Middle East Technical University Department of Computer Engineering

CEng 499

Introduction to Machine Learning

Spring 2019-2020

Homework 1 - Artificial Neural Networks

version 1.0

Due date: 10 March 2020, 23:59

1 Introduction

In this assignment, you will have the chance to get hands-on experience with artificial neural networks. The assignment consists of three tasks. Firstly, you will implement a multilayer perceptron (MLP) from scratch by using NumPy library. Secondly, you will optimize the hyperparameters of a neural network. Lastly, you will build a convolutional neural network (CNN) with PyTorch framework.

2 Multilayer Perceptron (30 pts)

In this section, you will implement a two layer (one hidden layer) MLP by using NumPy library, a general-purpose array-processing package. No other library or framework is allowed. After building and training the model, you will test it with the dataset we provided. The program will read three command line arguments: the path of the train set, the path of the test set, and the number of epochs respectively. An epoch is one complete pass through the dataset. As the output, the program will print the test accuracy to the screen and terminate. Both sets can be downloaded from ODTUClass.

- Write your code in a file called **task1.py**. When grading, we will run it with different datasets and number of epochs.
- The following code can be used for reading the train/test set.

```
import pickle, gzip
with gzip.open(dataset_path) as f:
    data, labels = pickle.load(f, encoding='latin1')
```

- While the number of nodes at the input layer (n_i) is the number of features, at the output layer (n_o) there is only one node (binary classification task).

The number of nodes at hidden layer (n_h) will be $n_h = \lfloor \frac{n_i+n_o}{2} + 1 \rfloor$ for this task.

- The activation function for the output layer is sigmoid function.

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

For the hidden layer, the activation function is the following piecewise function.

$$f(x) = \begin{cases} 0, & x \leq -1 \\ \frac{x}{2} + 0.5, & -1 < x < 1 \\ 1, & x \geq 1 \end{cases} \tag{2}$$

- L_2 loss function will be used to train the classifier.

$$L_2 = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2 \tag{3}$$

where N is the number of training examples, y is the true label, and $\hat{y}$ is the output of the model, the predicted label.

- Initialize the weights with small random numbers and the biases with zero. Use the following code and replace ... with dimension values of the layers.

```
    weight = np.random.randn(...) * 0.01
    bias = np.zeros((...))
```

- To get reproducible and consistent results, you need to fix the randomness in your code. Before writing anything else, prepend the following code so that NumPy can always produce the same random number sequence.

```
    import numpy as np
    np.random.seed(1234)
```

3 Hyperparameter Tuning (50 pts)

For the model in the previous section, we specified some parameters such as the number of layers, the number of nodes at a layer, and the type of an activation function. These parameters are called **hyperparameters**. Unlike parameters, they are not optimized via training process. Hyperparameters define the structure of a model, and how the model is trained. In this section, you are going to implement models with different hyperparameter configurations and report the results.

- Write your code in a file called **task2.py**. When grading, we run your program with the hyperparameter configurations you tried. You will write the configurations and the test accuracies you achieved in a report named **report.pdf**.

- You will employ PyTorch deep learning framework. No other library or framework is allowed for building the models.
- You are expected to use fully-connected layers in your network.
- To get reproducible and consistent results, you need to fix the randomness in your code. Before writing anything else, prepend the following code so that PyTorch can always produce the same random number sequence.

```
import torch
torch.manual_seed(1234)
```

- For this section, CIFAR-10 dataset is selected. The dataset consists of 32×32 pixel RGB images, each of which belongs to one of 10 categories. Use the following code to download the dataset. You can use any ratio for the validation set.

```
import torch
import torchvision.transforms as transforms
from torchvision.datasets import CIFAR10

ratio = 0.1
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

train_set = CIFAR10(root="task2_data", train=True,
                    transform=transform, download=True)
test_set = CIFAR10(root="task2_data", train=False,
                   transform=transform, download=True)
train_set, val_set = torch.utils.data.random_split(train_set,
                            [int((1 - ratio) * len(train_set)),
                             int(ratio * len(train_set))])
```

- The best configuration should achieve at least 50% accuracy in the test set.
- You can choose any optimizer. However, the loss function that is optimized must be **cross entropy**. Be aware that PyTorch's cross entropy implementation does not need explicit **softmax** activation function.
- You are going to tune the following hyperparameters. You can also experiment on additional hyperparameters.
  1. Try different number of layers. You should at least try 1, 2, and 3 layer networks. The number of neurons at each layer is not important as long as you meet the minimum test set accuracy.

2. Try different activation functions for each layer. You should try different combinations of at least 3 types of activation functions. Possible choices can be **Leaky ReLU**, **GELU**, and **SELU**.
3. Try different learning rates. You should try at least 5 different values. Possible choices can be 0.1, 0.03, 0.01, 0.003, and 0.001.

For each hyperparameter configuration, state the test result accuracy in your report. At minimum, you experiment $5 + 15 + 45$ different configurations. It is up to you how you present the results (for example, in the form of a table).

- Discuss how you decided the best hyperparameters. What method did you use?
- For each k-layer network, select the best performing hyperparameter configuration and draw its training and validation losses on the same graph and comment on it. What countermeasure did you take against overfitting? How may one understand when a network starts to overfit during training? What method did you use to decide where to end the training procedure?
- Discuss whether accuracy is a suitable metric to measure the performance of a network for this specific dataset. Explain your reasons. (You don't have to use another metric in your experiments if you can achieve the minimum accuracy required for this homework. Just discuss how using another metric might have changed hyperparameter optimization results.)

4 Convolutional Neural Networks (20 pts)

In this section, you are going to write a convolutional neural network (CNN) model that works with arbitrary input image sizes by using PyTorch. The dataset that you are going to work on is MNIST, which consists of 28×28 grayscale images, each of which belongs to one of 10 categories. However, we apply random croppings and paddings to the images, thereby making the size of the input image to the model arbitrary. In **task3.py**, we have created a skeleton for the model. You are free to design the architecture of your CNN as you like, however, it should have at least two convolutional layers.

5 Specifications

- Falsifying results, changing the composition of training and test data are strictly forbidden, and you will receive 0 if this is the case. Your programs will be examined to see if you have actually reached the results and if it is working correctly.

- Using any piece of code that is not your own is strictly forbidden and constitutes as cheating. This includes friends, previous homeworks, or the internet. The violators will be punished according to the department regulations.
- Follow the course page on ODTUClass for any updates and clarifications. Please ask your questions on the discussion section of ODTUClass instead of e-mailing.
- You have total of 3 late days for **all** your homeworks. For each day you have submitted late, you will lose 10 points. The homeworks you submit late after your total late days have exceeded 3 will not be graded.

6 Submission

Submission will be done via ODTUClass. You will submit a zip file called **hw1.zip** that contains **task1.py, task2.py, task3.py** and **report.pdf**.