ECE1513: Introduction to Machine Learning
Programming Assignment 3
Assigned: Oct 26, 2022; Due: Nov 18, 2022 @ 11:59 p.m.

Objectives

The purpose of this assignment is to investigate the classification performance of neural networks. You will implement, train and evaluate your neural network models using Pytorch on the notMNIST dataset. The functions are provided and **you will need to fill-in after the `TODO` comment instead of coding them from scratch**. In particular, in the first part, you will implement a fully connected neural networks (FNN). In the second part, you will implement a Convolutional Neural Networks (CNN), a to-go architecture for image recognition task. In the final part, you will implement the model training loop. You will also be asked to answer several questions related to your implementations. You are encouraged to look up Pytorch documentation for useful utility functions, at: `https://pytorch.org/docs/stable/index.html`. You can also refer to the demo covered in the lecture. The notMNIST dataset is stored as `notMNIST.npz` inside in the assignment folder. We highly recommend students to look at the tutorial files for this assignment.

To avoid any potential installation issue, you are encouraged to develop your solution using Google Colab notebooks. It is highly recommended that you train the neural network using a GPU.

Requirements

In your implementations, please use the function prototype provided (i.e. name of the function, inputs and outputs) in the detailed instructions presented in the remainder of this document. We will be testing your code using a test function that will evoke the provided function prototype. If our testing file is unable to recognize the function prototype you have implemented, you can lose significant portion of your marks. In the assignment folder, the following files are included in the `starter_code` folder:

- `NeuralNetPyTorch.py`
- `notMNIST.npz`: this is a dataset and you can upload this file to Google Colab. We provide functions to read the data in `NeuralNetPyTorch.py`.

These files contain the test function and an outline of the functions that you will be implementing. You also need to submit a separate `PA3_qa.pdf` file that answer questions related to your implementations.

Abbreviations

Following the definition in our code, we use the following abbreivations in this document:

- `BATCH_SIZE` refers to the batch size of images we are using for training. In this assignment, we set `BATCH_SIZE=32`.
- `F` is an abbreviated import name of `torch.nn.functional`.
- `nn` is an abbreviated import name of `torch.nn`.

Preliminaries

In this part, we explain several helper functions/classes in the `NeuralNetPyTorch.py` file. **Do not modify any of these functions/classes**.

- Function 1: `loadData(datafile="notMNIST.npz")`

- Inputs: `datafile`
  The default input is ""notMNIST.npz"", which is the dataset we are using in this assignment.
- Output: `trainData, validData, testData, trainTarget, validTarget, testTarget`
  The outputs are images and annotations in the form of Numpy matrices. `trainData` and `trainTarget` are the images and annotations for training. Similarly, `validData` and `validTarget` are the images and annotations for validation and `testData` and `testTarget` are the images and annotations for testing.
- Functionality:
  This function loads the notMNIST dataset and splits it into training, validation and testing set.

- Class 1: `class notMNIST(Dataset)`
  - Functionality:
    This class implements a PyTorch Dataset class for the notMNIST dataset.

  Read the description of the function below as you will use it in this assignment

- Function 2: `def experiment(model_type='CNN', learning_rate=0.0001, dropout_rate=0.5, weight_decay=0.01, num_epochs=50, verbose=False):`
  - Inputs: `model_type, learning_rate, dropout_rate, weight_decay, num_epochs, verbose`
    The input `model_type` is a `string` that specifies which type of model you will be using (either `"CNN"` for Convolutional Neural Network or `"FNN"` for fully-connected neural networks). The input `learning_rate, weight_decay, dropout_rate` are scalars of type `float` that specify the learning rate, the amount of L2 weight decay in your loss function and the dropout rate in your model respectively. The input `num_epochs` is an integer that specifies how many times your model will train through the whole training set. The input `verbose` is a Boolean that will let you print the training process if is `True` and nothing if is `False`.
  - Output: trained model and training history
    This function returns a trained model, which is a `torch.nn.Module` class and its training history. The training history is a dictionary that contains the accuracy on training, validation and test sets at each epoch, which is returned after calling the `train` function. Note: You will implement the `train` function in Part 3.
  - Functionality:
    This function will build your model (CNN or FNN), train it on the `notMNIST` dataset using the specified hyperparameters and returns a trained model with its training history.

Part 1: Fully Connected Neural Networks

In this part, you will be implementing a Fully Connected Neural Network using PyTorch. The model is defined in `class FNN(nn.Module)`. The neural network architecture that you will be implementing is as the following order:

1. An input layer for the images of size (`BATCH_SIZE x 1 x 28 x 28`). The second dimension is the image channel, which is 1 in this case since we are using gray-scale images. The third and forth dimension are the width and height of the input respectively. We will transform the (`BATCH_SIZE x 1 x 28 x 28`) matrix into a batch of 1D arrays of size (`BATCH_SIZE x 784`).
2. A fully connected layer followed by a ReLU activation. The size of the weight matrix is `784x10` where `784` is the size of the input array and `10` is the size of the 1st hidden layer.
3. A fully connected layer followed by a ReLU activation. The size of the weight matrix is `10x10` where `10` is the size of the 1st hidden layer and `10` is the size of the 2nd hidden layer.
4. A dropout layer with dropout probability `p`.

5. A fully connected layer **(without softmax activation)**. The size of the weight matrix is `10x10` where `10` is the size of the 2nd hidden layer and `10` is the size of the output layer.

Specifically, you will be implementing two functions in `class FNN(nn.Module)`, which are detailed in the following:

- Function 1: `def __init__(self, drop_out_p=0.0)`
  - Inputs: `self, drop_out_p`
    The input `drop_out_p` is a scalar that represents the dropout rate of the dropout layer in the neural network.
  - Output: This function does not return any output
    The purpose of this function is to setup the variables for this class. As shown in the tutorial, you need to define all the layers you will be using in this function.
  - Function implementation considerations:
    You will use the following PyTorch functions to setup the layers in your network: `nn.Linear()` and `nn.Dropout()`. The `nn.Linear()` function is a fully connected layer and is defined by the dimension of the input and output. For example, `nn.Linear(3, 4)` is a fully-connected layer for an input of dimension (`BATCH_SIZE, 3`) and an output of dimension (`BATCH_SIZE, 4`). To setup a dropout layer, use `nn.Dropout(p=drop_out_p)`.
- Function 2: `forward(self, x)`
  - Inputs: `self, x`
    The input `x` is the batch of images of size (`BATCH_SIZE, 1, 28, 28`). The input `self` represents the instance of the class.
  - Output: `out`
    This function computes the logits for each image in the batch. The output has a size of (`BATCH_SIZE, 10`), where each `(i,j)` position represent the class-logit score j of image `i`. The order of the layers (and activations) in this function must follow the described network architecture above.
  - Function implementation considerations:
    You will find the following PyTorch functions helpful: `torch.flatten(x, start_dim=1)`, `F.relu()`. The `torch.flatten(x, start_dim=1)` operation will flatten your input `x` of size (`BATCH_SIZE, 1, 28, 28`) to the size of (`BATCH_SIZE, 784`). Functions `F.relu()` applies the ReLU() activation to the input respectively.

The following is the mark breakdown for Part 1:

- Test file successfully runs implemented function: 15 marks
- Output is close to the expected output from the test file: 15 marks

Part 2: Convolutional Neural Networks

In this part, you will be implementing a Convolutional Neural Network using PyTorch. The model is defined in `class CNN(nn.Module)`. The neural network architecture that you will be implementing is as the following order:

1. **An input layer** for the images of size (`BATCH_SIZE x 1 x 28 x 28`). The second dimension is the image channel, which is 1 in this case since we are using gray-scale image. The third and forth dimension are the width and height of the input respectively.

2. **A convolutional layer** with the number of input channels and output channels to be 1 and 32 respectively. You will set the kernel size to 4 and leave the rest to default value. You will then apply **the `ReLU` activation function** to the output of this convolutional operation. After the activation, apply the **batch norm layer** to normalize the output vector. Finally, after applying the batch norm layer, you will apply a max pooling operation with a kernel size of `2x2`.

3. **A convolutional layer** with the number of input channels and output channels to be 32 and 64 respectively. You will set the kernel size to 4 and leave the rest to default value. You will then apply **the `ReLU` activation function** to the output of this convolutional operation. After the activation, apply the **batch norm layer** to normalize the output vector. Finally, after applying the batch norm layer, you will apply a max pooling operation with a kernel size of `2x2`.

4. **A flatten operation** to transform the previous hidden layer to a batch of 1D arrays. After this operation, add a **dropout layer** with dropout probability `p` before applying a fully-connected layer followed by **the `ReLU` activation function**. The size of the weight matrix is (`1024 x 784`), which means that the input dimension should be (`BATCH_SIZE x 1024`).

5. **A fully connected layer (without SoftMax activation)**. The size of the weight matrix is `784x10` where `784` is the size of the previous hidden layer and `10` is the size of the output layer.

Specifically, you will be implementing two functions in the `class CNN(nn.Module)`, which are detailed in the following:

- Function 1: `def __init__(self, drop_out_p=0.0)`
  - Inputs: `self, drop_out_p`
    The input `drop_out_p` is a scalar that represents the dropout rate of the dropout layer in the neural network. The input `self` represents the instance of the class.
  - Output: This function does not return any output
    The purpose of this function is to setup the variables for this class. As shown in the tutorial, you need to define all the layers you will be using in this function.
  - Function implementation considerations:
    Beside the previous layers in `FNN`, you will use the following PyTorch functions to setup the layers in your network: `nn.Conv2d`, `nn.BatchNorm2d` and `nn.MaxPool2D`. `nn.Conv2d` defines a 2D convolutional layer and you need to set the follow arguments: `in_channels`, `out_channels`, `kernel_size` to the number of input channels, the number of output channels and the kernel size respectively (leave the rest as in default). `nn.BatchNorm2d` is the batch-norm layer and you only need to set the `num_features` argument. `nn.MaxPool2D` is the pooling layer and you need to provide the kernel size.

- Function 2: `forward(self, x)`
  - Inputs: `self, x`
    The input `x` is the batch of images of size (`BATCH_SIZE, 1, 28, 28`). We use `self` input represents the instance of the class.
  - Output: `out`
    This function computes the logits for each image in the batch. The output has a size of (`BATCH_SIZE, 10`), where each `(i,j)` position represent the class-logit score `j` of image `i` belongs to class `j`. The order of the layers (and activations) in this function must follow the described network architecture above.
  - Function implementation considerations:
    You will find the following PyTorch functions helpful: `torch.flatten(x, start_dim=1)`, `F.relu()`.

The following is the mark breakdown for Part 2:

- Test file successfully runs implemented function: 25 marks
- Output is close to the expected output from the test file: 15 marks

Part 3: Model Training and Experiments

In this part, you will be implementing the training procedure for your neural networks. You will use the cross-entropy loss, Adam optimization method and L2 regularization to train your CNN and FNN. Implement (Complete) the following functions:

- Function 1: `def get_accuracy(model, dataloader):`
  - Inputs: `model, dataloader`
    `model` is an instance of the neural network class. `dataloader` is an instance of the notMNIST dataloader (see the function `experiment`).
  - Output:
    The output of this function is a scalar, which is the accuracy over all images in `dataloader`.
  - Function implementation considerations:
    This function will calculate the classification accuracy over the `dataloader` we specifed. It will be called whenever your model finishes one training epoch in the `train` function. Remember to set `model.eval()` to get the model into evaluation mode.
- Function 2: `def train(model, device, learning_rate, weight_decay, train_loader, val_loader, test_loader, num_epochs=50, verbose=False):`
  - Inputs: `model, device, learning_rate, weight_decay, train_loader, val_loader, test_loader, num_epochs=50, verbose`
    `model` is an instance of the neural network class. To train the neural networks with GPU, `device` should be `"cuda:0"` as shown in the `experiment` function. The `learning_rate`, `weight_decay` and float scalars that specify the learning rate and L2 weight decay of the model. `train_loader, val_loader, test_loader` are the notMNIST dataloader for training, validation and testing set respectively. `num_epochs` is the number of passes through the dataset we would like in our training (default is 50 in this assignment). Finally, set `verbose` to `True` will print out the training progress.
  - Output: `model, acc_hist`
    This function will return a trained `model` and the evolution of training, validation and testing accuracy.
  - Function implementation considerations:
    First, you need to specify the cross-entropy loss through the `criterion` variable that we will use as an optimization objective. After that, set the Adam optimizer through the `optimizer` variable (PyTorch Link) using the learning rate, weight decay value (for L2 regularization) and the model's weights (to be optimized). After that, complete the backpropagation process (forward, backward, update weights) in the training loop in the function. Remember to set the weight's gradients to zero.

Complete the following experiments, write and save your answer in a separate `PA3_qa.pdf` file. You will also need to write functions that prints/plots these experiment in your `NeuralNetPyTorch.py` submission. Remember to submit this file together with your code. Use the `experiment` function to complete these experiments. **For simplicity, we will discard the validation accuracy since fine-tuning the hyper-parameters for neural networks is a very time-consuming process.** To run the experiment, use the provided `experiment` function.

- Experiment 1: In this experiment, you will compare the performance between CNN and FNN in the image recognition task. Train your CNN and FNN model for a batch size of 32 (default), for 50 epochs (default) and the AdamW optimizer (Link to AdamW) for learning rate of 0.0001. Set the `dropout_rate=0.0` and `weight_decay=0.0`. Plot and compare the training/testing history of both models. The function name for this experiment will be `compare_arch()`.

- Experiment 2: In this experiment, you will study the effects of dropout rate on your CNN model. Train your CNN model for a batch size of 32 (default), for 50 epochs (default) and the AdamW optimizer (Link to AdamW) for learning rate of 0.0001. Fix your `weight_decay=0.0` and train 3 different models with `dropout_rate=0.5,0.8,0.95` respectively. Plot and compare the training/testing history of these three models. Explain the influence of dropout rate to the model's accuracy. The function name for this experiment will be `compare_dropout()`.
- Experiment 3: In this experiment, you will study the effects of weight decay on your CNN model. Train your CNN model for a batch size of 32 (default), for 50 epochs (default) and the AdamW optimizer (Link to AdamW) for learning rate of 0.0001. Fix your `drop_rate=0.0` and train 3 different models with `weight_decay=0.1,1.0,10.0` respectively. Plot and compare the training/testing history of these three models. Explain the influence of L2 regularization rate to the model's accuracy. The function name for this experiment will be `compare_l2()`.

The following is the mark breakdown for Part 3:

- Test file successfully runs implemented function: 6 marks
- Output is close to the expected output from the test file: 18 marks
- Questions are answered correctly: 6 marks