CS480/680, Spring 2023, Assignment 2

Designer: Yimu Wang; Instructor: Hongyang Zhang

Released: June 5; Due: June 25, 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.)
- Your grade is independent of the test accuracy (unless it is 10%, as 10% is the accuracy of random guess).

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

Conceptually the weights lie in between layers. Since the input layer has 100 nodes and each of the 100 nodes has connections to the 50 nodes in the first hidden layer, we can represent the connections in a $\mathbb{R}^{100\times 50}$ matrix. For each node in the first hidden layer, there is a bias term which can be represented in a $\mathbb{R}^{50}$ matrix. The first hidden layer has 50 nodes, each of which connect to the 25 nodes of the second hidden layer. We can represent these connections in a $\mathbb{R}^{50\times 25}$ matrix. Each node of the second hidden layer also has a bias term which can be represented in a $\mathbb{R}^{25}$ matrix. Lastly, each of the 25 nodes in the second hidden layer connects to all of the 10 nodes in the output layer. These weights can be represented in a $\mathbb{R}^{25\times 10}$ matrix. Each one of the output nodes also has a bias term which can be represented in a $\mathbb{R}^{10}$ matrix. Therefore the total number of parameters is

$$100 \times 50 + 50 + 50 \times 25 + 25 + 25 \times 10 + 10 = 6585$$

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,

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

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

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

For MSE the calculation is as follows

$$\ell_{MSE}(\hat{y}, y) = \sum_{i \in C} (\hat{y}_c - y_c)^2 = (0.5 - 0.3)^2 + (0.3 - 0.6)^2 + (0.8 - 0.1)^2 = 0.62$$

For the CE calculation

$$\begin{aligned}
\ell_{CE}(\hat{y}, y) &= -\sum_{i=1}^{C} y_i \log\left(\frac{\exp(\hat{y}_i)}{\sum_{j \in [C]} \exp(\hat{y}_j)}\right) \\
&= -\left(0.3 \log\left(\frac{e^{0.5}}{e^{0.5} + e^{0.3} + e^{0.8}}\right) + 0.6 \log\left(\frac{e^{0.3}}{e^{0.5} + e^{0.3} + e^{0.8}}\right) + 0.1 \log\left(\frac{e^{0.8}}{e^{0.5} + e^{0.3} + e^{0.8}}\right)\right) \\
&= -1.24328655533
\end{aligned}$$

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.]

According to lecture notes, for an input of size $m \times n \times c$, filter size of $a \times b \times c$, stride of size $s \times t$ and padding of size $p \times q$, the output size is $\lfloor 1 + \frac{m+2p-a}{s} \rfloor \times \lfloor 1 + \frac{n+2q-b}{t} \rfloor$. In our example, the padding is 0 therefore $p = q = 0$, the stride is 1 therefore $s = t = 1$, thre input image is $32 \times 32 \times 1$ therefore $m = n = 32$, the filter is $3 \times 3 \times 1$ therefore $a = b = 3$. The formula yeilds that the output dimension is

$$\lfloor 1 + \frac{m + 2p - a}{s} \rfloor \times \lfloor 1 + \frac{n + 2q - b}{t} \rfloor = \lfloor 1 + \frac{32 - 3}{1} \rfloor \times \lfloor 1 + \frac{32 - 3}{1} \rfloor = 30 \times 30$$

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.]

In the attention mechanism there are learnable parameters attached with each Q, K, V. Since we are doing self attention this means $Q = K = V$. Since $W^k, W^q, W^v \in \mathbb{R}^{512 \times 64}$ this implies there are $3 \times 512 \times 64 = 98304$ learnable params. Since there are $h = 8$ heads, and each head has its own set of learnable parameters we multiply the previous result by 8 getting $8 \times 98304 = 786432$. Lastly we need to add the final projection layer after the concatenation operation. This matrix $W^o \in \mathbb{R}^{512 \times 512}$ because $h = 8$ and $d_{model}/h = 64$. Therefore this final projection layer has $512 \times 512 = 262144$ learnable parameters. This gives a total of $786432 + 262144 = 1048576$ learnable parameters.

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

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

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. $\mathrm{softmax}$ and $\mathrm{sigmoid}$ 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., $\sum_{i \in [C]} \hat{y}_i = 1$.

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 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.

In [ ]:
```
### 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
```

```
/Users/alexivanov/opt/miniconda3/envs/cs479/lib/python3.10/site-packages/tqdm/auto.py:22: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

2.1 Implement the MLP with PyTorch (15 points)

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

You can refer to Define a NN for inspiration.

In [ ]:
```
import torch.nn as nn
import torchvision.models as models

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

# 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

        self.hidden1 = torch.nn.Linear(input_dim, hidden_dim)
        self.output = torch.nn.Linear(hidden_dim, output_dim)

        self.act1 = torch.nn.Sigmoid()
        self.act2 = torch.nn.Softmax(dim=1)

        # random init
        torch.manual_seed(1) # seed for random init
        with torch.no_grad():
            self.hidden1.weight = torch.nn.Parameter(torch.rand(self.hidden1.weight.shape))
            self.hidden1.bias = torch.nn.Parameter(torch.rand(self.hidden1.bias.shape))
            self.output.weight = torch.nn.Parameter(torch.rand(self.output.weight.shape))
            self.output.bias = torch.nn.Parameter(torch.rand(self.output.bias.shape))
```

```
        #reset seed back to 480
        torch.manual_seed(480)

    def forward(self, x):
        # TODO: Your code here: forward
        x = self.hidden1(x)
        x = self.act1(x)
        x = self.output(x)
        x = self.act2(x)
        return x
```

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.

In [ ]:
```
##### COMPLETE THE FOLLOWING CODE, YOU ARE ALLOWED TO CHANGE THE PARAMETERS#####
def accuracy(y, predicted):
    # TODO: Your code here: calculate accuracy
    """
    y: list of tensors. Each element of the list is a batch of one-hot encoded true label
    predicted: list of tensors. Each element of the list is a batch of one-hot encoded predictions
    """
    y = torch.cat(y)
    predicted = torch.cat(predicted)
    correct = 0

    return sum(y == predicted) / len(y)
```

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

In [ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
##### COMPLETE THE FOLLOWING CODE, YOU ARE ALLOWED TO CHANGE THE PARAMETERS#####

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 train_dataloader:
```

```
        inputs = inputs.view(-1, 3072).to(device)
        train_labels.append(labels)

        # Forward pass
        predictions = model(inputs)
        predictions = torch.argmax(predictions, dim=1)
        predict_train.append(predictions)

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

        # Forward pass
        predictions = model(inputs)
        predictions = torch.argmax(predictions, dim=1)
        predict_test.append(predictions)

    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(f"train_acc: {train_acc: .3f}")
print(f"test_acc: {test_acc: .3f}")
```

```
Files already downloaded and verified
train_acc:  0.100
test_acc:  0.100
```

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: [Your answer here]

Lets first define some intermediate variables to make the step by step derivation easier. Let

$$A = W_1 x + c_1$$
$$B = W_2 sigmoid(A) + c_2$$

Now we can view the MLP as

$$\hat{y} = softmax(B)$$

Now we can define the partial derivatives and begin the derivation

$$\frac{\partial \ell}{\partial W_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial B} \frac{\partial B}{\partial A} \frac{\partial A}{\partial W_1}$$
$$\frac{\partial \ell}{\partial c_1} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial B} \frac{\partial B}{\partial A} \frac{\partial A}{\partial c_1}$$
$$\frac{\partial \ell}{\partial W_2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial B} \frac{\partial B}{\partial W_2}$$
$$\frac{\partial \ell}{\partial W_2} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial B} \frac{\partial B}{\partial c_2}$$

We notice that all of the required derivatives begin with $\frac{\partial \ell}{\partial \hat{y}}$. Since $\hat{y}$ is a vector the gradient of the loss with respect to $\hat{y}$ is also a vector given by

$$\frac{\partial \ell}{\partial \hat{y}} = \begin{bmatrix} \frac{\partial \ell}{\partial \hat{y}_1} & \frac{\partial \ell}{\partial \hat{y}_2} & \cdots & \frac{\partial \ell}{\partial \hat{y}_C} \end{bmatrix} = \begin{bmatrix} \frac{-y_1}{\hat{y}_1} & \frac{-y_2}{\hat{y}_2} & \cdots & \frac{-y_C}{\hat{y}_C} \end{bmatrix}$$

Now $\frac{\partial \hat{y}}{\partial B}$. Notice that $\hat{y}$ is a probability vector therefore the gradient should be

$$\frac{\partial \hat{y}}{\partial B} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial B} & \frac{\partial \hat{y}_2}{\partial B} & \cdots & \frac{\partial \hat{y}_C}{\partial B} \end{bmatrix}$$

However, B itself is a vector with C dimensions. Therefore the gradient $\frac{\partial \hat{y}}{\partial B}$ is a Jacobian

$$\frac{\partial \hat{y}}{\partial B} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial B_1} & \frac{\partial \hat{y}_1}{\partial B_2} & \cdots & \frac{\partial \hat{y}_1}{\partial B_C} \\ \frac{\partial \hat{y}_2}{\partial B_1} & \frac{\partial \hat{y}_2}{\partial B_2} & \cdots & \frac{\partial \hat{y}_2}{\partial B_C} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_C}{\partial B_1} & \frac{\partial \hat{y}_C}{\partial B_2} & \cdots & \frac{\partial \hat{y}_C}{\partial B_C} \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} \frac{e^{B_1}(\sum_{i=1}^{C} e^{B_i}) - e^{B_1}e^{B_1}}{(\sum_{i=1}^{C} e^{B_i})^2} & \frac{0 \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_1}e^{B_2}}{(\sum_{i=1}^{C} e^{B_i})^2} & \cdots & \frac{0 \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_1}e^{B_C}}{(\sum_{i=1}^{C} e^{B_i})^2} \\ \frac{0 \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_2}e^{B_1}}{(\sum_{i=1}^{C} e^{B_i})^2} & \frac{e^{B_2} \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_2}e^{B_2}}{(\sum_{i=1}^{C} e^{B_i})^2} & \cdots & \frac{0 \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_2}e^{B_C}}{(\sum_{i=1}^{C} e^{B_i})^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{0 \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_C}e^{B_1}}{(\sum_{i=1}^{C} e^{B_i})^2} & \frac{0 \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_C}e^{B_2}}{(\sum_{i=1}^{C} e^{B_i})^2} & \cdots & \frac{e^{B_C} \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_C}e^{B_C}}{(\sum_{i=1}^{C} e^{B_i})^2} \end{bmatrix} \tag{2}$$

$$= \begin{bmatrix} \frac{e^{B_1}(\sum_{i=1}^{C} e^{B_i}) - e^{B_1}e^{B_1}}{(\sum_{i=1}^{C} e^{B_i})^2} & \frac{-e^{B_1}e^{B_2}}{(\sum_{i=1}^{C} e^{B_i})^2} & \cdots & \frac{-e^{B_1}e^{B_C}}{(\sum_{i=1}^{C} e^{B_i})^2} \\ \frac{-e^{B_2}e^{B_1}}{(\sum_{i=1}^{C} e^{B_i})^2} & \frac{e^{B_2} \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_2}e^{B_2}}{(\sum_{i=1}^{C} e^{B_i})^2} & \cdots & \frac{-e^{B_2}e^{B_C}}{(\sum_{i=1}^{C} e^{B_i})^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-e^{B_C}e^{B_1}}{(\sum_{i=1}^{C} e^{B_i})^2} & \frac{-e^{B_C}e^{B_2}}{(\sum_{i=1}^{C} e^{B_i})^2} & \cdots & \frac{e^{B_C} \times (\sum_{i=1}^{C} e^{B_i}) - e^{B_C}e^{B_C}}{(\sum_{i=1}^{C} e^{B_i})^2} \end{bmatrix} \tag{3}$$

$$= \begin{bmatrix} \frac{e^{B_1}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{\sum_{i=1}^{C} e^{B_i} - e^{B_1}}{\sum_{i=1}^{C} e^{B_i}} & \frac{-e^{B_1}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{e^{B_2}}{\sum_{i=1}^{C} e^{B_i}} & \cdots & \frac{-e^{B_1}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{e^{B_C}}{\sum_{i=1}^{C} e^{B_i}} \\ \frac{-e^{B_2}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{e^{B_1}}{\sum_{i=1}^{C} e^{B_i}} & \frac{e^{B_2}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{\sum_{i=1}^{C} e^{B_i} - e^{B_2}}{\sum_{i=1}^{C} e^{B_i}} & \cdots & \frac{-e^{B_2}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{e^{B_C}}{\sum_{i=1}^{C} e^{B_i}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{-e^{B_C}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{e^{B_1}}{\sum_{i=1}^{C} e^{B_i}} & \frac{-e^{B_C}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{e^{B_2}}{\sum_{i=1}^{C} e^{B_i}} & \cdots & \frac{e^{B_C}}{\sum_{i=1}^{C} e^{B_i}} \times \frac{\sum_{i=1}^{C} e^{B_i} - e^{B_C}}{\sum_{i=1}^{C} e^{B_i}} \end{bmatrix} \tag{4}$$

$$= \begin{bmatrix} \hat{y}_1 \times (1 - \hat{y}_1) & -\hat{y}_1 \times \hat{y}_2 & \cdots & -\hat{y}_1 \times \hat{y}_C \\ -\hat{y}_2 \times \hat{y}_1 & \hat{y}_2 \times (1 - \hat{y}_2) & \cdots & -\hat{y}_2 \times \hat{y}_C \\ \vdots & \vdots & \ddots & \vdots \\ -\hat{y}_C \times \hat{y}_1 & -\hat{y}_C \times \hat{y}_2 & \cdots & \hat{y}_C \times (1 - \hat{y}_C) \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} \hat{y}_C \times \hat{y}_1 & \hat{y}_C \times \hat{y}_2 & \hat{y}_C \times (1 - \hat{y}_C) \end{bmatrix}$$

Now to find

$$\frac{\partial \ell}{\partial B} = \frac{\partial \ell}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial B} \tag{6}$$

$$= \begin{bmatrix} \frac{-y_1}{\hat{y}_1} & \frac{-y_2}{\hat{y}_2} & \cdots & \frac{-y_C}{\hat{y}_C} \end{bmatrix} \cdot \begin{bmatrix} \hat{y}_1 \times (1 - \hat{y}_1) & -\hat{y}_1 \times \hat{y}_2 & \cdots & -\hat{y}_1 \times \hat{y}_C \\ -\hat{y}_2 \times \hat{y}_1 & \hat{y}_2 \times (1 - \hat{y}_2) & \cdots & -\hat{y}_2 \times \hat{y}_C \\ \vdots & \vdots & \ddots & \vdots \\ -\hat{y}_C \times \hat{y}_1 & -\hat{y}_C \times \hat{y}_2 & \cdots & \hat{y}_C \times (1 - \hat{y}_C) \end{bmatrix} \tag{7}$$

$$= \begin{bmatrix} -y_1(1 - \hat{y}_1) + \sum_{i=1, i \neq 1}^{C} y_i \hat{y}_1 & -y_2(1 - \hat{y}_2) + \sum_{i=1, i \neq 2}^{C} y_i \hat{y}_2 & \cdots & -y_C(1 - \hat{y}_C) + \sum_{i=1, i \neq C}^{C} y_i \hat{y}_C \end{bmatrix} \tag{8}$$

$$= \begin{bmatrix} -y_1 + y_1 \hat{y}_1 + \sum_{i=1, i \neq 1}^{C} y_i \hat{y}_1 & -y_2 + y_2 \hat{y}_2 + \sum_{i=1, i \neq 2}^{C} y_i \hat{y}_2 & \cdots & -y_C + y_C \hat{y}_C + \sum_{i=1, i \neq C}^{C} y_i \hat{y}_C \end{bmatrix} \tag{9}$$

$$= \begin{bmatrix} -y_1 + \sum_{i=1}^{C} y_i \hat{y}_1 & -y_2 + \sum_{i=1}^{C} y_i \hat{y}_2 & \cdots & -y_C + \sum_{i=1}^{C} y_i \hat{y}_C \end{bmatrix} \tag{10}$$

$$= \begin{bmatrix} -y_1 + \hat{y}_1 \sum_{i=1}^{C} y_i & -y_2 + \hat{y}_2 \sum_{i=1}^{C} y_i & \cdots & -y_C + \hat{y}_C \sum_{i=1}^{C} y_i \end{bmatrix} \tag{11}$$

$$= \begin{bmatrix} -y_1 + \hat{y}_1 & -y_2 + \hat{y}_2 & \cdots & -y_C + \hat{y}_C \end{bmatrix} \tag{12}$$

$$= \hat{\mathbf{y}} - \mathbf{y} \tag{13}$$

Now we find $\frac{\partial B}{\partial A}$

$$\frac{\partial B}{\partial A} = W_2 \times sigmoid'(A) \tag{14}$$

where $sigmoid'(A) = sigmoid(A) \times (1 - sigmoid(A))$ Now

$$\frac{\partial A}{\partial W_1} = x$$

and

$$\frac{\partial A}{\partial c_1} = \vec{1}$$

We can also find that

$$\frac{\partial B}{\partial W_2} = sigmoid(A)$$

and

$$\frac{\partial B}{\partial c_2} = \vec{1}$$

Putting it all together we get that

$$\frac{\partial \ell}{\partial W_1} = (\hat{\mathbf{y}} - \mathbf{y}) \times W_2 \times sigmoid'(A) \times x$$

$$\frac{\partial \ell}{\partial c_1} = (\hat{\mathbf{y}} - \mathbf{y}) \times W_2 \times sigmoid'(A)$$

$$\frac{\partial \ell}{\partial W_2} = (\hat{\mathbf{y}} - \mathbf{y}) \times sigmoid(A)$$

$$\frac{\partial \ell}{\partial c_2} = (\hat{\mathbf{y}} - \mathbf{y})$$