[FMAN45] - Assignment3

Filip Kalkan (fi1231ka-s)

May 2022

1 Exercise 1

Firstly, we simplify the equation

$$\frac{dL}{dx_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{dy_l}{dx_i} \tag{1}$$

as follows.

$$\frac{dL}{dx_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{d}{dx_i}\left(\sum_{j=1}^{m} W_{lj}x_j + b_j\right) = \sum_{l=1}^{n} \frac{dL}{dy_l} W_{li} \tag{2}$$

Translating the result to apply to the entire vector $\mathbf{x}$ results in

$$\frac{L}{d\mathbf{x}} = \begin{bmatrix} W_{1,1} & W_{2,1} & \dots & W_{m,1} \\ W_{1,2} & W_{2,2} & \dots & W_{m,2} \\ \vdots & \ddots & & \vdots \\ W_{1,n} & W_{2,n} & \dots & W_{m,n} \end{bmatrix} \begin{bmatrix} \frac{dL}{dy_1} \\ \frac{dL}{dy_2} \\ \vdots \\ \frac{dL}{dy_m} \end{bmatrix} = \mathbf{W}^T \frac{dL}{d\mathbf{y}} \tag{3}$$

Moving on to the equation

$$\frac{dL}{dW_{ij}} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{dy_l}{dW_{ij}} \tag{4}$$

we can apply similar computations.

$$\frac{dL}{dW_{ij}} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{d}{dW_{ij}}\left(\sum_{k=1}^{m} W_{lk}x_k + b_l\right) = \tag{5}$$

$$= \sum_{l=1}^{n} \frac{dL}{dy_l}\left(\sum_{k=1}^{m} x_k \frac{d}{dW_{ij}} W_{lk} + b_l\right) \tag{6}$$

Now, as $\frac{d}{dW_{ij}} W_{lk} = 1$ if $i = l$ and $j = k$ and 0 otherwise, we can simplify the equation to

$$\frac{dL}{dW_{ij}} = \frac{dL}{dy_i} x_j \tag{7}$$

Translating the result to apply to $\mathbf{W}$ results in

$$\frac{dL}{d\mathbf{W}} = \begin{bmatrix} \frac{dL}{dy_1} \\ \frac{dL}{dy_2} \\ \vdots \\ \frac{dL}{dy_m} \end{bmatrix} \begin{bmatrix} x_1 & x_2 & \dots & x_m \end{bmatrix} = \frac{dL}{d\mathbf{y}}\mathbf{x}^T \tag{8}$$

Finally we investigate

$$\frac{dL}{db_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{dy_l}{db_i}. \tag{9}$$

This can be expanded as

$$\frac{dL}{db_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{d}{db_i}\left(\sum_{j=1}^{m} W_{lj}x_j + b_j\right) \tag{10}$$

Once again $\frac{db_j}{db_i} = 1$ if $i = j$ and 0 otherwise. Thus,

$$\frac{dL}{db_i} = \frac{dL}{dy_i} \tag{11}$$

which, when concidering $\mathbf{b}$ translates to

$$\frac{dL}{d\mathbf{b}} = \frac{dL}{d\mathbf{y}} \tag{12}$$

2 Exercise 2

In order to derive the expressions, we make use of the results from Exercise 1. We start off with

$$\frac{dL}{d\mathbf{X}} = \begin{bmatrix} \mathbf{W}^T\frac{dL}{d\mathbf{y}^{(1)}} & \mathbf{W}^T\frac{dL}{d\mathbf{y}^{(2)}} & \dots & \mathbf{W}^T\frac{dL}{d\mathbf{y}^{(N)}} \end{bmatrix} = \tag{13}$$

$$= \mathbf{W}^T \begin{bmatrix} \frac{dL}{d\mathbf{y}^{(1)}} & \frac{dL}{d\mathbf{y}^{(2)}} & \dots & \frac{dL}{d\mathbf{y}^{(N)}} \end{bmatrix} = \mathbf{W}^T\frac{dL}{d\mathbf{Y}} \tag{14}$$

Furthermore, we can rewrite $\mathbf{Y}$ as

$$\mathbf{Y} = \begin{bmatrix} \mathbf{W}\mathbf{x}^{(1)} + \mathbf{b} & \mathbf{W}\mathbf{x}^{(2)} + \mathbf{b} & \dots & \mathbf{W}\mathbf{x}^{(N)} + \mathbf{b} \end{bmatrix} = \mathbf{W}\mathbf{X} + \mathbf{b} \tag{15}$$

where $\mathbf{b}$ is added to each column of $\mathbf{WX}$.

Using equation (7), we find that

$$\frac{dL}{d\mathbf{W}} = \frac{dL}{d\mathbf{Y}}\mathbf{X}^T. \tag{16}$$

As for the bias, we can use equation (12) to express it as

$$\frac{dL}{d\mathbf{b}} = \sum_{i=1}^{N} \frac{dL}{d\mathbf{Y}^{(i)}} \tag{17}$$

Equation (15) was implemented as

```
b = repmat(b, 1, batch);
Y = W * X + b;
```

and (14), (16) and (17) were implemented as

```
dldX = W' * dldY;
dldX = reshape(dldX, sz);
dldW = dldY * X';
dldb = sum(dldY, 2);
```

3 Exercise 3

$$\frac{dL}{dx_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{dy_l}{dx_i} = \sum_{l=1}^{n} \frac{dL}{dy_l}\frac{d}{dx_i}\Big(max(x_l, 0)\Big) = \tag{18}$$

$$= \begin{cases} \frac{dL}{dy_i} & \text{if } x_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

The forward pass was implemented as

```
function Y = relu_forward(X)
    Y = max(X, 0);
end
```

and the backward pass as

```
function dldX = relu_backward(X, dldY)
    dldX = heaviside(X) .* dldY;
end
```

4 Exercise 4

Using

$$L(\mathbf{x}, c) = -x_c + log\Bigg(\sum_{j=1}^{m} e^{x_j}\Bigg) \tag{20}$$

we can derive an equation for $\frac{dL}{dx_i}$ as follows

$$\frac{dL}{dx_i} = \frac{d}{dx_i}\Bigg(-x_c + log\Bigg(\sum_{j=1}^{m} e^{x_j}\Bigg)\Bigg) = \tag{21}$$

$$= \begin{cases} \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} - 1 & , i = c \\ \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} & , i \neq c \end{cases} = \begin{cases} y_i - 1 & , i = c \\ y_i & , i \neq c \end{cases} \tag{22}$$

The softmax forward pass was implemented as

```
    i_equals_c_index = sub2ind(sz, labels', 1:batch);
    y = exp(x) ./ sum(exp(x));
    negative_log_y = -log(y);
    L = mean(negative_log_y(i_equals_c_index));
```

and the backward pass as

```
    i_equals_c_index = sub2ind(sz, labels', 1:batch);
    dldx = exp(x) ./ sum(exp(x));
    dldx(i_equals_c_index) = dldx(i_equals_c_index) - 1;
    dldx = dldx ./ batch;
```

5 Exercise 5

Gradient descent with momentum was implemented as

```
momentum{i}.(s) = opts.moving_average * momentum{i}.(s) +
     (1 - opts.moving_average) * (grads{i}.(s) +... opts.
    weight_decay * net.layers{i}.params.(s));

net.layers{i}.params.(s) = net.layers{i}.params.(s) -
    opts.learning_rate * momentum{i}.(s);
```

6 Exercise 6

The filters used in the first layer of the network are shown in figure 1. Judging by their appearance they seem to be detecting lines. This might be expected from a network which classifies images of digits.

The confusion matrix in figure 2 shows how the model classified the images in the test data set in relation to the images' true classes. As shown in the matrix, the network classified the numbers 2 and 3 with quite high accuracy. However, it also shows that the network often confused - for instance - images of the digit 9 with 4.

Looking at the top-left image in figure 3, it is clear that there are some grounds for confusion between the digits 9 and 4. Also, some other images indicate that some misclassifications are to be expected as I personally find it hard to classify them.

When evaluated on the test set, the network performed well as can be seen in table 6. The classes with the lowest precision are 5 and 8. Even though the model found it relatively hard to correctly predict that an image belonged to one of these classes, it managed to do so in 96% of the cases. In the remaining 4% of the cases the model incorrectly predicted that the images belonged to the aforementioned classes. That is, it predicted that there were more 5's and 8's than there acually were. However, concidering the recall rate, the model managed to classify 99% and 98% of the images belonging to the respective

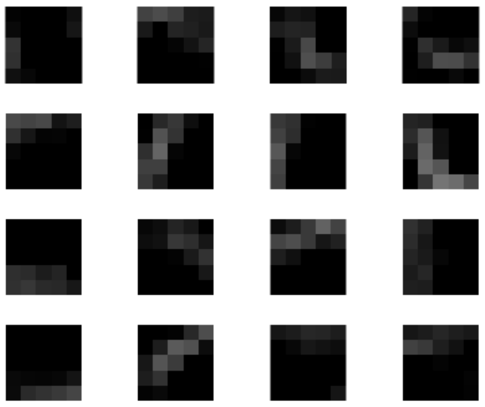


Figure 1: Images representing the convolutional filters used in the first layer of the network.

classes correctly. In other words, the model predicted that 99% of the actual 5's were 5's.

Class	0	1	2	3	4	5	6	7	8	9
Precision	0.99	0.97	0.98	0.98	0.97	0.96	0.99	0.98	0.96	0.98
Recall	0.98	0.99	0.97	0.98	0.99	0.99	0.97	0.94	0.98	0.96

Table 1: Precision & recall for the test set.

The model contained a total of 14682 trainable weights. These weights were located in the convolutional- and fully connected layers. The input-, relu-, max pooling- and softmax loss layers had no trainable parameters. A more detailed summary of the parameter distribution can be found in table 2.

7 Exercise 7

7.1 Model Improvements

The test accuracy of the unaltered model was 0.4547.

Increasing the number of training images from 20000 to 50000 resulted in a test accuracy of 0.4369, which was quite strange. Presumably, the decrease was due to the stochastic nature of the training process.

After this, another fully connected layer was added the existing one. This layer had 2048 nodes. It resulted in a test accuracy of 0.4336. In other words, no significant difference. The layer was then removed, and the network structure restored to it's original state.

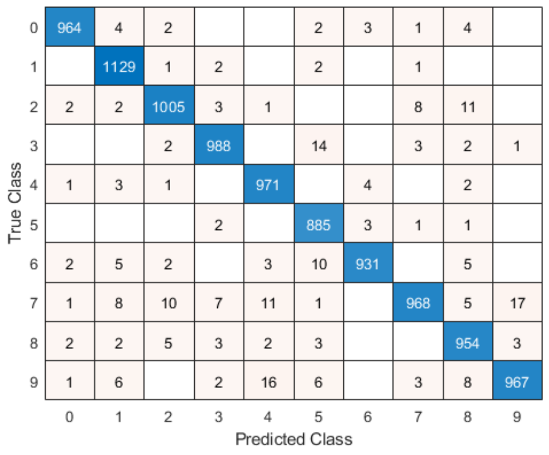


Figure 2: Confusion matrix showing the network's classification of the test data set.

Layer Name	**Number of Weights**	**Number of Biases**	**#Trainable Parameters**
Input	0	0	0
Convolution	400	16	416
Relu	0	0	0
MaxPooling	0	0	0
Convolution	6400	16	6416
Relu	0	0	0
MaxPooling	0	0	0
Fully Connected	7840	10	7850
SoftmaxLoss	0	0	0
Total	14640	42	**14682**

Table 2: Number of trainable parameters by layer in the MNIST model.

Now, a third convolutional layer was added after the second. This layer was identical to the second one. This network managed to obtain an accuracy of **0.5229**

7.2 Model Analysis

The filters of the first convolutional layer (seen in figure 4) mostly look like noise, no clear interpretable structure is visible, unlike the corresponding filters of the MNIST classifier. The strange visualizations are understandable since the data in the CIFAR data set has higher dimensionality and higher variance.

As the CIFAR data set has text labels, a mapping between digits and numbers are needed to make sense of the model's predictions. The used mapping is presented in table 3.

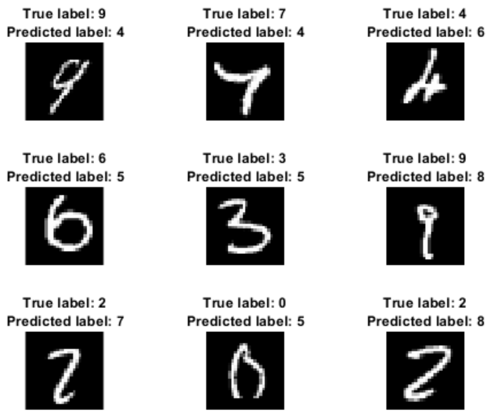


Figure 3: Some of the samples which the model failed to classify correctly in the test set.

Class Number	1	2	3	4	5	6	7	8	9	10
Class Text	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck

Table 3: The mapping between numbers and text used along with the CIFAR model.

The confusion matrix in figure 5 shows that the model quite often misclassifies images. However, there is an interesting pattern to be found. Namely, that the model often manages to tell animals and vehicles apart.

Class	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
Precision	0.6832	0.5878	0.3503	0.4225	0.4292	0.5254	0.5170	0.5180	0.6761	0.5587
Recall	0.4270	0.7330	0.4200	0.2590	0.3910	0.4130	0.6980	0.6760	0.6200	0.5850

Table 4: Precision and recall for the CIFAR model.

The number of trainable parameters are presented in table 5. As the total amount of trainable parameters (55018) exceeds that of the MNIST model, and the amount of data does as well, it is clear that this is a harder task to solve given the accuracy.

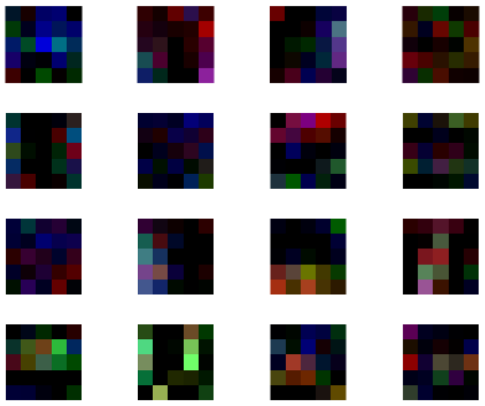
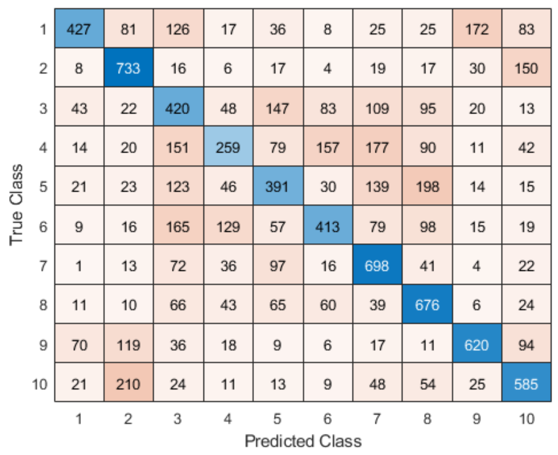


Figure 4: Visualization of the filters in the first convolutional layer of the CIFAR model.

Layer Name	**Number of Weights**	**Number of Biases**	**#Trainable Parameters**
Input	0	0	0
Convolution	1200	16	1216
Relu	0	0	0
MaxPooling	0	0	0
Convolution	6400	16	6416
Relu	0	0	0
Convolution	6400	16	6416
Relu	0	0	0
Fully Connected	40960	10	40970
SoftmaxLoss	0	0	0
Total	54960	58	**55018**

Table 5: Number of trainable parameters by layer in the CIFAR model.



True Class \ Predicted Class	1	2	3	4	5	6	7	8	9	10
1	427	81	126	17	36	8	25	25	172	83
2	8	733	16	6	17	4	19	17	30	150
3	43	22	420	48	147	83	109	95	20	13
4	14	20	151	259	79	157	177	90	11	42
5	21	23	123	46	391	30	139	198	14	15
6	9	16	165	129	57	413	79	98	15	19
7	1	13	72	36	97	16	698	41	4	22
8	11	10	66	43	65	60	39	676	6	24
9	70	119	36	18	9	6	17	11	620	94
10	21	210	24	11	13	9	48	54	25	585

Figure 5: Confusion matrix for the CIFAR model.

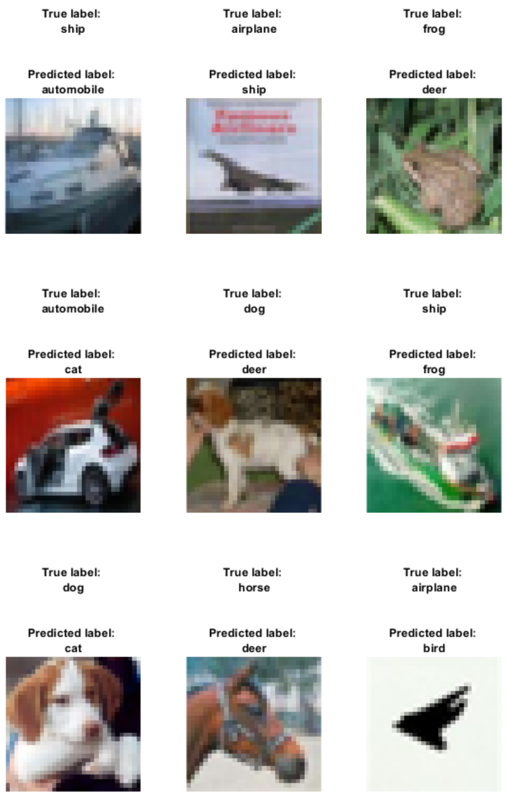


Figure 6: Some of the samples which the CIFAR model failed to classify correctly in the test set.