need to implement. There is also another skeleton_code.zip where you will find logistic_regression.py and main.py in case you wanna use your own machine.

MARKING:

25% Marks Describe your program for TAs (Sep 30 – October 4)

TAs can select five random questions based on your code, results, and algorithms. The time to present will be in your lab section in the week when this lab is due. Note that you must present this part to your TA in your designated lab section. *You will not get any marks for this part if you don't present in your own lab section.*

75% Marks Your final score depends on the correct implementation of algorithms and it must work on both MNIST and CIFAR10 datasets.
There is no time constraint for this assignment. However, your code should run in a reasonable amount of time. One trick to improve your run time during testing is: Grid search the hyperparameters first but only put in the best hyperparameters you found for submission. *Each dataset will give you 50% of the total score in this assignment.*
MNIST: score scales linearly from 82% to 92% on the test set
CIFAR10: score scales linearly from 28-38% on the test set

Main Concepts:

1) Loss Function

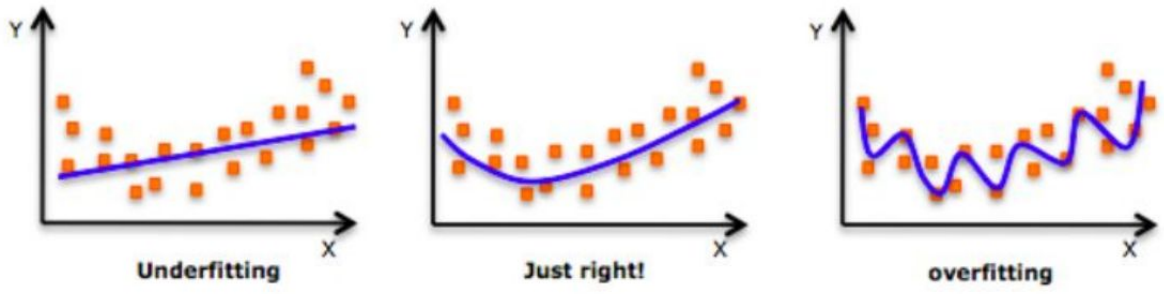
In order to train our model, we first need to define an indicator to evaluate this model is good. In fact, in machine learning, we usually define indicators to indicate that a model is bad, this indicator is called loss, and then try to minimize this indicator. A very common loss function is cross-entropy. Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverge from the actual label.

$$Loss = -\sum_{i=1}^{M} y_i \log \acute{y}_i$$

M - the number of classes
log - the natural log
y - the actual distribution (our one-hot vector).
y’ - our predicted probability distribution.

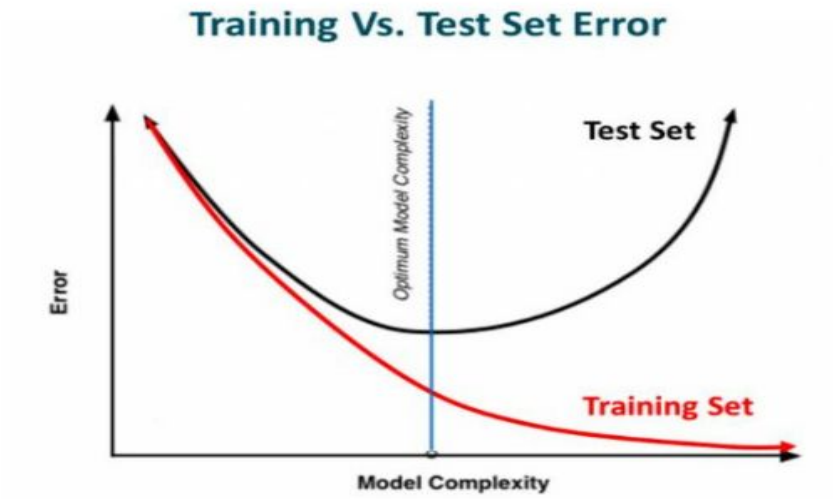
2) Regularization

Before we deep dive into the topic, take a look at this image:



As we move towards the right in this image, our model tries to learn too well the details and the

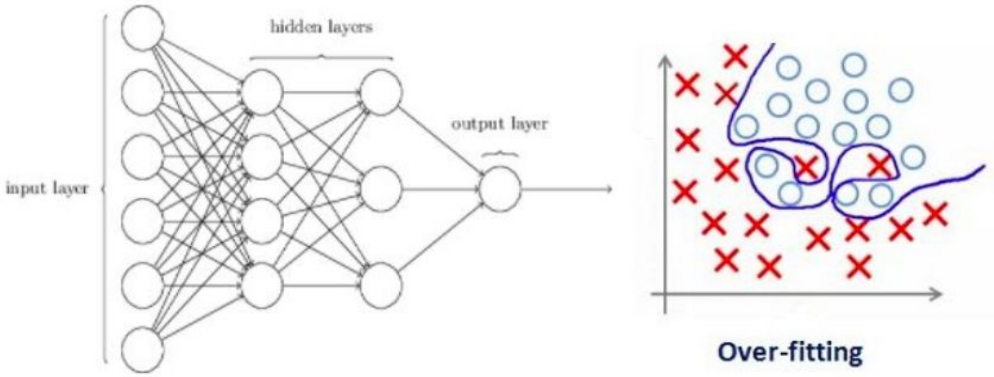
noise from the training data, which ultimately results in poor performance on unseen data. In other
words, while going towards the right, the complexity of the model increases such that the training error reduces but the testing error doesn't. This is shown in the image below.



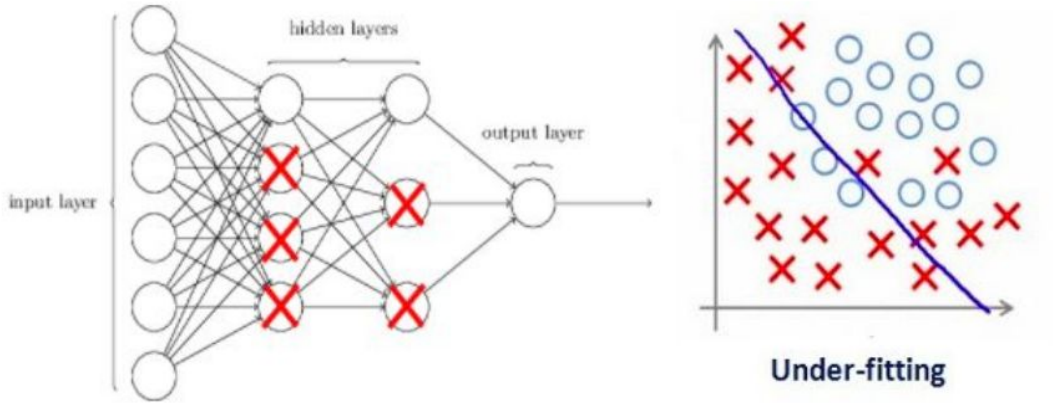
If you've built a neural network before, you know how complex they are. This makes them more prone to overfitting. Regularization is a technique which makes slight modifications to the learning algorithm such that the model generalizes better. This, in turn, improves the model's performance on unseen data as well.

How does regularization help reduce overfitting?

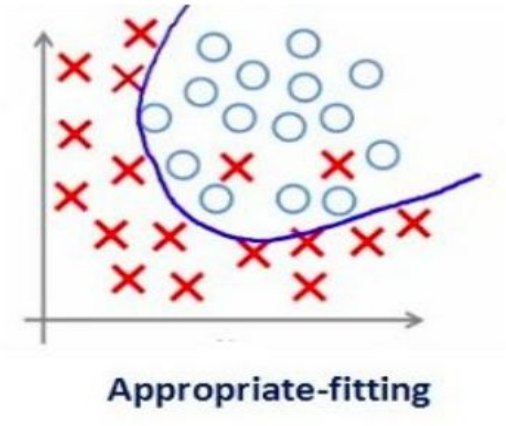
Let's consider a neural network (a more general version of logistic regression) which is overfitting on the training data as shown in the image below.



If you have studied the concept of regularization in machine learning, you will have a fair idea that regularization penalizes the coefficients. In deep learning, it actually penalizes the weight matrices of the nodes. Assume that our regularization coefficient is so high that some of the weight matrices are nearly equal to zero.



This will result in a much simpler linear network and slight underfitting of the training data Such a large value of the regularization coefficient is not that useful. We need to optimize the value of the regularization coefficient in order to obtain a well-fitted model as shown in the image below.



Different Regularization Techniques in Deep Learning:

Now that we have an understanding of how regularization helps in reducing overfitting, we'll learn a few different techniques in order to apply regularization in deep learning.

L2 & L1 regularization

L1 and L2 are the most common types of regularization. These update the general cost function by adding another term known as the regularization term.

Cost function = Loss (say, binary cross entropy) + Regularization term

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to quite an extent.

However, this regularization term differs in L1 and L2.
In **L2**, we have:

$$cost\ function = Loss + \lambda \times \sum \|W\|^2$$

Here, **lambda** is the regularization hyperparameter, whose value is optimized over the validation set for better results. L2 regularization is also known as weight decay as it forces the weights to decay towards zero (but not exactly zero).

In **L1**, we have:

$$cost\ function = Loss + \lambda \times \sum \|W\|$$

In this, we penalize the absolute value of the weights. Unlike L2, the weights may be reduced to zero here. Hence, **it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it**.