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

Instructions

- *For all questions, show your work!*
- *Submit your code ("solution.py" only) and your report (pdf) separately and electronically via the course Gradescope page.*
- *Work off the template and DO NOT modify the name of the file "solution.py" since the code will be automatically graded.*
- ***TAs for this assignment are Jie Fu, Sai Rajeswar, and Akilesh B***

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).

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

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).

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

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%.
2. Report the hyper-parameters you tried and the corresponding $r^{(valid)}$.

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

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 $\theta = (\theta_1, \dots, \theta_m)$. 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

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

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

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.

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

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