Due Date: February 12th (11pm), 2019

Problem 1

The first part of this assignment is the coding part. We provide the template in the course GitHub repository [1]. You need to fill up the blanks in the 'solution.py' script, WITHOUT modifying the template. In this problem, we will build a Multilayer Perceptron (MLP) and train it on the MNIST handwritten digit dataset.

Building the Model (code) [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. 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]. Implement the forward and backward propagation of the MLP in numpy without using any of the deep learning frameworks that provides automatic differentiation. Train the MLP using the probability loss (*cross entropy*) as the optimization criterion. We minimize this criterion to optimize the model parameters using *stochastic gradient descent*.

The following parts will be automatically graded:

1. Write the function `initialize_weights`.
2. Write all of the given activvation functions, i.e. `relu`, `tanh`, `sigmoid` and `softmax`, and the selector function `activation` that reads the string "activation_str" and select the corresponding activation function.
3. Write the `forward` pass function.
4. Write the `backward` pass function.
5. Write the `loss` functions (cross entropy for multi-class classification).
6. Write the `update` function that takes in the gradient to update the parameters according to the stochastic gradient descent update rule.
7. Finish the `train_loop` by performing the `forward` pass and the `backward` pass and updating the parameters (using the `update` function).

[1] `https://github.com/CW-Huang/IFT6135H20_assignment`

Solution
All the solutions were submitted to gradescope

Problem 2

The second part of this assignment will be graded based on your report (including comments and figures). You may create additional scripts aside from the template that we provide and/or reuse the template WITHOUT modifying or renaming it! One way to do it is to create another class object that inherits from the `NN` template.

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*. The following tasks should be written as a report and submitted via Gradescope (separately from the code).

Initialization (report) [15] 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).
- **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 [2] 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).
   Solution:
   The results shown in the figures correspond to the following architecture:

   - Number of hidden layers: 2. First layer with 784 units and the second one with 256
   - The input images have a size of $28 \times 28 = 784$. The output has a size of 10 (10 digits) so the total of parameters is: $615440 + 200960 + 2816 = 819216$
   - The learning rate is 0.1 and the non-linearity is the Relu function and the minibatch size is 64.

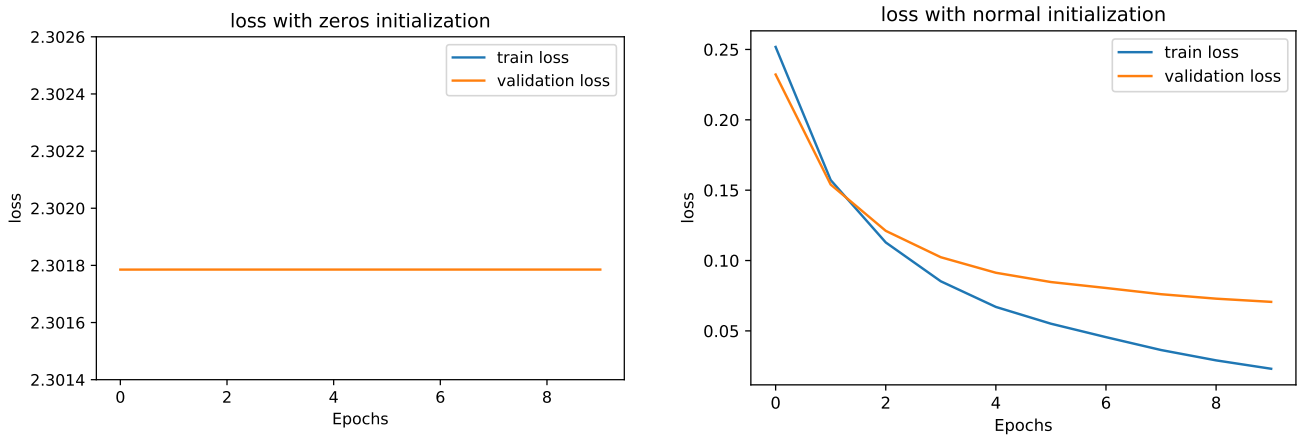

Figure 1: Loss on training and validation data. Left: Weight parameters were unitialized as zero. Right: Weight parameters were initialized using a normal distribution.

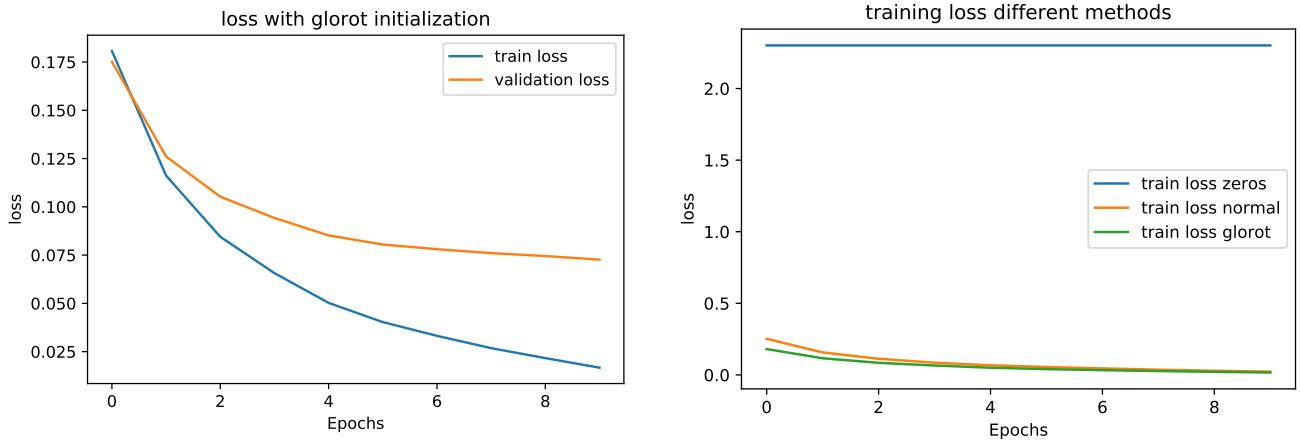

Figure 2: Left: Loss on training and validation data, weight parameters were unitialized using glorot distribution. Right: Loss on training data, comparation of the three methods (zeros, normal and glorot)

2. Compare the three setups by plotting the losses against the training time (epoch) and comment on the result.

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

Solution:
The training loss for the three methods is compared in the Fig.2 (right). We can see that both the normal distribution and glorot have a similar behaviour for epochs > 5. On the other hand, the initialization with zeros has a poor result on the training loss, having a similar value for different epochs. This happens because the derivative of the loss function is the same for each w, so all the weights values have the same value in subsequent iterations, causing symmetry in the hidden layers in all the iterations.

Hyperparameter Search (report) [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%.

   Solution:
   A validation accuracy of 98.02% was obtained with the following architecture:

   - Number of hidden layers: 2. First layer with 784 units and the second one with 256
   - minibatches of size 64.
   - The learning rate is 0.1 and the non-linearity is the Relu function.

2. Report the hyper-parameters you tried and the corresponding $r^{(valid)}$.
   Solution
   The parameters explored were:

   - Batch size (BS) = $\{64, 80\}$, learning rate (LR) = $\{0.1, 0.01\}$
   - nodes in hadden layer (HL) = $\{(784, 256), (1000, 150)\}$, epochs (E) = $\{10, 20\}$

   The results using different combinations are:

   - BS:64, LR:0.1, HL: (784,256), E:10 $\rightarrow$ 98.02%
   - BS:64, LR:0.1, HL: (1000, 150), E:10 $\rightarrow$ 97.58%
   - BS:80, LR:0.1, HL: (784,256), E:10 $\rightarrow$ 97.21%
   - BS:80, LR:0.01, HL: (784, 256), E:10 $\rightarrow$ 95.01%
   - BS:80, LR:0.1, HL: (784,256), E:20 $\rightarrow$ 97.92%

Validate Gradients using Finite Difference (report) [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

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

Use at least 5 values of N from the set $\{k10^i : \ i \in \{0, \ldots, 5\}, \ k \in \{1, 5\}\}$.

Solution
We accessed the values of the weight matrix of the second layer using the dictionary of the weights, and the values were interpreted as a vector. Subsequently, the first 10 values were perturbated separately with a variable ϵ value, for each disturbance the loss was calculated and the numerical derivative was calculated using the formula shown above.

2. Plot the maximum difference between the true gradient and the finite difference gradient ($\max_{1 \leq i \leq p} |\nabla_i^N - \frac{\partial L}{\partial \theta_i}|$) as a function of N. Comment on the result.

Solution
The results are shown in Fig.3. In Fig.3 the numerical derivative using finite difference is compared with the derivative obtained using the backpropagation method for the layer 2 (using the grads dictionary). From the results we see that as N is small (large epsilon) the difference between the derivatives is considerable, as N increases (ϵ decreases) the derivatives tend to have similar values as expected in the finite difference method .

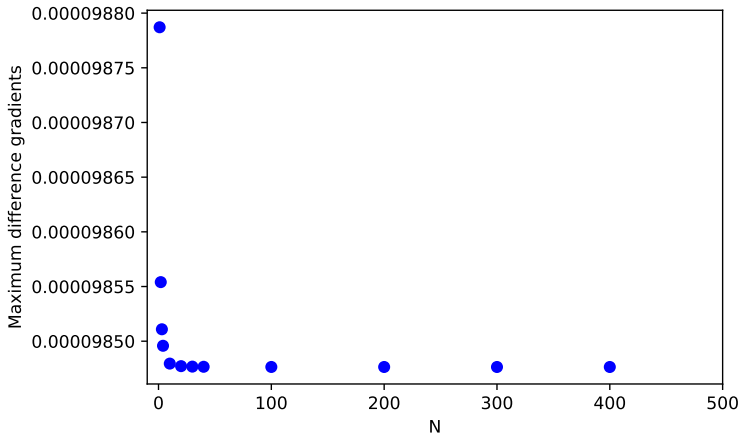

Figure 3: Difference between the finite difference gradient and the true gradient obtained using backpropagation as a function of N (ϵ)

Convolutional Neural Networks (report) [25] Many techniques correspond to incorporating certain prior knowledge of the structure of the data into the parameterization of the model. Convolution operation, for example, was originally designed for visual imagery. For this part of the assignment we will train a convolutional network on MNIST for 10 epochs using PyTorch. 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.

Solution
The architecture has the following features:

- Layer 1:
  - **Convolution**: Input: 28 (input image has a size of 28×28), padding: 2, kernel: 5, stride:1, channels: 16, so output: $28 \times 28 \times 16$
  - **Pooling**: Input: 28, padding: 0, stride: 2, kernel: 2, so output: $14 \times 14 \times 16$
- Layer 2:
  - **Convolution**: Input: 14 (output last layer), padding: 2, kernel: 5, stride:1, channels: 20, so output: $14 \times 14 \times 20$
  - **Pooling**: Input: 14, padding: 0, stride: 2, kernel: 2, so output: $7 \times 7 \times 20$
- Total parameters in convolution: $p = 2^2 * 20 * 16 = 1280$
- Input fully connected layer: $7 \times 7 \times 20 = 980$ nodes
- 2 fully hidden connected layers. The First layer was defined with 660 nodes, and the second layer with 254 nodes.
- Output fully connected layers: 10 (classes of the input data)
  So the weight matrix in the first fully connected layer has the dimension $[980, 660]$, the weight matrix of the second layer $[660, 254]$, and the third layer $[254, 10]$ so the total number parameters are: $647460 + 167894 + 2794 = 817904$ and the total of parameters will be $p = 817904 + 1280 = 819184$ very similar to the number of parameters of the MLP.

2. Compare the performances of CNN vs MLP. Plot the training loss and validation loss curve.

   Solution
   In Fig.4 we see the loss on training and validation as function of different epochs. From the results, we observe that up to 10 epochs the minimal loss minimal on training is 0.015 and for the validation is 0.05, therefore the loss using CNN is less than the loss of MLP (see Fig.2(left)). In the same way, the final accuracy in the validation using CNN is 99.13%, which is better than MLP (98.02%) using architectures with a similar amount of training parameters.

3. Explore *one* regularization technique you've seen in class (e.g. early stopping, weight decay, dropout, noise injection, etc.), and compare the performance with a vanila CNN model without regularization.

   Solution
   In Fig.5, we can observe the behavior of the validation loss as function of different values of weight decay. In general terms, there is an increase in the loss using L2 penalty as compared with the case without regularization The weight decay parameter adds a L2 penalty to the cost which can effectively lead to smaller model weights. This smaller model weights affect the final accuracy of the method, because the best accuracy (99.13%) was achieved without regularization.

You could take inspiration from the architecture mentioned here (hyperlink).

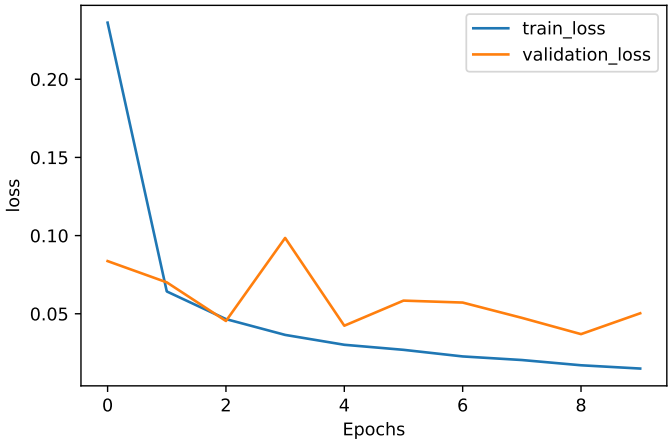

Figure 4: Loss on training and validation using CNN

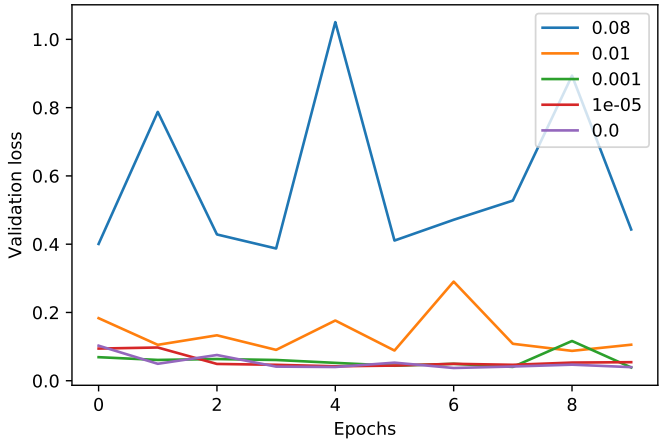

Figure 5: Loss on validation using different values of weight decay