Deadline: 31 Jan, Tue, 23h59

Problem:

Aim of this assignment is to make you implement and understand the backpropagation algorithm. You have to implement a classifier for the task of handwritten digit recognition on the MNIST dataset with standard split.

- You have to code a simple multilayer perceptron from scratch, in a programming language of your choice, to solve the above task. Your code should take the number of hidden layers, number of nodes and the type of activation function, in each layer, as arguments.
- You also have to show that the gradients computed by backpropagation are correct, by calculating them numerically for all the weights and biases and showing them to be equal.
- You need to implement and experiment at least two of following activation functions:
  1. ReLU
  2. Maxout
  3. Tanh
- Also implement at least two of following gradient methods:
  1. GD with momentum
  2. Adam
  3. Adagrad
  4. RMSprop
- Finally, you have to experiment with different architectures i.e. how changing the number of layers, nodes in layers and different activation functions affect the performance of the network and plot their train and validation error graphs.

Submission:

- Submit the assignment in a zip format with name as "*yourrollno_A1.zip*"
- Your assignment will be graded on the cleanliness and readability of your code and the accuracy of your designed model. Also provide a README file explaining essential parts of you code and experiments (setup, graphs, findings etc.).

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# The implementation could follow the following pattern

# initialize the MLP
my_mlp.init(nhidden_layers, nnodes, actfun)
# where
# nhidden_layers   int 1x1
# nnodes    int  1x nhidden_layers
# actfun   string or int to index to activation function types

# train the MLP; plot the train and val errors and performances
my_mlp.train(train_data, train_labels, val_data, val_labels)

# test the MLP
my_mlp.test(test_data, test_labels)

# plot the backprop and the numerical gradients for all params
over a given mini batch
my_mlp.plot_grads(train_data[mini_batch])
```