CS480/680, Spring 2023, Assignment 2, Solutions

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 accelearte. It is your choice and it does not affect your grades.)

Tips for sumbission:

- 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:

First hidden layer: The first hidden layer has 50 neurons. Each neuron receives inputs from the 100 neurons in the input layer, and there is a bias term associated with each neuron. Therefore, there are a total of (100 * 50) + 50 = 5,050 parameters in the first hidden layer.

Second hidden layer: The second hidden layer has 25 neurons. Each neuron receives inputs from the 50 neurons in the first hidden layer, and there is a bias term associated with each neuron. Therefore, there are a total of (50 * 25) + 25 = 1,275 parameters in the second hidden layer.

Output layer: The output layer has 10 neurons. Each neuron receives inputs from the 25 neurons in the second hidden layer, and there is a bias term associated with each neuron. Therefore, there are a total of (25 * 10) + 10 = 260 parameters in the output layer.

Total number of parameters in the network: Adding up the parameters from all the layers, we get:

5,050 + 1,275 + 260 = 6,585

Please note that the following alternative answer is also acceptable, and it will not result in a deduction of points for the final results:

Total number of parameters: 100 * 50 + 50 * 25 + 25 * 10 = 6,500

Explanation: This calculation does not consider any bias terms in the network.

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,

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

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

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:

Mean squared error (MSE) loss:

Squared differences: (0.5 - 0.6)^2 = 0.01 (0.3 - 0.4)^2 = 0.01 (0.7 - 0.8)^2 = 0.01

Mean squared error (MSE) loss: MSE = 0.01 + 0.01 + 0.01 = 0.03

Therefore, the MSE loss is 0.03. (0.01 is also accetable if divided by 3.)

The cross-entropy (CE) loss:

CE = -[(0.6 * log(0.5)) + (0.4 * log(0.3)) + (0.8 * log(0.7))]

Please note that the base of the logarithm used is not specified in the question. All bases are acceptable, but it is recommended to use either base 2 or base e (natural logarithm). Therefore, the cross-entropy (CE) loss is approximately 1.1828 (e for the base) or 1.7064 (2 for the base).

No grade deduction will be applied for the number of decimals.

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:

To calculate the size of the output feature map in a convolutional layer, we can use the following formula:

output_size = (input_size - filter_size + 2 * padding_size) / stride + 1

Given: Input feature map size: 32x32 Filter size: 3x3 Stride: 1 No padding

Applying the formula:

output_size = (32 - 3) / 1 + 1 = 30 / 1 + 1 = 30 + 1 = 31

Therefore, the size of the output feature map in the convolutional layer is 31x31.

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:

For self-attention, the key, query, and value vectors are linearly transformed using separate learned weight matrices. In each attention head, there are three weight matrices involved: one for the key, one for the query, and one for the value. Each weight matrix has a shape of (embedding dimension) x (embedding dimension/number of attention heads).

Therefore, the total number of weight parameters involved in the self-attention mechanism of a single encoder layer can be calculated as follows:

Number of weight parameters = Number of attention heads * (3 * embedding dimension * embedding dimension / number of attention heads)

Number of weight parameters = 8 * (3 * 256 * 256 / 8) = 8 * (3 * 256 * 32) = 196,608

Therefore, there are 196,608 weight parameters involved in the self-attention mechanism of a single encoder layer in this Transformer model.

No grade deduction if the final answer includes the parameter of FC after the attention.

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 PARAMETERS OF EACH FUNCTION#####

# 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__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.sigmoid = nn.Sigmoid()
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.stmx = nn.Softmax(dim=1)

    def forward(self, x):
        x = x.view(x.size(0), -1)
        hidden = self.fc1(x)
        hidden_activation = self.sigmoid(hidden)
        output = self.fc2(hidden_activation)
        return self.stmx(output)
```

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 PARAMETERS OF EACH FUNCTION#####
import numpy as np
def accuracy(y, predicted):
    predicted = np.array(predicted)
    y = np.array(y)
    return (predicted == y).mean()
```

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 PARAMETERS OF EACH FUNCTION#####
def test(model, train_dataloader, test_dataloader):
    predict_train = []
    predict_test = []
    train_labels = []
    test_labels = []

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

        # Forward pass
        pred_label = torch.argmax(model(inputs), dim=1)
        predict_train.extend(pred_label.cpu().detach().numpy().tolist())

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

        # Forward pass
        pred_label = torch.argmax(model(inputs), dim=1)
        predict_test.extend(pred_label.cpu().detach().numpy().tolist())

    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 you have to ensure the following code can outp
model = MLPinPyTorch().to(device)
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)
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:01<00:00, 99964010.81it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
```

```
train_acc:  0.09694
test_acc:  0.0954
```

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:

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

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:

1. Gradient with respect to W_2:

$$\frac{\partial \ell}{\partial W_2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial W_2} \ .$$

From the given loss function $\ell(\hat{y}, y) = -\sum_{i=1}^{C} y_i \log(\hat{y_i})$, we have:

$$\frac{\partial \ell}{\partial \hat{y}} = -\frac{y}{\hat{y}},$$

where $\frac{y}{\hat{y}}$ is element-wise.

The derivative of the softmax function with respect to W_2 is given by:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial W_2} =& \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2) \\ & (1 - \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2)) \ \text{sigmoid}(W_1 x + c_1) \ .\end{aligned}$$

Therefore, combining the above equations, we get:

$$\begin{aligned}\frac{\partial \ell}{\partial W_2} =& -\frac{y}{\hat{y}} \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2) \\ & (1 - \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2)) \ \text{sigmoid}(W_1 x + c_1) \ .\end{aligned}$$

2. Gradient with respect to c_2:

$$\frac{\partial \ell}{\partial c_2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial c_2} \ .$$

The derivative of the softmax function with respect to c_2 is simply:

$$\begin{aligned}\frac{\partial \hat{y}}{\partial c_2} =& \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2) \\ & (1 - \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2)) \ .\end{aligned}$$

Therefore, we have:

$$\begin{aligned}\frac{\partial \ell}{\partial c_2} =& -\frac{y}{\hat{y}} \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2) \\ & (1 - \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2)) \ .\end{aligned}$$

3. Gradient with respect to W_1:

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \ \text{sigmoid}} \frac{\partial \ \text{sigmoid}}{\partial (W_1 x + c_1)} \frac{\partial (W_1 x + c_1)}{\partial W_1} \ .$$

$$\begin{aligned}\frac{\partial \hat{y}}{\partial \ \text{sigmoid}} =& \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2) \\ & (1 - \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2)) \ W_2 \ .\end{aligned}$$

The derivative of the sigmoid function with respect to its input is:

$$\frac{\partial \ \text{sigmoid}}{\partial (W_1 x + c_1)} = \text{sigmoid}(W_1 x + c_1) \cdot (1 - \text{sigmoid}(W_1 x + c_1)) \ .$$

Now, we have

$$\begin{aligned}&\frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \ \text{sigmoid}} \frac{\partial \ \text{sigmoid}}{\partial (W_1 x + c_1)} \\ =& -\frac{y}{\hat{y}} \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2) \\ & (1 - \text{softmax}(W_2 \ \text{sigmoid}(W_1 x + c_1) + c_2)) \\ & \text{sigmoid}(W_1 x + c_1)(1 - \text{sigmoid}(W_1 x + c_1)) \ .\end{aligned}$$

The derivative of the linear transformation $(W_1 x + c_1)$ with respect to W_1 is simply x:

$$\frac{\partial (W_1 x + c_1)}{\partial W_1} = x \, .$$

Therefore, combining the above equations, we get:

$$\begin{aligned}
&\frac{\partial \ell}{\partial W_1} \\
&= -\frac{y}{\hat{y}} \text{softmax}(W_2 \, \text{sigmoid}(W_1 x + c_1) + c_2) \\
&\quad (1 - \text{softmax}(W_2 \, \text{sigmoid}(W_1 x + c_1) + c_2)) \, \text{sigmoid}(W_1 x + c_1) \\
&\quad (1 - \text{sigmoid}(W_1 x + c_1)) x \, .
\end{aligned}$$

4. Gradient with respect to c_1:

$$\frac{\partial \ell}{\partial c_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \, \text{sigmoid}} \frac{\partial \, \text{sigmoid}}{\partial (W_1 x + c_1)} \frac{\partial (W_1 x + c_1)}{\partial c_1} \, .$$

The derivative of the linear transformation $(W_1 x + c_1)$ with respect to W_1 is simply x:

$$\frac{\partial (W_1 x + c_1)}{\partial c_1} = 1 \, .$$

Therefore, combining the above equations, we get:

$$\begin{aligned}
&\frac{\partial \ell}{\partial W_1} \\
&= -\frac{y}{\hat{y}} \text{softmax}(W_2 \, \text{sigmoid}(W_1 x + c_1) + c_2) \\
&\quad (1 - \text{softmax}(W_2 \, \text{sigmoid}(W_1 x + c_1) + c_2)) \\
&\quad \text{sigmoid}(W_1 x + c_1)(1 - \text{sigmoid}(W_1 x + c_1)) \, .
\end{aligned}$$