Programming Project 2: Neural Nets

Task Overview. In this project, you are going to work with different types of datasets and build neural net models based on these datasets. You will need to complete the following tasks:

1. Build and train a simple vanilla neural net model for a simple classification task based on a small dataset with 10,000 samples.
2. Build and train (i) a simple vanilla neural net model and (ii) a convolutional neural net model for a more complex classification task based on a large image dataset.
3. Tune hyper-parameters such as the learning rate, hidden size, number of training epochs, etc.
4. Report the training performance of your models.

Required Libraries. To complete your tasks, you will need to install the following libraries:

- `torch`: To install via Anaconda, run the command: `conda install pytorch`
- `torchvision`: To install via Anaconda, run the command: `conda install torchvision`

Task 1: Simple Classification (40 points)

Data Description. In this task, you will work with a simple dataset of 10,000 samples in which each data sample has only two features and belong to one of four classes. This dataset is divided into a training set and a test set. You will need to train your model using the training set and evaluate the performance of your model on the test set. The data distribution is shown in Figure 1.

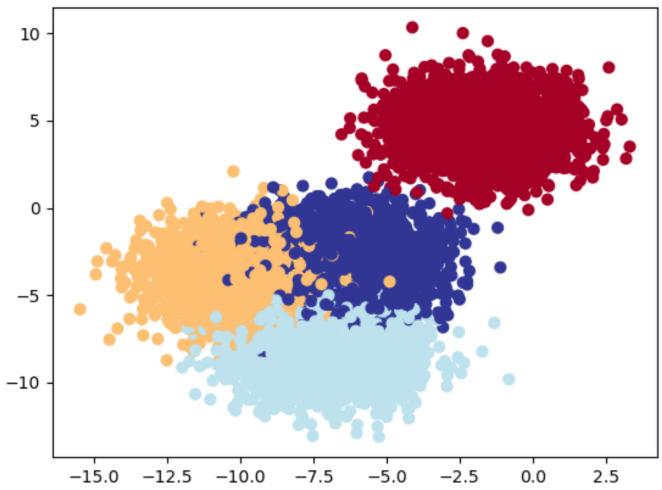

Figure 1: The Simple Dataset Distribution

Task Description. You are provided a file with starter code in it, `project2_simple_classification`. You have to implement:

1. (10 pts) `Class SimpleModel(nn.Module)`: In this class, you need to build a simple vanilla neural network which only consists of linear layers and non-linear activation functions. You need to implement two methods:

   (a) `__init__(self, input_size, output_size)`: The input of this method is the input and output size. You will determine which layers will be used in your network model here.

   (b) `forward(self, x)`: The input is data samples and the output is your model prediction.

2. (7.5 pts) `Method train_step(model, X_train, y_train, loss_fn, optimizer, device)`: In this method, you are going to write codes to run a complete training epoch of forward and backward pass to update your model. The input of this method includes: your model (model), the training dataset (X_train, y_train), the loss function you use (loss_fn), the optimizer (optimizer), and the device (either cpu or gpu) on which you run the training of your model (device). The output of this method includes the train loss and train accuracy of your model after the model update.

3. (5 points) `Method evaluation_step(model, X_test, y_test, loss_fn, device)`: In this method, you are going to write codes to evaluate your model (model) on the test set. The output is the loss and accuracy of your model on the test set.

4. (7.5 pts) `Method train_simple_model(X_train, y_train, X_test, y_test, random_seed)`: In this method, you will write codes to train your model using the methods `train_step` and `evaluation_step` that you implemented. You will need to initialize your model and determine which loss function and which optimizer to use, etc. In addition, you need to keep track of the loss and accuracy of your model on both the training and test sets after every training epoch. The output of this method is your trained model together with the training loss, training accuracy, test loss, and test accuracy of all training epochs.

5. (2.5 pts) `Method plot_accuracy_performance(train_accuracy, test_accuracy)`: In this method, you will need to plot the accuracy curves of your model on the training and test sets during the training process. You can reuse some your your codes in project 1.

6. (2.5 pts) `Method plot_loss_performance(train_loss, test_loss)`: In this method, you will need to plot the loss curves of your model on the training and test sets during the training process. You can also reuse some your your codes in project 1.

7. (5 pts) `Method plot_decision_boundary(model, X_train, y_train)`: In this method, you will need to plot the decision boundary outcome of your trained model with the scatter distribution of the training dataset. An example of the decision boundary is shown in Figure 2.

Figure 2: An example of decision boundary outcome

Important note. The accuracy of your trained model on the test set must be at least 90%.

Task 2: Image Classification (50 points)

Data Description. In this task, you will work with an image dataset. In this dataset, there are 48,000 images in the training set, 12,000 images in the validation set, and 10,000 images in the test set. Each image is of size $1\times28\times28$. The validation set is typically used to tune hyper-parameters such as the learning rate, the neural net architecture, and the number of training epochs, etc. Examples of images are shown in Figure 3.

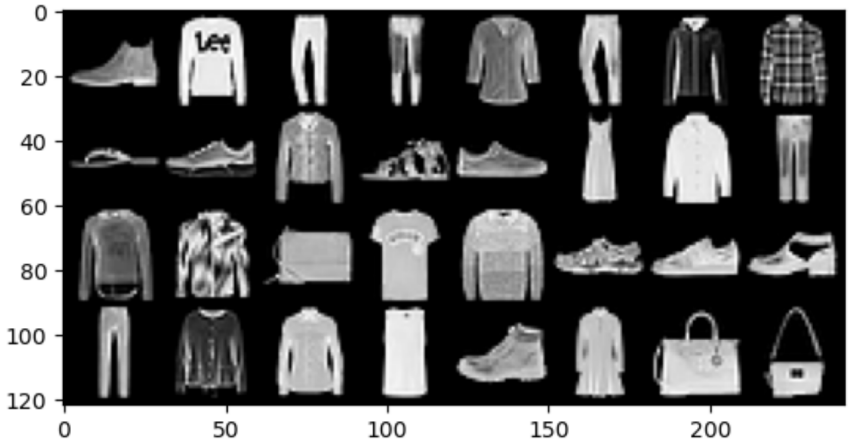

Figure 3: Examples of images in the dataset

Each image is assigned to one of the following 10 labels: {T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle Boot}. You will need to build and train a neural net model using the training set. You can also use the validation set to tune your hyper-parameters. Finally the test set is used to evaluate the performance of your trained model.

Task Description. You are provided a file with starter code in it, `project2_vision`. Your task is to build and train a convolutional neural network model. In addition, you will need to incorporate regularization using the L2 norm regularization term to train the model. That is, the loss function used to train your model should be defined as follows:

$$\text{loss} = \text{prediction\_loss} + \text{reg\_param} \times R(W)$$

where `prediction_loss` is the prediction loss of your model, $R(W)$ is the L2 norm of your model weights, and `reg_param` is the regularization parameter.

In the `project2_vision` file, all datasets can be extracted from `train_loader`, `valid_loader`, and `test_loader`. You can find more information about DataLoader at: `https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader`.

In particular, you have to implement the following components:

1. (14 pts) `Class ConvModel(nn.Module)`: In this class, you need to build a convolutional neural network. You need to complete two methods: `__init__(self, input_size, output_size)` and `forward(self, x)`. The input and output of these methods are similar to Task 1.

2. (14 pts) `Method train_step(model, train_loader, loss_fn, optimizer, reg_param, device)`: In this method, you are going to write codes to run a complete training epoch of forward and backward pass to update your model. The input of this method includes: your model (model), the training dataset (train_loader), the loss function you use (loss_fn), the optimizer (optimizer), the regularization parameter (reg_param), and the device (either cpu or gpu) on which you run the training of your model (device).

The output of this method includes the train loss and train accuracy of your model after the model update.

3. (6 pts) `Method evaluation_step(model, data_loader, loss_fn, reg_param, device)`: In this method, you are going to write codes to evaluate your model (model) on either the validation or test set, which is represented as `data_loader`. The output is the loss and accuracy of your model on either the test set or the validation set.

4. (14 pts) `Method train_conv_model(train_loader, valid_loader, test_loader, random_seed)`: In this method, you will write codes to train your model using the methods `train_step` and `evaluation_step` that you implemented. You will need to initialize your model and determine which loss function and which optimizer to use. Keep in mind that you should tune the hyper-parameters including your model architecture, the learning rate, and the regularization parameter, etc based on the performance of your model on the validation set during the training process. In addition, you need to keep track of the loss and accuracy of your model on the training, validation, and test sets after every training epoch. The output of this method is your trained model together with the training loss, training accuracy, validation loss, validation accuracy, test loss, and test accuracy of all training epochs.

5. (2 pts) Methods `plot_accuracy_performance` and `plot_loss_performance`: These methods are similar to the ones in Task 1, except you will need to plot the curves for all three datasets: train, validation, and test sets. You can re-use the codes you wrote in Task 1.

Important Note. The accuracy of your trained model on the test set must be at least 90%.
Extra Credit (5 points). In Task 2, you will get an extra of 5 points if the accuracy of your trained model on the test set is at least 95%.

3 Task 3: Write Report (10 points)

Finally, you will write a report on the results of your model performance. Your report should include the following results:

1. The loss and accuracy curves of your model on the training and test sets during the training process in Task 1. Write 2–3 sentences summarizing your observations regarding these results, with a comment on the under-fitting/over-fitting/convergence performance of your model.

2. The loss and accuracy curves of your model on the training, validation, and test sets during the training process in Task 2. Write 2–3 sentences summarizing your observations regarding these results, with a comment on the under-fitting/over-fitting/convergence performance of your model.

Save your report in a file named `proj2_report.pdf`.

4 Submission

You will need to submit the following files: (i) `project2_simple_classification.ipynb`; (ii) `project2_vision.ipynb`; and (iii) `proj2_report.pdf` on Canvas.