Assignment 3 - Machine Learning

Andreas Timürtas

May 2021

Exercise 1

In figure 1 we have an example of how a neural network can look like. Here we have three inputs x_j and two outputs z_i with biases. We will use this figure to derive the expressions for $\frac{\partial L}{\partial \mathbf{x}}$, $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \mathbf{b}}$.

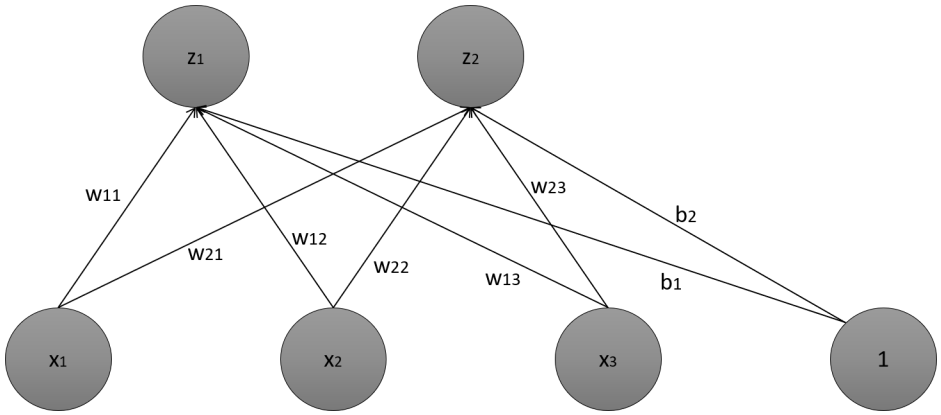

Figure 1: Illustration of a neural network with three inputs x_j and two outputs z_i with weights between the connections. Weight w_{ij} is connecting input x_j and output z_i. b_1 and b_2 are the biases.

We first want to derive $\frac{\partial L}{\partial \mathbf{x}}$ and start by deriving the partial derivative of L with respect to x_1 in the neural network in figure 1.

$$\frac{\partial L}{\partial x_1} = \sum_{i=1}^{2} \frac{\partial L}{\partial z_i}\frac{\partial z_i}{\partial x_1} = \tag{1}$$

$$= \frac{\partial L}{\partial z_1}\frac{\partial}{\partial x_1}(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1) + \frac{\partial L}{\partial z_2}\frac{\partial}{\partial x_1}(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2) = \tag{2}$$

$$= \frac{\partial L}{\partial z_1}w_{11} + \frac{\partial L}{\partial z_2}w_{21} = \begin{bmatrix} \frac{\partial L}{\partial z_1} & \frac{\partial L}{\partial z_2} \end{bmatrix} \begin{bmatrix} w_{11} \\ w_{21} \end{bmatrix} \tag{3}$$

We notice that if we differentiate with respect to x_2 instead we only have to change w_{11} and w_{21} with w_{12} respectively w_{22}. In the general case with m inputs x_j and n outputs z_i the partial derivative with respect to input x_i we would get the expression

$$\frac{\partial L}{\partial x_i} = \begin{bmatrix} \frac{\partial L}{\partial z_1}, \ldots, \frac{\partial L}{\partial z_n} \end{bmatrix} \begin{bmatrix} w_{1i} \\ \vdots \\ w_{ni} \end{bmatrix} = \frac{\partial L}{\partial \mathbf{z}}^{\top} \begin{bmatrix} w_{1i} \\ \vdots \\ w_{ni} \end{bmatrix} \tag{4}$$

Doing this for all inputs x we get our desired expression of $\frac{\partial L}{\partial \mathbf{x}}$.

$$\frac{\partial L}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial L}{\partial x_1} \\ \vdots \\ \frac{\partial L}{\partial x_m} \end{bmatrix} = \begin{bmatrix} w_{11} & \dots & w_{n1} \\ \vdots & \ddots & \vdots \\ w_{1m} & \dots & w_{nm} \end{bmatrix} \begin{bmatrix} \frac{\partial L}{\partial z_1} \\ \vdots \\ \frac{\partial L}{\partial z_n} \end{bmatrix} = \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{z}} \tag{5}$$

Next we want to derive $\frac{\partial L}{\partial \mathbf{W}}$ and notice that the partial derivative of L with respect to w_{11} in our network from figure 1 is equal to

$$\frac{\partial L}{\partial w_{11}} = \sum_{i=1}^{2} \frac{\partial L}{\partial z_i} \frac{\partial z_i}{\partial w_{11}} = \frac{\partial L}{\partial z_1} x_1. \tag{6}$$

In the general case the partial derivative of L with respect to weight w_{ij} would then have the expression

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial z_i} x_j. \tag{7}$$

The matrix expression can then be expressed as

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial L}{\partial w_{11}} & \dots & \frac{\partial L}{\partial w_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{n1}} & \dots & \frac{\partial L}{\partial w_{nm}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial z_1} x_1 & \dots & \frac{\partial L}{\partial z_1} x_m \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial z_n} x_1 & \dots & \frac{\partial L}{\partial z_n} x_m \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial z_1} \\ \vdots \\ \frac{\partial L}{\partial z_n} \end{bmatrix} \begin{bmatrix} x_1, \dots, x_m \end{bmatrix} = \frac{\partial L}{\partial \mathbf{z}} \cdot \mathbf{x}^\top. \tag{8}$$

Finally we was to derive the partial derivative of L with respect to $\mathbf{b}$. From our network in figure 1 we can derive that the derivative of L with respect to b_1 is equal to

$$\frac{\partial L}{\partial b_1} = \sum_{i=1}^{2} \frac{\partial L}{\partial z_i} \frac{\partial z_i}{\partial b_1} = \frac{\partial L}{\partial z_1}. \tag{9}$$

Notice that only z_1 includes the bias b_1. The general case for the partial derivative of L with respect to the bias b_i is then equal to

$$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial z_i} \tag{10}$$

and in matrix notation we get that

$$\frac{\partial L}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial L}{\partial b_1} \\ \vdots \\ \frac{\partial L}{\partial b_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial z_1} \\ \vdots \\ \frac{\partial L}{\partial z_n} \end{bmatrix} = \frac{\partial L}{\partial \mathbf{z}}. \tag{11}$$

The resulting expression in matrix notations is then:

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{z}}, \quad \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \mathbf{z}} \mathbf{x}^\top, \quad \frac{\partial L}{\partial \mathbf{b}} = \frac{\partial L}{\partial \mathbf{z}}. \tag{12}$$

Exercise 2

In this exercise we want to derive expressions for $\mathbf{Z}$, $\frac{\partial L}{\partial \mathbf{X}}$, $\frac{\partial L}{\partial \mathbf{W}}$ and $\frac{\partial L}{\partial \mathbf{b}}$ in terms of $\frac{\partial L}{\partial \mathbf{Z}}$, $\mathbf{W}$, $\mathbf{X}$ and $\mathbf{b}$. Here $\mathbf{X}$ is a batch of N input elements $\mathbf{x}$ and $\mathbf{Z}$ is then a corresponding batch of N outputs $\mathbf{z}$. In mathematical expressions:

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

$\mathbf{Z}$ can then be simplified to the expression

$$\mathbf{Z} = \mathbf{W}\mathbf{X} + \mathbf{b}. \tag{13}$$

We know want to derive an expression of $\frac{\partial L}{\partial \mathbf{X}}$. We have that

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial L}{\partial \mathbf{x}^{(1)}} & \frac{\partial L}{\partial \mathbf{x}^{(2)}} & \cdots & \frac{\partial L}{\partial \mathbf{x}^{(N)}} \end{bmatrix} \tag{14}$$

and from the previous exercises we know that

$$\frac{\partial L}{\partial \mathbf{x}} = \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{z}}. \tag{15}$$

Putting these two together we get

$$\frac{\partial L}{\partial \mathbf{X}} = \begin{bmatrix} \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{z}^{(1)}} & \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{z}^{(1)}} & \cdots & \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{z}^{(1)}} \end{bmatrix} = \mathbf{W}^\top \frac{\partial L}{\partial \mathbf{Z}} \tag{16}$$

We know move on to express $\frac{\partial L}{\partial \mathbf{W}}$ using the chain rule with respect to each weight in each $\mathbf{z}^{(i)}$ and sum them up. This means that the partial derivative of L with respect to W_{ij} is given by

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{N} \sum_{k=1}^{n} \frac{\partial L}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial W_{ij}}. \tag{17}$$

From the previous exercise we showed that the partial derivative of L with respect to W_{ij} is $\frac{\partial L}{\partial z_i} x_j$ for a given $\mathbf{z}^{(i)}$. We will now sum that for the N elements in $\mathbf{Z}$.

$$\frac{\partial L}{\partial W_{ij}} = \sum_{l=1}^{N} \frac{\partial L}{\partial z_i^{(l)}} x_j^{(l)} = \begin{bmatrix} \frac{\partial L}{\partial z_i^{(1)}} & \frac{\partial L}{\partial z_i^{(2)}} & \cdots & \frac{\partial L}{\partial z_i^{(N)}} \end{bmatrix} \begin{bmatrix} x_j^{(1)} \\ x_j^{(2)} \\ \vdots \\ x_j^{(N)} \end{bmatrix} = \frac{\partial L}{\partial \mathbf{Z}_i} \mathbf{X}_j^\top \tag{18}$$

We can now express $\frac{\partial L}{\partial \mathbf{W}}$ using the expression for $\frac{\partial L}{\partial W_{ij}}$.

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{bmatrix} \frac{\partial L}{\partial W_{11}} & \cdots & \frac{\partial L}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial W_{n1}} & \cdots & \frac{\partial L}{\partial W_{nm}} \end{bmatrix} = \begin{bmatrix} \frac{\partial L}{\partial \mathbf{Z}_1} \mathbf{X}_1^\top & \cdots & \frac{\partial L}{\partial \mathbf{Z}_1} \mathbf{X}_m^\top \\ \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial \mathbf{Z}_n} \mathbf{X}_1^\top & \cdots & \frac{\partial L}{\partial \mathbf{Z}_n} \mathbf{X}_m^\top \end{bmatrix} = \tag{19}$$

$$= \begin{bmatrix} \frac{\partial L}{\partial \mathbf{z}^{(1)}} & \cdots & \frac{\partial L}{\partial \mathbf{z}^{(N)}} \end{bmatrix} \begin{bmatrix} \mathbf{x}^{(1)^\top} \\ \mathbf{x}^{(2)^\top} \\ \vdots \\ \mathbf{x}^{(N)^\top} \end{bmatrix} = \frac{\partial L}{\partial \mathbf{Z}} \mathbf{X}^\top \tag{20}$$

Lastly we want to derive an expression for $\frac{\partial L}{\partial \mathbf{b}}$. The partial derivative of L with respect to one bias b_i is given by

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{N} \sum_{j=1}^{n} \frac{\partial L}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_i}. \tag{21}$$

We again use a result from the previous exercise where we showed that the partial derivative of L with respect to b_i is equal to $\frac{\partial L}{\partial z_i}$ for one sample. We insert this to the sum above to get

$$\frac{\partial L}{\partial b_i} = \sum_{l=1}^{N} \frac{\partial L}{\partial z_i^{(l)}} = \frac{\partial L}{\partial \mathbf{Z}_i} = \begin{bmatrix} \frac{\partial L}{\partial z_i^{(1)}} & \frac{\partial L}{\partial z_i^{(2)}} & \cdots & \frac{\partial L}{\partial z_i^{(N)}} \end{bmatrix} \begin{bmatrix} 1^{(1)} \\ 1^{(2)} \\ \vdots \\ 1^{(N)} \end{bmatrix}. \tag{22}$$

Extending this to all biases we get the expression

$$\frac{\partial L}{\partial \mathbf{b}} = \begin{bmatrix} \frac{\partial L}{\partial \mathbf{z}^{(1)}} & \cdots & \frac{\partial L}{\partial \mathbf{z}^{(N)}} \end{bmatrix} \begin{bmatrix} 1^{(1)} \\ 1^{(2)} \\ \vdots \\ 1^{(N)} \end{bmatrix} = \frac{\partial L}{\partial \mathbf{Z}} \begin{bmatrix} 1^{(1)} \\ 1^{(2)} \\ \vdots \\ 1^{(N)} \end{bmatrix} \tag{23}$$

The listings below are the relevant code implemented in MATLAB.

Listing 1: Fully connected forward MATLAB implementation

```
Z = W*X + b;
```

Listing 2: Fully connected backward MATLAB implementation

```
onevec = ones(batch,1);
dldX = W'*dldZ;
dldW = dldZ*X';
dldb = dldZ*onevec;

dldX = reshape(dldX, sz);
```

Exercise 3

The rectifier linear unit (ReLU) is a common activation function that either returns the input x_i or 0 depending on if the input is positive or negative. In a more mathematical formulation the output z_i is equal to $\max(x_i, 0)$ given the input x_i. We want to derive an expression to $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial z_i}$. We begin by using the chain rule to get the equality

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z_i} \frac{\partial z_i}{\partial x_i}. \tag{24}$$

Because the ReLU activation function is not differentiable we split the derivation into two cases, one where x_i is positive and one where the input is negative. When x_i is positive the partial derivative $\frac{\partial z_i}{\partial x_i}$ is simple 1 due to the fact that the activation function is $z_i(x_i) = x_i$. The other case gives us that the activation function is $z_i(x_i) = 0$ which will result in that the derivative is equal to 0. The partial derivative of L with respect to x_i can then be expressed as

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z_i} \frac{\partial z_i}{\partial x_i} = \begin{cases} \frac{\partial L}{\partial z_i}, & x_i > 0 \\ 0, & x_i \leq 0 \end{cases} \tag{25}$$

The listings below are the relevant code implemented in MATLAB.

Listing 3: ReLU forward MATLAB implementation

```
X(X<0) = 0;
Z = X;
```

Listing 4: ReLU backward MATLAB implementation

```
X(X>=0) = 1;
X(X<0) = 0;
dldX = dldZ.*X;
```

Exercise 4

The softmax function is well known method to map the inputs (in most cases this is at the end of the network so inputs may not be the best word but we stick to it) so they can be interpreted as probabilities. Suppose that we have an input vector $\mathbf{x}$ with m elements representing the m classes. To calculate the probability of class i we use the softmax function

$$z_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}. \tag{26}$$

Because z_i is now probabilities we can use the negative log likelihood as the loss function we want to minimize. Suppose c is the true class of the sample, then the loss function is

$$L(\mathbf{x}, c) = -\log(z_c) = -x_c + \log\left(\sum_{j=1}^{m} e^{x_j}\right). \tag{27}$$

We now want to derive an expression of $\frac{\partial L}{\partial x_i}$ in terms of z_i. We start again by using the chain rule to get the equality

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z_c}\frac{\partial z_c}{\partial x_i}. \tag{28}$$

The first thing we do is to calculate $\frac{\partial L}{\partial z_c}$ from equation 27 which is simply equal to $-\frac{1}{z_c}$. We know want to calculate $\frac{\partial z_c}{\partial x_i}$. Here we have two cases, one where i is the true class c and the other where i is not the true class. We begin with the first case and set $x_i = x_c$.

$$\frac{\partial z_i}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}\right) = \frac{e^{x_i}\sum_{j=1}^{m} e^{x_j} - e^{x_i}e^{x_i}}{(\sum_{j=1}^{m} e^{x_j})^2} = \frac{e^{x_i}(\sum_{j=1}^{m} e^{x_j} - e^{x_i})}{(\sum_{j=1}^{m} e^{x_j})^2} = \tag{29}$$

$$= \left[z_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}\right] = z_i\frac{\sum_{j=1}^{m} e^{x_j} - e^{x_i}}{\sum_{j=1}^{m} e^{x_j}} = z_i(1 - z_i) \tag{30}$$

Notice that when $x_i = x_c$ then $z_i = z_c$. By multiplying the result above with $-\frac{1}{z_c}$ we get the partial derivative of L with respect to x_i.

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z_c}\frac{\partial z_c}{\partial x_i} = -\frac{1}{z_i}z_i(1 - z_i) = z_i - 1 \tag{31}$$

The second case is when i is not the same class as c.

$$\frac{\partial z_c}{\partial x_i} = \frac{\partial}{\partial x_i}\left(\frac{e^{x_c}}{\sum_{j=1}^{m} e^{x_j}}\right) = -\frac{e^{x_c}e^{x_i}}{(\sum_{j=1}^{m} e^{x_j})^2} = \left[z_i = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}\right] = -z_c z_i \tag{32}$$

Again by multiplying the result above with $-\frac{1}{z_c}$ we get

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z_c}\frac{\partial z_c}{\partial x_i} = (-\frac{1}{z_i})(-z_c z_i) = z_i \tag{33}$$

The expression becomes then

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial z_c}\frac{\partial z_c}{\partial x_i} = \begin{cases} z_i - 1, & i = c \\ z_i, & i \neq c \end{cases} \tag{34}$$

The listings below are the relevant code implemented in MATLAB.

Listing 5: Softmaxloss forward MATLAB implementation

```
idxc = sub2ind(size(x'), (1:numel(labels)).',labels);
xvec = reshape(x', [], 1);
xc = xvec(idxc);
L = (1/batch)*sum(log(sum(exp(x)))-xc');
```

Listing 6: Softmaxloss backward MATLAB implementation

```
idxc = sub2ind(size(x'), (1:numel(labels)).',labels);
z = exp(x)./sum(exp(x));
zvec = reshape(z', [], 1);

dldxvec = zvec;
dldxvec(idxc) = dldxvec(idxc)-1;
dldx = reshape(dldxvec, [batch, features])';
dldx = dldx/batch;
```

Exercise 5

The implementation of the gradient descent with momentum we used in our MATLAB code can be seen in the listing below.

Listing 7: hi

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

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

Exercise 6

The MNIST data set is a collection of labeled handwritten images of digits. We trained a convolutional neural network on these images and achieved a accuracy of 98.07% on the test set (i.e. images that the network had not trained on or seen before). The network had nine layers including the activation functions and maxpooling. The first layer was a placeholder for the inputs. The second layer was a convolutional layer with 16 channels of the size 5x5 and 16 biases, giving us a total of $5 \times 5 \times 1 \times 16 + 16 = 416$ trainable parameters. We then have a ReLU activation function layer followed by a maxpooling layer. The next layer is again a convolutional layer with 16 channels of size 5x5 giving us $5 \times 5 \times 16 \times 16 + 16 = 6416$ trainable parameters. After yet additional ReLU and maxpooling layers we have a fully connected layer with 784 inputs from the previous layer and 10 outputs. This gives us a total of $784 \times 10 + 10 = 7850$ trainable parameters. The last layer is a softmaxloss layer and does not have any trainable parameters. This means that we have a total of $416 + 6416 + 7850 = 14682$ trainable parameters with this architecture.

Figure 2 is the 16 filters after training and as we can see it seems that the filters focus on different areas of the digits. For example filter 11 focus on the bottom area while filter 14 focus on a diagonal area. The idea here is that the filters in combination with each other will detect characteristics of the digits.

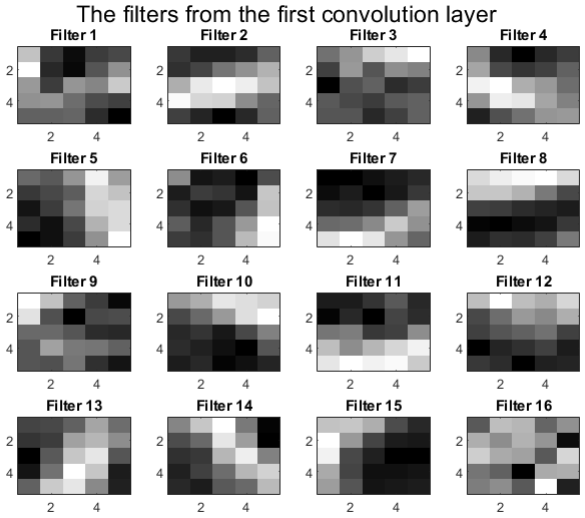

Figure 2: The 16 filters from the first convolutional layer after training of the MNIST data.

In figure 3 we can see six images that was incorrectly classified. The misclassification may happen when the number is written loosely or badly but also if the digit resembles a different digit too much. For example the nine that was predicted as a seven was probably loosely written and therefore hard to detect and the three that was predicted as a seven would probably be hard to classify for a human as well. Then there is images as the nine that was predicted as a five which we human clearly sees is a nine but the model gave a higher probability to be a five instead. This could be because the digit "contains" a five and if the upper right vertical line would not be connected humans would probably also classify the image as a five.

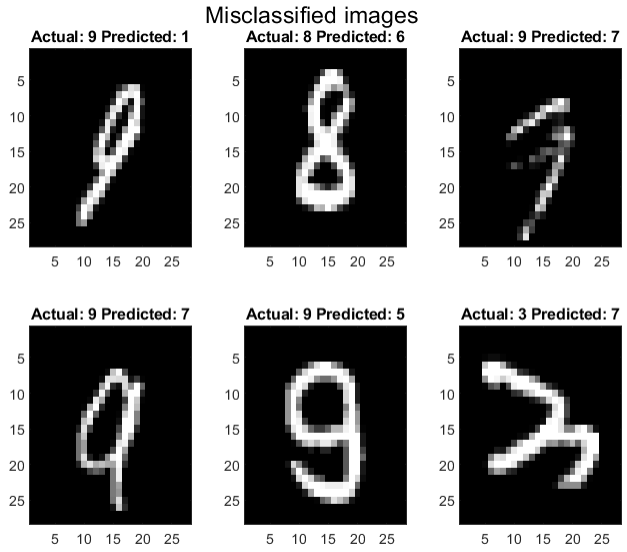

Figure 3: Six images that was misclassified by our model.

Figure 4 illustrates the confusion matrix for the predictions on the test set. Because the model had an high accuracy the matrix is mostly diagonal but we can make some interesting observations. The most common mistake is images of the digit 2 that get predicted to be the digit 8. This may be because the digit 2 can be written with a curl in the end which may make it resemble the digit 8 more. The second most common

mistake is images of the digit 5 that is predicted to be the digit 6 instead. This is understandable because the two digits looks like each other but it is interesting hat the reverse is not a common mistake. This can be because the model may put a lot of effort into the small line that normally separate the digits making it biased to the digit 6.

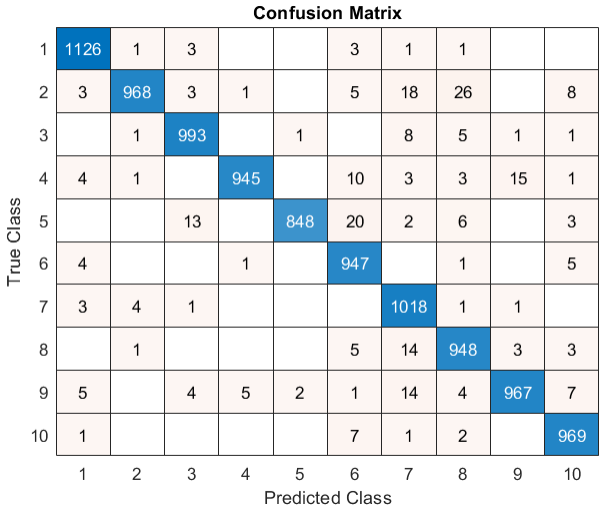

Figure 4: The confusion matrix for the predictions on the test set.

In table 1 we have the calculated precision and recall values for the ten classes. Precision is calculated with the formula

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} \tag{35}$$

and recall with the formula

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}. \tag{36}$$

A high precision value means that the result is relevant while a high recall value tells us that the relevant result was correctly classified. In the table we can see that the digit 8 have the lowest precision score while the digit 2 have the lowest recall score, nevertheless all classes achieves acceptable scores.

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10
Precision	0.9808	0.9774	0.9890	0.9749	0.9887	0.9863	0.9767	0.9625	0.9898	0.9692
Recall	0.9894	0.9622	0.9812	0.9878	0.9809	0.9791	0.9805	0.9743	0.9643	0.9939

Table 1: Precision and recall for the ten classes.

Exercise 7

The CIFAR10 data set is a collection of colored images in 10 different classes. The 10 classes represent images containing airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships or trucks in that order. We used this data set to train a convolutional neural network and test different architectures with the aim to get as high accuracy as possible on the test set. The original model had 8 layers including the activation functions and maxpooling. After the input layer the model had a convolutional layer with 16 filters with size 5x5x3 (the third dimension is because the images is colored). The model then had a ReLU activation layer followed by a maxpooling layer. The next layer was again a convolutional layer with 16 filters with the same size as before and again followed by a ReLU layer and maxpooling layer. After the maxpooling the model had a fully connected neural network with 4096 inputs from the previous layer and 10 outputs that went into the final layer, the softmaxloss layer. This model achieved an accuracy of 45.7%.

The first thing we wanted to implement to improve the model was a way to warm start the training by using the previously trained weights. The idea here was that this would make training faster and we could therefore test more models. We experimented to do this by saving the trained model and using it again with some modifications. The problem we had however was that the connections between the layers had to match which we did not find a way to solve. We therefore proceeded by testing different models with a cold start.

The first model we tested had an extra convolutional layer after the first one with the same number of filters with the same size. This model improved the accuracy. We then proceeded to increase the number of filters each convolutional layer had, the first had 64 filters, the second 32 filters and the last 16 filters. This model barely improved the accuracy. We also tested to have less filters in every convolutional layer but this model decreased the accuracy. The final model had again three convolutional layers with 16, 32 and 48 filters in that orders. This model also had two fully connected layers, the first one had 768 inputs and 20 outputs while the second had 10 outputs. This model achieved an accuracy of 55.3% on the test set.

With 16 filters each with size 5x5x3 the number of trainable parameters in the first convolutional layer will be $5 \times 5 \times 3 \times 16 + 16 = 1216$ with the biases. The ReLU and maxpooling does not have any trainable parameters. The second convolutional layer has $5 \times 5 \times 16 \times 32 + 32 = 12832$ trainable parameters. The last convolutional layer has $5 \times 5 \times 32 \times 48 + 48 = 38448$ parameters. After the ReLU and maxpooling layers we have 768 inputs that goes into the fully connected layer with 20 outputs, which gives us $768 \times 20 + 20 = 15380$ parameters. Lastly we have yet another fully connected layer with 10 outputs, giving us $20 \times 10 + 10 = 210$ additional parameters. This means that we have a total of $1216 + 12832 + 38448 + 15380 + 210 = 68086$ parameters the model has to fit.

In figure 5 we have plotted the 16 filters in the first convolutional layer. Notice that the original images are colored and therefore have three values for each pixel (red, green and blue). The plotted images in the figure are therefore made by the three colors. We may see that some filters tend to focus on one color more then the other but it is hard to interpret why the filters are trained like this. It may be because filter 8 want to detect grass while filter 7 wants to detect water but we can never know. A good allegory is the black box model where we insert something and the black box returns something else. We do not know what the box is doing but if the result is good we accept the method. Similarly in our model we insert the images and the model uses those images to return a prediction but we do not know why this prediction was made.

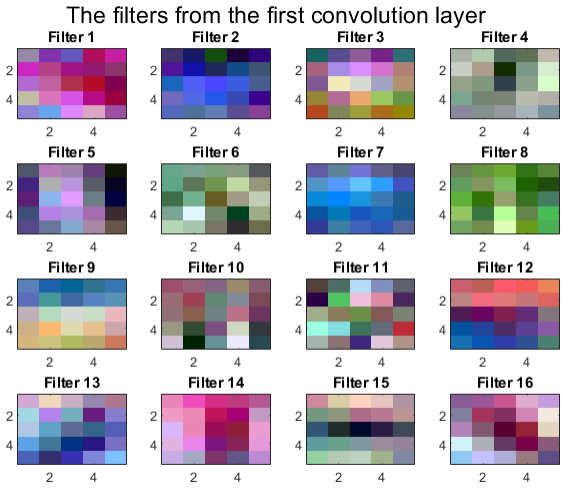

Figure 5: The 16 filters from the first convolutional layer after training with the CIFAR10 data set. The filters is plotted as RGB images.

In figure 6 we can see six images that was incorrectly classified. Class 10 is images of trucks while class 2 is

images of cars which makes it understandable why the middle image on the bottom row was misclassified. Notice that the right image on the top row is the only image of an animal that is misclassified as an vehicle.

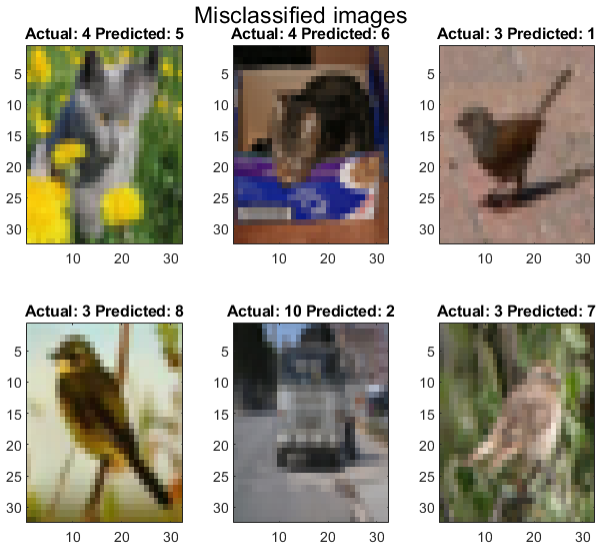

Figure 6: Six images that was misclassified by our final model.

Below in figure 7 the confusion matrix for the predictions on the test set is plotted. Notice how images of trucks is mostly misclassified as cars while images of birds is incorrectly classified as deer. Other common misclassification are images of dogs that are predicted to be cats and images of frogs that are predicted to be deer. Class 1, 2, 9 and 10 are all images of vehicle and as we can see they are mostly misclassified as another vehicle. Interesting to see is that birds are commonly classified as airplanes but the reverse is not true.

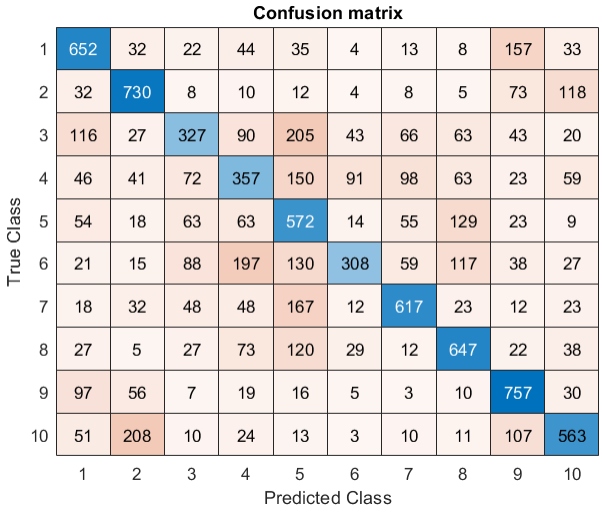

Figure 7: The confusion matrix for the predictions on the test set.

Table 2 the calculated precision and recall values for the ten classes is shown. We see that the cats class have the lowest precision while the dogs class has the lowest recall.

	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7	Class 8	Class 9	Class 10
Precision	0.5853	0.6271	0.4866	0.3859	0.4028	0.6004	0.6557	0.6013	0.6032	0.6120
Recall	0.6520	0.7300	0.3270	0.3570	0.5720	0.3080	0.6170	0.6470	0.7570	0.5630

Table 2: Precision and recall for the ten classes in the CIFAR10 data set.