Image Processing - Assignment 2

Group 3
Martin Eek Gerhardsen

October 25

Department of Engineering Cybernetics

Contents

1 Convolutional Neural Networks

1.1 Task 1: Theory

1.2 Task 2: Programming

2 Filtering in the Frequency Domain

2.1 Task 3: Theory

a)

The binary operation XOR, or exclusive or, cannot be represented by a single-layer neural network. If we look at the two binary input values as the two digits of a binary number, we get the range $[0, 3]$. Then applying the XOR function for this range, we get $[0, 1, 1, 0]$, which obviously cannot be represented by a linear function.

b)

A hyperparameter is a parameter which is set before, and not changed by, the learning process. Batch size and learning rate are examples of hyperparameters.

c)

The softmax function is a smooth approximation of the arg max function. The values being applied to the are shifted into the range $[0, 1]$, so that they can be interpreted as probabilities, and further as a probability density function, where the sum of the result is 1.

d)

$$C(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2, \hat{y}_n = 1 \tag{1}$$

Evaluating the network, we may conclude that $c_1 = 2$ and $c_2 = -4$, and therefore $y = c_1$.

Using eq. (1), we calculate:

$$\begin{aligned}
w_1' &= \frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_1}\frac{\partial c_1}{\partial a_1}\frac{\partial a_1}{\partial w_1} = 2 * (y_n - \hat{y}_n) * 1 * 1 * w_1 = -2 * (y_n - 1) \\
w_2' &= \frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_1}\frac{\partial c_1}{\partial a_2}\frac{\partial a_2}{\partial w_2} = 2 * (y_n - \hat{y}_n) * 1 * 1 * w_2 = 2 * (y_n - 1) \\
w_3' &= \frac{\partial C}{\partial w_3} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_2}\frac{\partial c_2}{\partial a_3}\frac{\partial a_3}{\partial w_3} = 2 * (y_n - \hat{y}_n) * 0 * 1 * w_3 = 0 \\
w_4' &= \frac{\partial C}{\partial w_4} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_2}\frac{\partial c_2}{\partial a_4}\frac{\partial a_4}{\partial w_4} = 2 * (y_n - \hat{y}_n) * 0 * 1 * w_4 = 0 \\
b_1' &= \frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_1}\frac{\partial c_1}{\partial b_1} = 2 * (y_n - \hat{y}_n) * 1 * 1 = 2 * (y_n - 1) \\
b_2' &= \frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial c_2}\frac{\partial c_2}{\partial b_2} = 2 * (y_n - \hat{y}_n) * 0 * 1 = 0
\end{aligned}$$

e)

Calculating the current value for y:

$$\begin{aligned} &a_1 = 1, \quad &&a_2 = 0, \quad &&a_3 = 1, \quad &&a_4 = -4 \\ &c_1 = 2, \quad &&c_2 = -4, \quad &&y_n = max(c_1, c_2) = c_1 = 2 \end{aligned}$$

$$\theta_{t+1} = w_t - \alpha \frac{\partial C}{\partial \theta_t} \tag{2}$$

Using eq. (2) and $\alpha = 0.1$, we get:

$$\begin{aligned} w_{1,t+1} &= w_{1,t} - \alpha \frac{\partial C}{\partial w_1} = -1 - 0.1 * (-2 * (2 - 1)) = -0.8 \\ w_{3,t+1} &= w_{3,t} - \alpha \frac{\partial C}{\partial w_3} = -1 - 0.1 * 0 = -1 \\ b_{1,t+1} &= b_{1,t} - \alpha \frac{\partial C}{\partial b_1} = 1 - 0.1 * 2 * (2 - 1) = 0.8 \end{aligned}$$

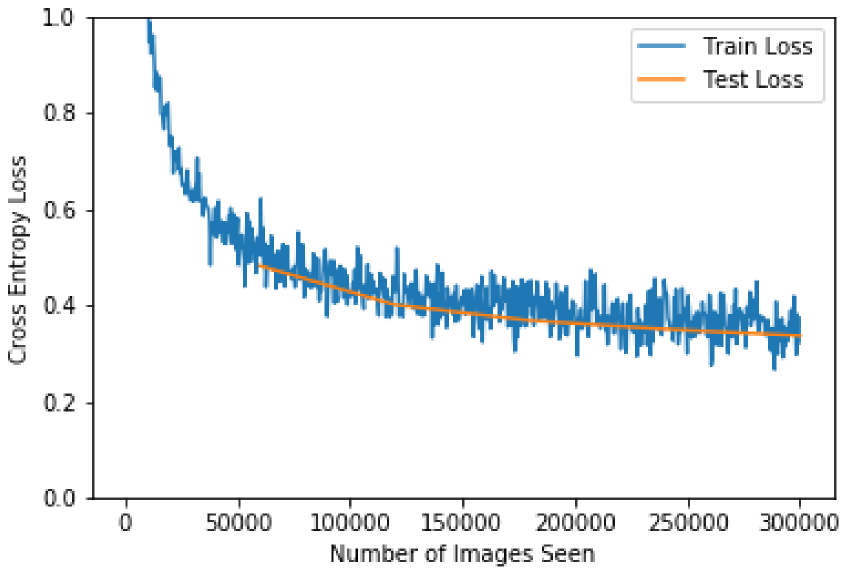

Figure 1: Training and test loss before the task

2.2 Task 4: Programming

a)

The original training loss can be seen fig. 1 and the normalized here: fig. 2. We may note that the loss converges quicker, and to a smaller value than the non-normalized data. This is as expected.

We may note that it seems like certain of the validation images were too different to the training images to be classified, which manifests as a spike in the test-loss of the normalized figures. I think this is the reason, but I'm not quiet sure, and would like some pointers if there are any specifics. It is interesting that this problem only manifests when using the single-layered neural network with the normalized images. If the images were not normalized, or the neural network had a hidden layer, this problem disappeared.

b)

For each class, the gray-scale images were generated by normalizing the weights for each class and reshaping the weights inverse of the way we convert an image to the input layer. We can see the weight image in the figures: fig. 3, fig. 4, fig. 5, fig. 6, fig. 7, fig. 8, fig. 9, fig. 10, fig. 11 and fig. 12.

This weights give what points of the image should increase the probability of that specific class being the correct one. As we only have one layer, this is a mapping of the input image to the output, and we can see the shapes

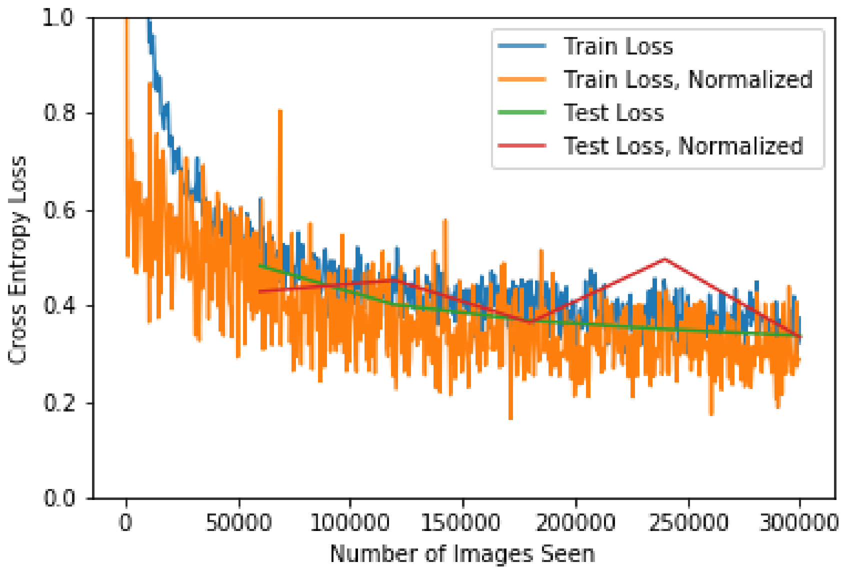

Figure 2: Original vs. normalized training and test loss

that are *extracted*, or emphasized, by the weights, and we can note that they are similar to the number we are looking for. This can be especially well from fig. 3, with the large negative space in the center being important to determining if a number is zero or not.

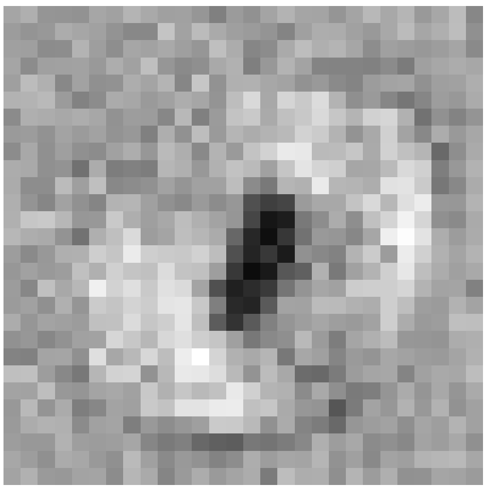
Figure 3: Class weight 0

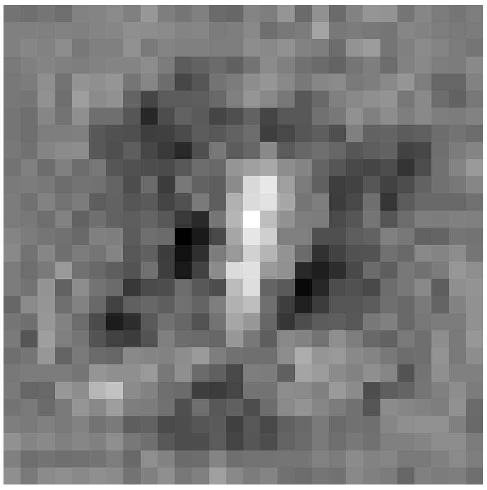
Figure 4: Class weight 1

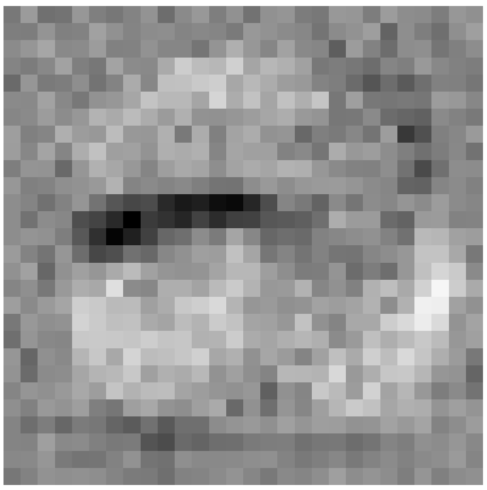

Figure 5: Class weight 2

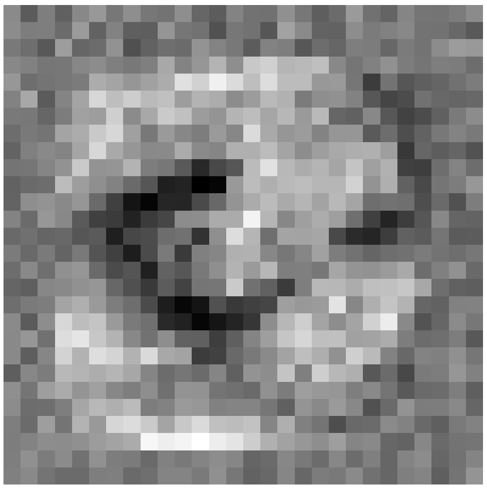

Figure 6: Class weight 3

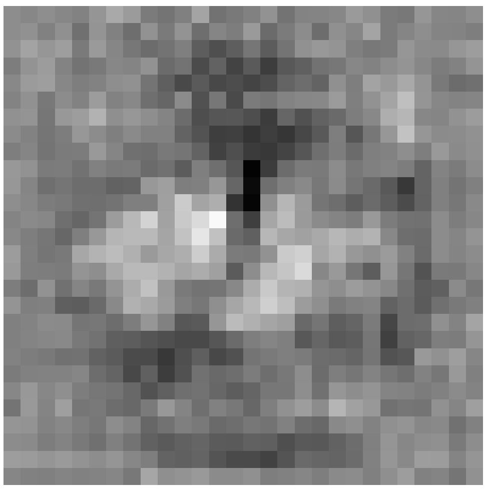

Figure 7: Class weight 4

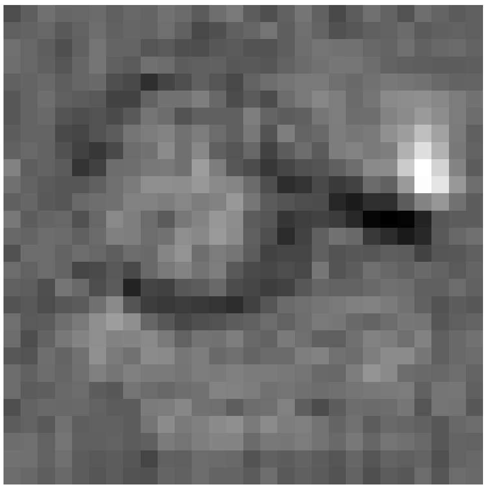

Figure 8: Class weight 5

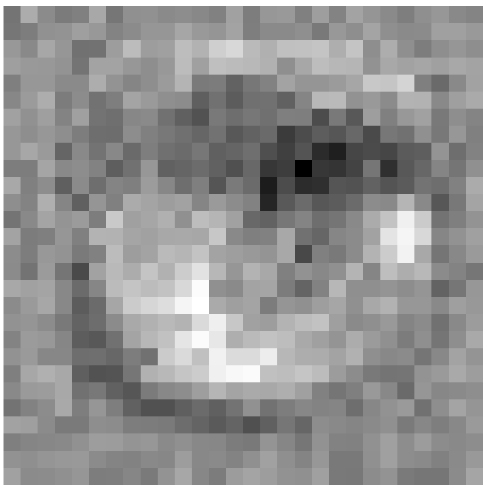

Figure 9: Class weight 6

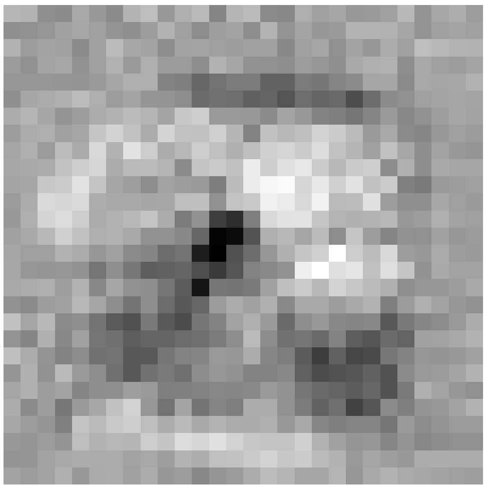

Figure 10: Class weight 7

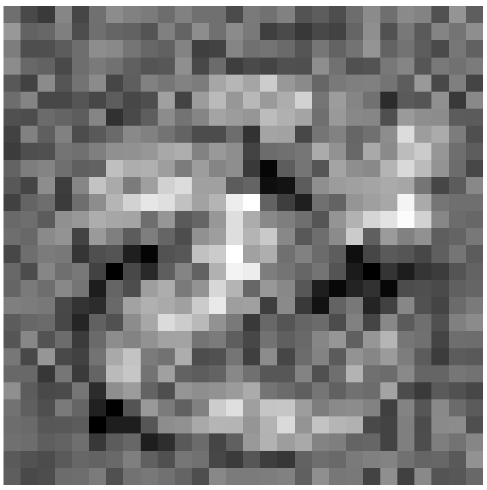
Figure 11: Class weight 8

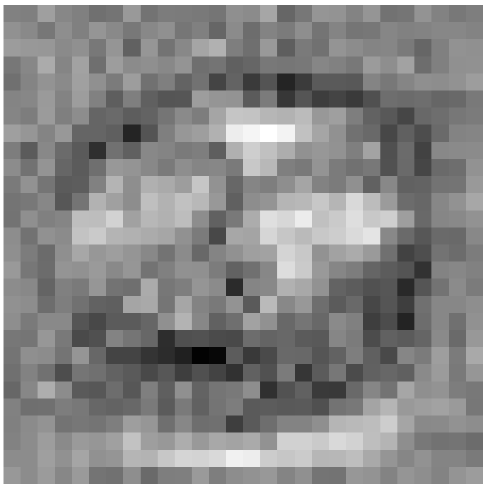
Figure 12: Class weight 9

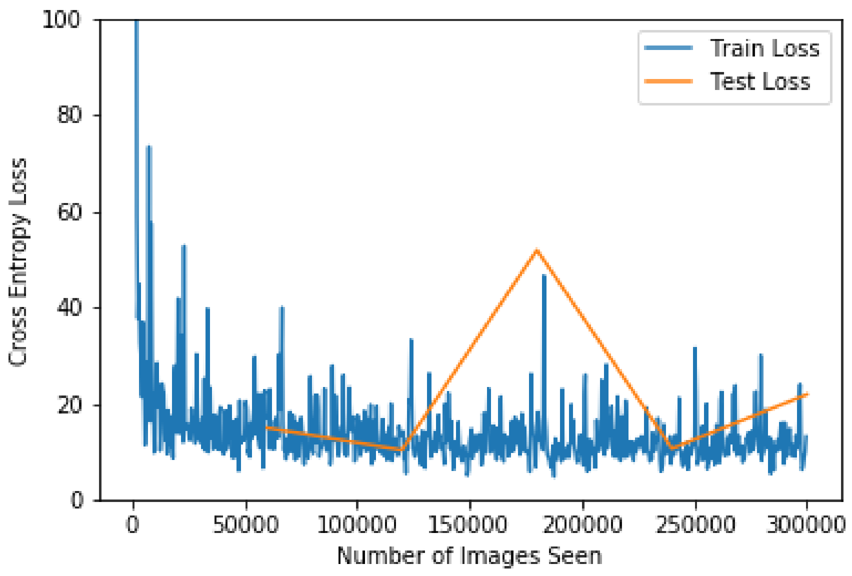

Figure 13: Training and test loss with learning rate 1

c)

The accuracy is now way worse, as could be expected. This is due to the large learning rate. Even though a larger learning rate may give faster learning, it is also makes the gradient descent be less accurate, and we will not reach the correct weights to minimize the cost function. See fig. 13 for the cross entropy loss, but note that the plot does initially start even higher on the y-axis, but limiting it to 100 was done to show the shape of the function better.

d)

From fig. 14 we can see that the training loss and test loss end up converging faster to a lower cross entropy loss than the original neural network of fig. 1 and fig. 2. This means that using the hidden layer both helps the neural network learn faster (using the same hyperparameters), as well as converge to a lower value, and the network is therefore both faster (to teach) and better (more likely to get the answer correct) than the original network.

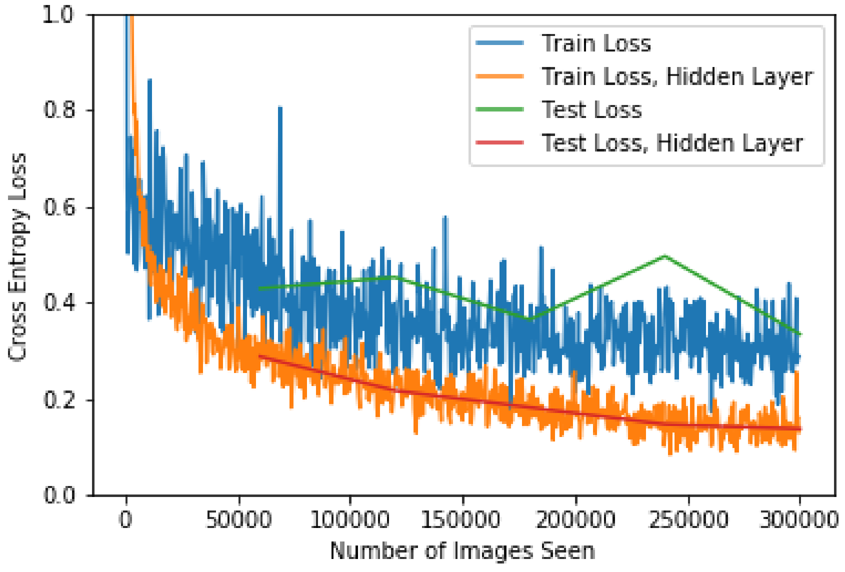

Figure 14: Training and test loss with new hidden layer