CMPUT 328 Fall 2024

Assignment 1

Worth 8% of the total weight

Part 1: Logistic Regression for MNIST [worth 2% of the total weight]:

Implement Logistic Regression in PyTorch (You can make use of the multiple linear regression notebook from lecture):

a) Train and test on MNIST by defining your own data pipeline for training, validation and testing using PyTorch dataloader.
   - Use the last 12,000 samples of the train set as validation set.
   - Test the trained model on the validation set every few epochs to prevent overfitting.
   - **Do not use the test set for training**.
   - Use Stochastic Gradient Descent (SGD) as an optimizer.
   - Use the CrossEntropy loss.

b) Add a **regularization term** to improve your model (L1 or L2 regularization, whichever gives better accuracy)

Expected Performance: A correctly implemented, and somewhat well-tuned version of this algorithm will have an accuracy of **92-94%** on both test and validation sets of MNIST.

You need to complete the function *logistic_regression* in *A1_submission.py* for this part.

Part 2: Fully-connected Neural Network for CIFAR10[Worth 4% of the total weight]

Implement a Fully-connected Neural Network using the built-in functions of PyTorch. Train this network on the CIFAR10 dataset with CrossEntropy loss. The CIFAR10 dataset each image dimension is 32x32x3 as it is a color RGB image.
Please refer to the ***Network Architecture*** section for the specifications.

You need to complete the class *FNN* in *A1_submission.py* to implement the forward pass as well as the loss computation.

- The dataset split: Training = 40000 and Validation set = 10000
- Adjust the normalization for CIFAR10
- __init__(self, loss type, num classes) initializes your network layers
- forward(self, x) takes a batch of images as a tensor of size N × (32*32*3) and returns the class probabilities as a tensor of size N × 10 where N is the batch size
- get_loss(self, output, target) takes the output of the forward pass and ground truth labels of the corresponding images and returns a tensor containing the loss computed according to the loss type argument of __init__

Network Architecture

$$Y_p = Softmax(Relu(Tanh(XW_1 + b_1)W_2 + b_2))W_3 + b_3)$$

You can use built-in torch functions for defining the layers (nn.Linear) and activations. Dimensions of the vectors and matrices are as follows: X contains the input images having a shape ($N \times (32*32*3)$). N is the batch size. W_1 is ($32*32*3 \times 64$), b_1 is (1×64), W_2 is (64×32), b_2 is (1×32), W_3 is (32×10), b_3 is (1×10). Output probabilities Y_p has the shape ($N \times 10$). Note that for each of N indices in the first dimension, the softmax function is applied along the second dimension of its input matrix.

Part 3: Hyperparameter Search [Worth 2% of the total weight]

Find optimal hyperparameters using Adaptive Moment Estimation (Adam) on both part 1 (Logistic Regression) (1% of the part-3 weight) and part 2 (FNN) (1% of the part-3 weight).

- You *should* perform **grid search** or **random search** for finding the optimal hyper-parameters using accuracy on the validation set and select the best configuration.
- You can also use more advanced search strategies like evolutionary search, but you are **not** allowed to use any automatic parameter search methods like *scorch.*
- You **cannot** use the test set during this process.

You need to complete the function ***tune_hyper_parameter*** in *A1_submission.py* for this part.

Template Code

You are provided with template code in the form of three files: *A1_main.py, FNN_main.py* and *A1_submission.py*.

You need to complete the two functions (i.e. *logistic_regression* for part 1, *tune_hyper_parameter* for part 3) and *FNN* class in *A1_submission.py*. You can add any other functions or classes you want to *A1_submission.py* but do not make any changes to *FNN_main.py* and *A1_main.py.*

Running the code

Your own machine

Install python (version >= 3.6) if needed and install the required packages by running:

python3 -m pip install numpy torch torchvision tqdm paramparse

Run the code using:

python3 A1_main.py

python3 FNN_main.py

It is recommended to use an IDE like pycharm or vscode to make debugging easier.

Colab

Run this from a code cell in notebooks:

!python3 "<full path to A1_main.py >"

!python3 "<full path to FNN_main.py >"

You can optionally install the *paramparse* package to enable command line arguments:

!python3 -m pip install paramparse

You can then use command line arguments as:

part 1:

!python3 "<full path to A1_main.py >" mode=logistic

part 2:

!python3 "<full path to FNN_main.py >" mode=fnn

part 3:

!python3 "<full path to FNN_main.py >" mode=tune target_metric=accuracy

Submission

You need to submit only the completed *A1_submission.py*. Make sure to import any additional libraries you need so it can be used as a standalone Python module from *FNN_main.py* and *A1_main.py*.

To reiterate, please **do not** submit *FNN_main.py* and *A1_main.py* or any other files generated by running the code.

Marking

Part 1: Marks will depend on correctness of the implementation along with the following metrics:

- **Runtime**: The total runtime of your submission (including training and testing) **should not exceed 300 seconds** for either dataset on Colab GPU.
    - One trick to improve your run time is to grid search the hyperparameters first but only put in the best hyperparameters you found in your submission.
- **Accuracy**: Score scales linearly from **83 - 93%** accuracy on the test set

Part 2: Marks will depend on correctness of the implementation along with the following metrics:

- **Runtime**: The total runtime of your submission (including training and testing) **should not exceed 300 seconds** for either dataset on Colab GPU.
- **Accuracy**: Score scales linearly from **36-40%** accuracy on the test set

Part 3: Marks will depend on the correctness of your search implementation.

- **Runtime**: The runtime of your submission **should not exceed 1500 seconds** on Colab GPU.
- **Accuracy**: There are no specific accuracy requirements except there should be improvement in loss / accuracy compared to the baseline