Due Date: February 16th, 2019
Instructions

- *For all questions, show your work!*
- *Submit your code and your report (pdf) electronically via the course Gradescope page. If you use the Jupyter notebook, please export it as pdf and submit via Gradescope.*
- *You should push your code to a Github repository, and include the link to the repository in your report!*

Problem 1

In this problem, we will build a Multilayer Perceptron (MLP) and train it on the MNIST handwritten digit dataset [1].

Building the Model [35] Consider an MLP with two hidden layers with h^1 and h^2 hidden units. For the MNIST dataset, the number of features of the input data h^0 is 784. The output of the neural network is parameterized by a softmax of $h^3 = 10$ classes.

1. Build an MLP and choose the values of h^1 and h^2 such that the total number of parameters (including biases) falls within the range of [0.5M, 1.0M].
2. Implement the forward and backward propagation of the MLP in numpy without using any of the deep learning frameworks that provides automatic differentiation. Use the class structure provided here[2].
3. Train the MLP using the probability loss (*cross entropy*) as training criterion. We minimize this criterion to optimize the model parameters using *stochastic gradient descent*.

In the following sub-questions, please specify the *model architecture* (number of hidden units per layer, and the total number of parameters), the *nonlinearity* chosen as neuron activation, *learning rate*, *mini-batch size*.

Initialization [10] In this sub-question, we consider different initial values for the weight parameters. Set the biases to be zeros, and consider the following settings for the weight parameters:

- **Zero**: all weight parameters are initialized to be zeros (like biases).

[1] `http://yann.lecun.com/exdb/mnist/` - Use the standard train/valid/test splits such as the one provided *here*.
[2] `https://github.com/CW-Huang/IFT6135H19_assignment/blob/master/assignment1.ipynb`

- **Normal**: sample the initial weight values from a standard Normal distribution; $w_{i,j} \sim \mathcal{N}(w_{i,j}; 0, 1)$.
- **Glorot**: sample the initial weight values from a uniform distribution; $w^l_{i,j} \sim \mathcal{U}(w^l_{i,j}; -d^l, d^l)$ where $d^l = \sqrt{\frac{6}{h^{l-1}+h^l}}$.

1. Train the model for 10 epochs [3] using the initialization methods above and record the average loss measured on the training data at the end of each epoch (10 values for each setup).
2. Compare the three setups by plotting the losses against the training time (epoch) and comment on the result.

Hyperparameter Search [10] From now on, use the Glorot initialization method.

1. Find out a combination of hyper-parameters (model architecture, learning rate, nonlinearity, etc.) such that the average accuracy rate on the validation set ($r^{(valid)}$) is at least 97%.
2. Report the hyper-parameters you tried and the corresponding $r^{(valid)}$.

Validate Gradients using Finite Difference [15] The finite difference gradient approximation of a scalar function $x \in \mathbb{R} \mapsto f(x) \in \mathbb{R}$, of precision ϵ, is defined as $\frac{f(x+\epsilon)-f(x-\epsilon)}{2\epsilon}$. Consider the second layer weights of the MLP you built in the previous section, as a vector $\theta = (\theta_1, \ldots, \theta_m)$. We are interested in approximating the gradient of the loss function L, evaluated using **one** training sample, at the end of training, with respect to $\theta_{1:p}$, the first $p = \min(10, m)$ elements of θ, using finite differences.

1. Evaluate the finite difference gradients $\nabla^N \in \mathbb{R}^p$ using $\epsilon = \frac{1}{N}$ for different values of N

$$\nabla^N_i = \frac{L(\theta_1, \ldots, \theta_{i-1}, \theta_i + \epsilon, \theta_{i+1}, \ldots, \theta_p) - L(\theta_1, \ldots, \theta_{i-1}, \theta_i - \epsilon, \theta_{i+1}, \ldots, \theta_p)}{2\epsilon}$$

   Use at least 5 values of N from the set $\{k10^i : \ i \in \{0, \ldots, 5\}, \ k \in \{1, 5\}\}$.
2. Plot the maximum difference between the true gradient and the finite difference gradient ($\max_{1 \le i \le p} |\nabla^N_i - \frac{\partial L}{\partial \theta_i}|$) as a function of N. Comment on the result.

[3] One epoch is one pass through the whole training set.

Problem 2

Convolutional Networks: Many techniques correspond to incorporating certain prior knowledge of the structure of the data into the parameterization of the model. Convolution operation, for example, is designed for visual imagery.

Instructions: [25] For this part of the assignment we will train a convolutional network on MNIST for 10 epochs using your favorite deep learning frameworks such as Pytorch of Tensorflow. Plot the train and valid errors at the end of each epoch for the model.

1. Come up with a CNN architecture with more or less similar number of parameters as MLP trained in Problem 1 and describe it.
2. Compare the performances of CNN vs MLP. Comment.

You could take reference from the architecture mentioned here[4].

Problem 3

Dogs vs. Cats is an InclassKaggle challenge for image classification. Using what you have learned from the class and from previous problems, make an attempt at training the best CNN classifier for this task. Use the link here[5] to download the data and access the competition. Make sure to read the description and the rules thoroughly.

- The data is already preprocessed and ready to use, but you are free to add more layers of preprocessing if you wish. You may for example do your own cropping. Any extra preprocessing step should be mentioned in your report. You are not allowed to use any external data sources, but you can use usual data augmentation techniques. Please make sure to mention them. You are responsible for splitting the train set to an actual train and validation sets yourself.
- In addition to Kaggle submissions, we require that you submit your code through Gradescope, along with your report. We will run your code and make sure we can recreate your submission.
- You cannot use things that we have not covered in the class, directly from the deep learning library you are using, such as BatchNorm/WeightNorm/LayerNorm layers, regularization techniques (including dropout), and optimizers such as ADAM, **unless you implement them yourself**. Essentially, here is what you can use from your favorite deep learning library (e.g. Pytorch):

[4] `https://github.com/MaximumEntropy/welcome_tutorials/tree/pytorch/pytorch`
[5] `https://www.kaggle.com/c/ift6135h19`

  - Conv/Pool/Linear layers
  - Any activation function you want
  - Vanilla SGD (i.e. no momentum - no Adam)
  - Binary Cross Entropy loss function (in particular, no regularization like weight decay)
  - Data augmentation

  Anything else you want to use has to be implemented using numpy for example.

- This problem is meant to give you a chance to play with the methods seen in class and apply them to a real classification task. Show your efforts by recording what you have tried on the report (even things that did not lead to high scores).
- Your mark will be a combination of the accuracy score you obtained on the private leaderboard (more details on Kaggle) and the quality of your report. Your report should include the answers to the following questions:

Important: The Kaggle submission deadline is 1 day before the assignment deadline.

1. [20 score/5 description] Describe the architecture (number of layers, filter sizes, pooling, etc.), and report the number of parameters. You can take inspiration from some modern deep neural network architectures such as the VGG networks to improve the performance.
2. [15] Plot the training error and validation error curves, along with the training and validation losses. Comment on them. What techniques (you did not implement) could be useful to improve the validation performance. How does your validation performance compare to the test set performance (that you can only get in Kaggle).
3. [15] Compare different hyperparameter settings and report the final results of performance on your validation set. Aside from quantitative results, also include some visual analysis such as visualizing the feature maps or kernels, or showing examples where the images are (a) clearly misclassified and (b) where the classifier predicts around 50% on both classes. Explain your observation and/or suggest any improvements you think may help.