

## ▼ CS480/680, Spring 2023, Assignment 2

Designer: Yimu Wang; Instructor: Hongyang Zhang

Released: June 5; Due: **Extended: June 29, noon**

[Link of the Assignment](#)

Tips:

- Please save a copy of this notebook to avoid losing your changes.
- Debug your code and ensure that it can run.
- Save the output of each cell. Failure to do so may result in your coding questions not being graded.
- **To accelerate the training time, you can choose 'Runtime' -> 'Change runtime type' -> 'Hardware accelerator' and set 'Hardware accelerator' to 'GPU'. With T4, all the experiments can be done in 5 minutes. In total, this notebook can be run in 20 minutes.** (You can either use or not use GPU to accelerate. It is your choice and it does not affect your grades.)
- Your grade is independent of the test accuracy (unless it is 10%, as 10% is the accuracy of random guess).

Tips for submission:

- Do not change the order of the problems.
- Select 'Runtime' -> 'Run all' to run all cells and generate a final "gradable" version of your notebook and save your ipynb file.
- **We recommend using Chrome to generate the PDF.**
- Also use 'File' -> 'Print' and then print your report from your browser into a PDF file.
- **Submit both the .pdf and .ipynb files.**
- **We do not accept any hand-written report.**

**Question 0. [Important] Please name your submission as {Last-name}\_{First-name}\_assignment2.ipynb and {Last-name}\_{First-name}\_assignment2.pdf. If you do not follow this rule, your grade of assignment 2 will be 0.**

## ▼ Question 1. Basics of MLP, CNN, and Transformers (40 points)

- 1.1 Given an MLP with an input layer of size 100, two hidden layers of size 50 and 25, and an output layer of size 10, calculate the total number of parameters in the network. (10 points)

This MLP network is a standard MLP with bias terms. The activation function is ReLU. But you do not need to consider the parameters of the activation function.

Solution: [Your answer here, You should give the full calculation process.]

- 1.2 Given the loss functions of [mean squared error \(MSE\)](#) and [CE \(cross-entropy\) loss \(between logits and target\)](#), and the predicted logit of a data example (before softmax) is [0.5, 0.3, 0.8], while the target of a data example is [0.3, 0.6, 0.1], calculate the value of the loss (MSE and CE). (10 points)

The loss functions of MSE and CE are as follows,

$$\ell_{MSE}(y, \hat{y}) = \sum_{i \in C} (y_i - \hat{y}_i)^2,$$

$$\ell_{CE}(y, \hat{y}) = - \sum_{i=1}^C y_i \log \left( \frac{\exp(\hat{y}_i)}{\sum_{j \in C} \exp(\hat{y}_j)} \right),$$

where  $\hat{y}_i$  is the  $i$ -th element of predict logit (before softmax),  $y$  is the  $i$ -th element of target, and  $C$  is the number of classes.

Solution: [Your answer here, You should give the full calculation process.]

1.3 Given a convolutional layer in a CNN with an input feature map of size 32x32, a filter size of 3x3, a stride of 1, and no padding, calculate the size of the output feature map. (10 points)

You can refer to [PyTorch Convolutional Layer](#) for the definition of the convolutional layer.

Solution: [Your answer here, You should give the full calculation process.]

1.4 Given a Transformer encoder with 1 layer, the input and output dimensions of attention of 256, and 8 attention heads, calculate the total number of parameters in the **self-attention mechanism**. (10 points)

You can refer to [Attention is all you need \(Transformer paper\)](#) for reference.

Solution: [Your answer here, You should give the full calculation process.]

## Question 2. Implementation of Multi-Layer Perceptron and understanding Gradients (60 points)

In this question, you will learn how to implment a Multi-Layer Perceptron (MLP) PyTorch, test the performance on CIFAR10, and learn what is gradient. Please refer to [CIFAR10](#) and [PyTorch](#) for details.

Assuming the MLP follows the following construction

$$\hat{y} = \text{softmax}(W_2 \text{sigmoid}(W_1 x + c_1) + c_2),$$

where  $\hat{y}$  is the prediction (probability) of the input  $x$  by the MLP and  $W_1$ ,  $W_2$ ,  $c_1$ , and  $c_2$  are four learnable parameters of the MLP. softmax and sigmoid are the [softmax](#) and [sigmoid](#) function.

Please note that the label is one-hot vectors, i.e.,  $y_i$  are either 0 or 1. And the sum of prediction is equal to 1, i.e.,  $\sum_{i \in [C]} \hat{y}_i = 1$ . **However, usually in PyTorch, the labels are integers. You may need to transfer integers into one-hot vectors.**

Tips: The process of using SGD to train a model is as follows:

1. Initialize the model parameters.
2. Calculate the gradients of the model parameters.
3. Update the model parameters using the gradients.
4. Test the model.
5. Repeat 2 - 4 until the model converges or for a given epochs.

**You do not need to implement the training procedure. Please follow the instruction of each question. This is just set to help you understanding the training procedure.**

You can also refer to [SGD](#) and [PyTorch Tutorial](#) for inspiration.

**Please note that you are allowed to use any PyTorch api and any public code to complete the following coding questions. This assignment is help you to learn PyTorch.**

## 2.0 Get CIFAR10 data (0 points)

We will show you how to get the data using PyTorch. **You are not allowed to edit the following code.** Please see [Dataset and DataLoaders](#) for details.

```
### YOU ARE NOT ALLOWED TO EDIT THE FOLLOWING CODE. ###
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

def get_CIFAR10():
    # Set the random seed for reproducibility
```

```

torch.manual_seed(480)

# Load the CIFAR10 dataset and apply transformations
train_dataset = datasets.CIFAR10(root='./data', train=True,
                                  transform=transforms.ToTensor(),
                                  download=True)
test_dataset = datasets.CIFAR10(root='./data', train=False,
                                  transform=transforms.ToTensor())

# Define the data loaders
batch_size = 100
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
return train_dataset, test_dataset, train_loader, test_loader

```

## ▼ 2.1 Implement the MLP with PyTorch (15 points)

The shape of  $W_1$  and  $W_2$  should be **256 × 3072** and **10 × 256**.

You can refer to [Define a NN](#) for inspiration.

```

import torch.nn as nn
import torchvision.models as models

##### COMPLETE THE FOLLOWING CODE, YOU ARE ALLOWED TO CHANGE THE SKLETON CODE#####

# Define the MLP model in PyTorch
class MLPinPyTorch(nn.Module):
    def __init__(self, input_dim=3072, hidden_dim=256, output_dim=10):
        super(MLPinPyTorch, self).__init__()
        # TODO: Your code here: init with parameters
        # you are allowed to use any initialization method
        pass

    def forward(self, x):
        # TODO: Your code here: forward
        pass

```

## ▼ 2.2 Calculate the accuracy given true labels and predicted labels (5 points)

You should complete the following code, including the calculation of the accuracy.

```

##### COMPLETE THE FOLLOWING CODE, YOU ARE ALLOWED TO CHANGE THE SKLETON CODE#####
def accuracy(y, predicted):
    # TODO: Your code here: calculate accuracy
    pass

```

## ▼ 2.3 Test your implementation on CIFAR10 and reports the accuracy on training and testing datasets (20 points)

```

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
##### COMPLETE THE FOLLOWING CODE, YOU ARE ALLOWED TO CHANGE THE SKLETON CODE#####
def test(model, train_dataloader, test_dataloader):
    # TODO: Your code here: calculate accuracy
    predict_train = []
    predict_test = []
    train_labels = []
    test_labels = []

    for inputs, labels in enumerate(train_dataloader):
        inputs = inputs.view(-1, 3072).to(device)
        train_labels.append(labels)

        # Forward pass

    for inputs, labels in enumerate(test_dataloader):
        inputs = inputs.view(-1, 3072).to(device)

```

```

test_labels.append(labels)

# Forward pass

train_acc = accuracy(train_labels, predict_train)
test_acc = accuracy(test_labels, predict_test)
return train_acc, test_acc

##### the following is served for you to check the functions #####
##### you can change but ensure the following code can output the accuracies #####
model = MLPinPyTorch()
train_dataset, test_dataset, train_loader, test_loader = get_CIFAR10()
train_acc, test_acc = test(model, train_loader, test_loader)
print("train_acc: ", train_acc)
print("test_acc: ", test_acc)

```

▼ 2.4 Calculate the gradients  $\frac{\partial \ell}{\partial W_1}$ ,  $\frac{\partial \ell}{\partial c_1}$ ,  $\frac{\partial \ell}{\partial W_2}$ , and  $\frac{\partial \ell}{\partial c_2}$  using algebra given a data example  $(x, y)$  (20 points)

The loss function we use for training the MLP is the [log-loss](#), which is defined as follows:

$$\ell(y, \hat{y}) = - \sum_{i=1}^C y_i \log(\hat{y}_i),$$

where  $C$  is the number of classes,  $y_i$  and  $\hat{y}_i$  are the  $i$ -th index of  $y$  and  $\hat{y}$ .

Considering the MLP model in question 2, please use chainrule to calculate the gradients. You can directly write LaTeX/Markdown in the following cell.

Solution: [Your answer here]

[Colab paid products](#) - [Cancel contracts here](#)

✓ 5s completed at 2:08 PM

