**CSE641 Deep Learning**
**Assignment-1 [65 marks]**
**Deadline: 11th Feb 2023 EOD**

**General Instructions:**

**Use of any Deep Learning libraries like Pytorch or TF is not allowed. You can use other python libraries like numpy, scipy, matplotlib.**

1. Each group member must do at least one of the following tasks. But all should know the working of all the tasks. (Recommended: Divide the sections among yourselves.)
2. For Plagiarism, institute policies will be followed strictly.
3. **Make sure to use Pickle or any other library to save all your trained models. There will not be enough time during the demo to retrain your model. This is a strict requirement.** You must upload your best models on Classroom to reproduce your results during the demo. If you are not able to reproduce your results during the demo then no marks will be given. You are advised to prepare a well-documented code file.
4. You need to submit Output.pdf, Code files (it should include both .py files and .ipynb files), and models dumped after training. Mention your sample outputs in the output.pdf. The Output.pdf should enlist all hyperparameters clearly for quick reference.
5. Submit code, models, and output files in ZIP format with the following name: A1_Member1_Member2_Member3.zip

## *Part-1: PTA*

**Implement a Perceptron Training Algorithm (PTA) [15 Marks]**

PTA updates weights when a mistake is made (DL Intro Slides 35). Follow the basic algorithm outlined in class, we require you to write a code and show the results for:

a. Calculate the number of steps (i.e. weight updates) necessary for convergence for the following operations using: **[2*3] marks**
   i. Two variables: AND
   ii. Two variables: OR
   iii. One variable: NOT
b. Draw the decision boundary at each step of learning for all 3 operations (AND, OR, NOT). **[2*3] marks**
c. Theoretically, PTA can only converge on linearly separable data as observed in 1a. Demonstrate via code that PTA cannot compute XOR operation. How many steps do you need to take to prove it? You can either prove this by showing the output or printing the decision boundary at each step. **[3] marks**

## *Part-2: Gradient Descent*

**Implement Gradient Descent Algorithm from scratch [25 marks]**

Consider the following optimization problem:-

Min $_{x_1, x_2}$ $f(x_1, x_2) = x_1^2 + \gamma x_2^2 - x_1 {*} x_2 - x_1 - x_2$

If we assume $\gamma = 1$ for the parts [a-d], then :

a. Write a function that implements the gradient descent algorithm (its subparts will involve differentiation and weight updation). **[4] marks**

b. Using some initial values of x1 , x2 and step size = 0.1 as initial conditions, report the number of iterations required for convergence. **[2.5] marks**
c. Plot the iteration v/s value of the function **f** in that iteration for the above setup. (x-axis = iteration no., y-axis = value of the function **f** in that iteration). **[2.5] marks**
d. Arbitrarily choose 5 different values of step size in the interval [0,2]. Find out the number of iterations required to converge for each step size and plot step size v/s iterations graph (x-axis = step size, y-axis = iterations required to converge). You must keep the initial condition of x1 and x2 the same in all the cases. **[6] marks**
e. Arbitrarily choose 5 different values of γ in the interval [0,1] and find out the number of iterations required to converge for each value of γ. Plot γ v/s iterations graph (x-axis = γ, y-axis = iterations required to converge). You must keep the initial condition of x1 and x2 the same in all the cases. **[6] marks**
f. Run your gradient descent implementation for γ = -1 and plot Iteration v/s value of the function **f** in that iteration graph (x-axis = Iteration number, y-axis = value of the function **f** in that iteration). What do you observe? Can you think of an explanation for the observed behavior? …Hint: think about the kind of curve a gradient descent draws. **[2.5 marks + 1.5 marks]**

## ***Part-3: MLP***

**Implement a Multilayer Perceptron from scratch [25 marks]**

Implement a modular Deep Learning Toolkit for training a multi-layer perceptron on the given Fashion-MNIST dataset.

Create the neural network using the descriptions given below. **Implement forward propagation and backward propagation from scratch on the constructed neural network.** You are required to create the toolkit.py file.

Network Structure: Input 784, output 10. You are free to use any number of hidden layers (>=1) (FNN) with any number of neurons. **[4 for forward + 8 for backward] marks.** After choosing appropriate sizes of hidden layers, learning rate and other hyperparameters. Analyze the model and report results for various combinations of the following:

a. Activation: Try using the activation functions for all hidden layers as either Sigmoid or ReLU. Output layer will always have Softmax activation. For each of the activation and its differential, write your own implementation of it. **[4 marks]**
b. Weight Initialization: Try using the initial functions for all layers as either all zeros or Normal/Gaussian. **[4 marks]**
c. For the 4 combination of activation and weight initialization that you try above generate plots for: **[5 marks]**
    - Loss plot - Training Loss and Validation Loss V/s Epochs.
    - Accuracy plot - Training Accuracy, Validation Accuracy V/s Epochs)
    - Analyze and Explain the plots obtained