Problem 3.1

Surface plot

The error surface is shown in Figure 1. The loss is computed as the sum of the individual losses over all samples.

$$J\left(\mathbf{w}, \mathbf{x}^{(i)}, y^{(i)}\right) = \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2 \tag{1}$$

This can be written as a quadratic form

$$\sum_{i=0}^{N-1} \mathbf{w}^T \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^T \mathbf{w} - 2y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} + \left(y^{(i)}\right)^2 = \mathbf{w}^T \mathbf{A} \mathbf{w} + \mathbf{b}^T \mathbf{w} + c. \tag{2}$$

By evaluating

$$\mathbf{A} = \sum_{i=0}^{N-1} \mathbf{x}^{(i)} \left(\mathbf{x}^{(i)}\right)^T = \begin{bmatrix} 120.4 & 6.25 \\ 6.25 & 100.1 \end{bmatrix} \tag{3}$$

it can be seen that this matrix is positive definite and the loss is therefore convex in the argument $\mathbf{w}$.

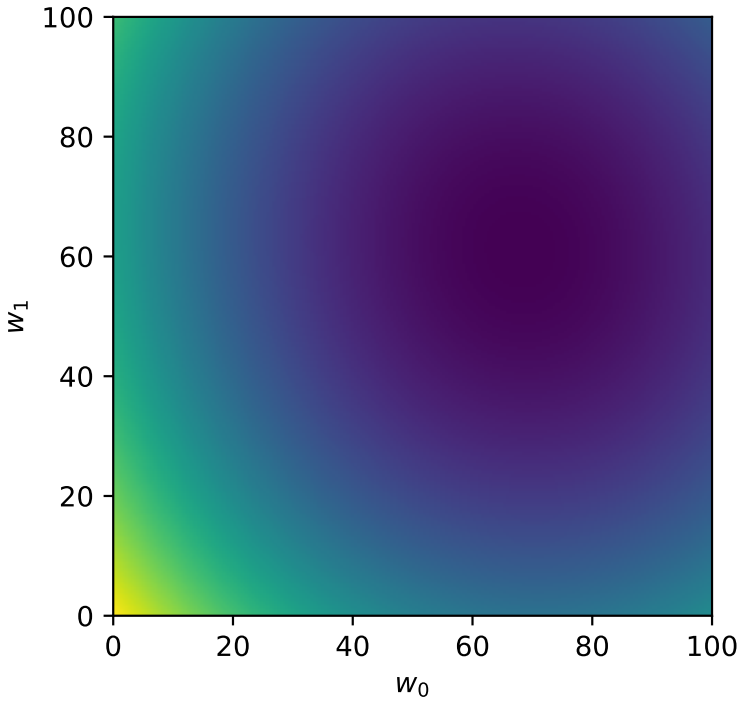


Figure 1: Error surface

Analytical gradient

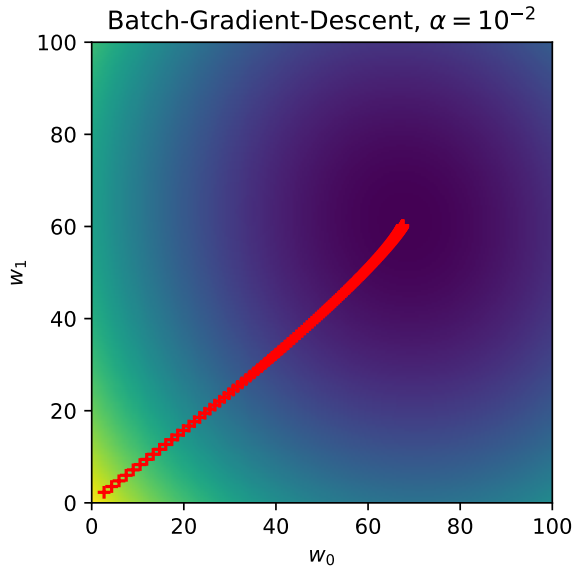
The loss for a single sample by (1).

The gradient with respect to the weight vector is therefore

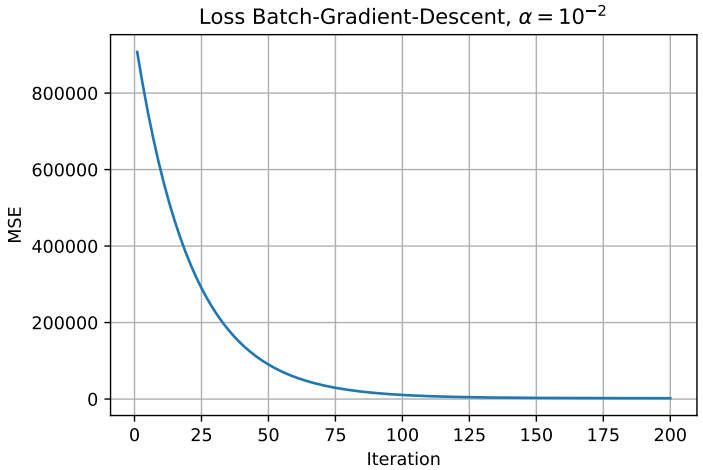
$$\nabla_{\mathbf{w}} J\left(\mathbf{w}, \mathbf{x}^{(i)}, y^{(i)}\right) = 2\left(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)}\right)\left(-\mathbf{x}^{(i)}\right) = 2\mathbf{x}^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} - y^{(i)}\right). \tag{4}$$

Batch-Gradient-Descent

For the settings $K = 200$, $\alpha = 10^{-2}$ and the initial weight $\hat{\mathbf{w}}^{(0)} = [1, 1]^T$ the history of the loss and a scatter plot of the weights are shown in Figure 2. The final weight vector is $\hat{\mathbf{w}}^{(K)} = [67.53, 60.17]^T$



(a) Scatter plot of weight vectors

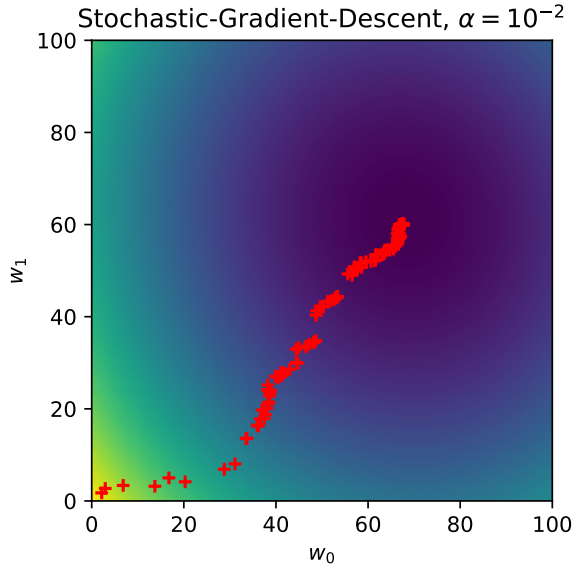


(b) Loss

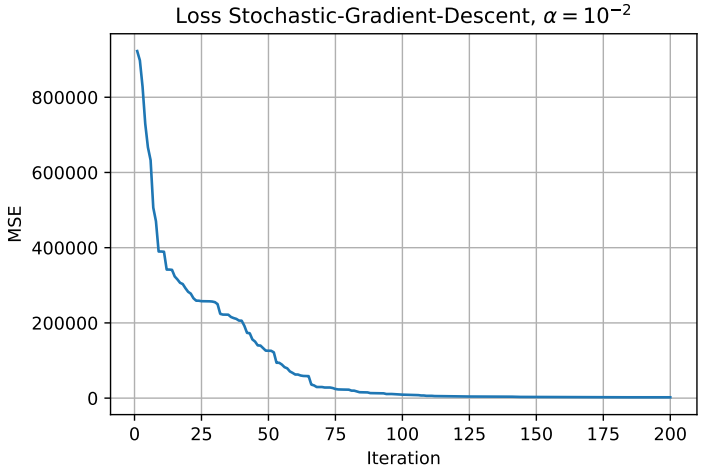
Figure 2: History of optimization with BGD

Stochastic-Gradient-Descent

For the settings $K = 200$, $\alpha = 10^{-2}$ and the initial weight $\hat{\mathbf{w}}^{(0)} = [1, 1]^T$ the history of the loss and a scatter plot of the weights are shown in Figure 3. The final weight vector is $\hat{\mathbf{w}}^{(K)} = [67.54, 60.17]^T$.



(a) Scatter plot of weight vectors



(b) Loss

Figure 3: History of optimization with SGD

Discussion SGD and BGD

BGD almost approaches the optimum in a straight line as can be seen from the scatter plot. For SGD the trajectory of the weight vector is more noisy, but it still approaches

the minimum of the loss. As SGD computes the gradient on a single sample, it is more noisy and therefore the direction of the gradient varies from iteration to iteration. For BGD the gradients of all samples are averaged and therefore less noisy which leads to a more steady path towards the optimum.

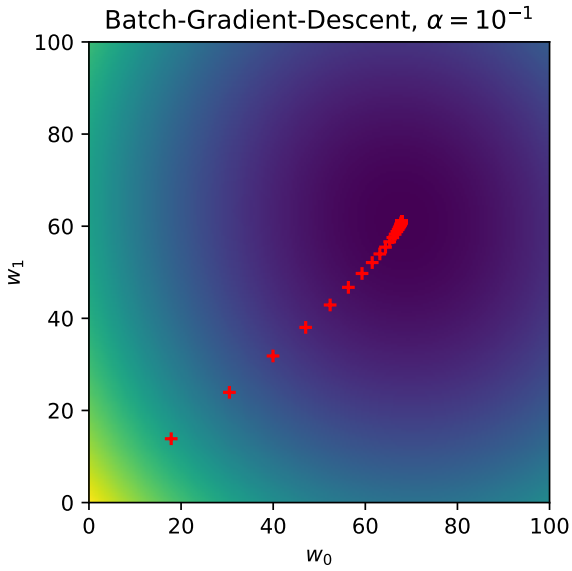
The same behaviour can be seen in the loss plots, where the loss decreases strictly monotonic for BGD, whereas for SGD the loss sometimes decreases rapidly and then it stays almost constant for some iterations.

Effect of learning rate

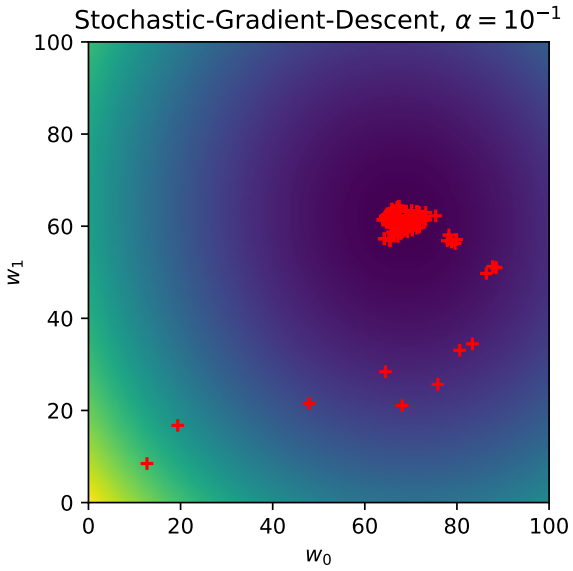
In Figure 4 the effect of the learning rate on both BGD and SGD is shown. Overall it can be seen, that the Loss decreases significantly faster with the higher learning rate. For BGD the trajectory of the weights is nearly unchanged, it just approaches the optimum faster. For SGD the weights show a significantly larger fluctuation close to the optimum and from the loss curve it can be seen by the bumps close to 0, that sometimes the loss even increases from one iteration to another.

For a learning rate of $\alpha = 1$ both solvers diverged, the loss increased and the weight vector does not approach the minimum of the loss.

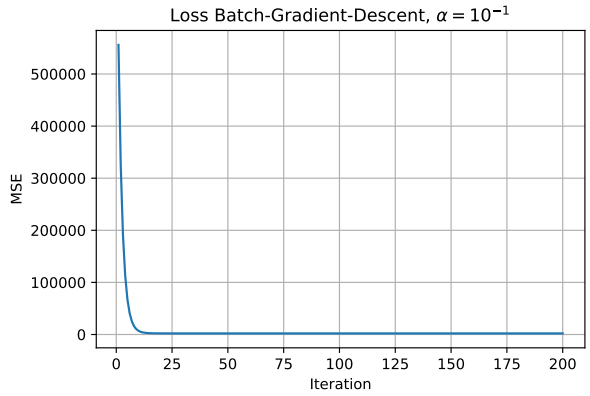
Overall a higher learning rate is beneficial because less iterations are needed to train the model, but it can't be set too high because this might lead to divergence or the weights walking around the optimum for a long time.



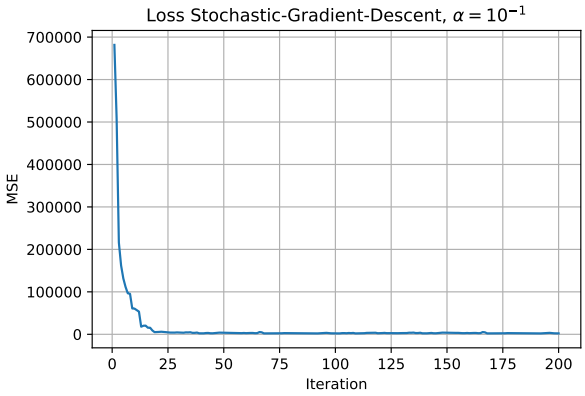
(a) Scatter plot of weight vectors for BGD



(b) Scatter plot of weight vectors for SGD



(c) Loss for BGD



(d) Loss for SGD

Figure 4: Comparison BGD and SGD for higher learning rate

Epochs

One full iteration over the traing dataset is called an epoch. For BGD one iteration of the optimizer is one iteration over the training set, therefore 200 epochs where performed in this example.

For SGD one iteration uses only one sample, as the dataset includes 100 samples, this means 2 epochs where performed.

For a large dataset I would choose SGD, because when using BGD the weights are only updated after going through all the training samples which would take up a lot of time. With SGD the weights are updated with each sample decreasing the computation time.

Problem 3.2

XOR Approximation

The configuration in Table 1 was found by selecting a linear activation function for the neuron in Layer 1. For both neurons in Layer 0 a relu activation function was choosen and the intermediate outputs $z_0^{(0)}$ and $z_1^{(0)}$ were selected as shown in the table.

From there on the weights and biases can be found to give the desired outputs

$$\mathbf{w}_0^{(0)} = [-1, +2] \quad b_0^{(0)} = -1 \quad \sigma_0^{(0)} = \text{relu} \tag{5}$$

$$\mathbf{w}_1^{(0)} = [+2, -1] \quad b_1^{(0)} = -1 \quad \sigma_1^{(0)} = \text{relu} \tag{6}$$

$$\mathbf{w}_0^{(1)} = [+1, +1] \quad b_0^{(1)} = 0 \quad \sigma_0^{(1)} = \text{linear} \tag{7}$$

$\mathbf{x}^T$	y	$a_0^{(0)}$	$z_0^{(0)}$	$a_1^{(0)}$	$z_1^{(0)}$	$a_0^{(1)}$	$\hat{y} = z_0^{(1)}$
$[0, 0]$	0	-1	0	-1	0	0	0
$[0, 1]$	1	1	1	-2	0	1	1
$[1, 0]$	1	-2	0	1	1	1	1
$[1, 1]$	0	0	0	0	0	0	0

Table 1: Network output and activations to approximate a logical XOR

Backpropagation

The loss function is $J(\mathbf{w}, b) = (y - \hat{y})^2$. The gradient with respect to the weights and bias of the first layer are

$$\frac{\partial J}{\partial \mathbf{w}_0^{(1)}} = \frac{\partial J}{\partial z_0^{(1)}} \frac{\partial z_0^{(1)}}{\partial a_0^{(1)}} \frac{\partial a_0^{(1)}}{\partial \mathbf{w}_0^{(1)}} \tag{8}$$

$$\frac{\partial J}{\partial b_0^{(1)}} = \frac{\partial J}{\partial z_0^{(1)}} \frac{\partial z_0^{(1)}}{\partial a_0^{(1)}} \frac{\partial a_0^{(1)}}{\partial b_0^{(1)}} \tag{9}$$

As $\hat{y} = z_0^{(1)}$, $z_j^{(i)} = \sigma_j^{(i)}\left(a_j^{(i)}\right)$ and $a_j^{(i)} = \mathbf{w}_j^{(i)} \mathbf{z}^{(i-1)} + b_j^{(i)}$ the terms can be calculated to

$$\frac{\partial J}{\partial z_0^{(1)}} = 2(\hat{y} - y) \tag{10}$$

$$\frac{\partial z_0^{(1)}}{\partial a_0^{(1)}} = \frac{\partial \sigma_0^{(1)}\left(a_0^{(1)}\right)}{\partial a_0^{(1)}} = 1 \tag{11}$$

$$\frac{\partial a_0^{(1)}}{\partial \mathbf{w}_0^{(1)}} = \left(\mathbf{z}^{(0)}\right)^T \tag{12}$$

$$\frac{\partial a_0^{(1)}}{\partial b_0^{(1)}} = 1 \tag{13}$$

where in (11) a linear activation function was assumed, this term would change for a nonlinear activation function. Therefore, the gradients for layer 1 are

$$\frac{\partial J}{\partial \mathbf{w}_0^{(1)}} = 2\left(\hat{y} - y\right)\left(\mathbf{z}^{(0)}\right)^T \tag{14}$$

$$\frac{\partial J}{\partial b_0^{(1)}} = 2\left(\hat{y} - y\right). \tag{15}$$

Analogously the gradients for neuron 0 of layer 0 are

$$\frac{\partial J}{\partial \mathbf{w}_0^{(0)}} = \frac{\partial J}{\partial z_0^{(1)}}\frac{\partial z_0^{(1)}}{\partial a_0^{(1)}}\frac{\partial a_0^{(1)}}{\partial z_0^{(0)}}\frac{\partial z_0^{(0)}}{\partial a_0^{(0)}}\frac{\partial a_0^{(0)}}{\partial \mathbf{w}_0^{(0)}} = 2\left(\hat{y} - y\right) w_{0,0}^{(0)} \mathbf{x}^T \tag{16}$$

$$\frac{\partial J}{\partial b_0^{(0)}} = \frac{\partial J}{\partial z_0^{(1)}}\frac{\partial z_0^{(1)}}{\partial a_0^{(1)}}\frac{\partial a_0^{(1)}}{\partial z_0^{(0)}}\frac{\partial z_0^{(0)}}{\partial a_0^{(0)}}\frac{\partial a_0^{(0)}}{\partial b_0^{(0)}} = 2\left(\hat{y} - y\right) w_{0,0}^{(0)} \tag{17}$$

During optimization the gradient flows backwards through the network. First the gradient of the loss function with respect to the output of the last neuron is calculated

$$\frac{\partial J}{\partial z_0^{(i)}}. \tag{18}$$

With the weights, biases and activation function of the neuron the gradient with respect to the input of that neuron can be calculated

$$\frac{\partial z_0^{(i)}}{\partial z_0^{(i-1)}} \tag{19}$$

which is then again passed backwards the the previous layer where this step is repeated until the input of the network is reached.

In this way the calculation of the gradients with respect to all trainable variables can be calculated efficiently, because redundant calculations are avoided. The gradient information from the preceding layer is used to compute the gradient of the current layer.

AND Approximation

A network with a linear activations in the first layer and a sigmoid in the second layer was trained to approximate a logical AND function. The network outputs are shown in Table 2. The weights and biases are summarized in (20) to (22).

$\mathbf{x}^T$	y	$a_0^{(0)}$	$z_0^{(0)}$	$a_1^{(0)}$	$z_1^{(0)}$	$a_0^{(1)}$	$\hat{y} = z_0^{(1)}$
$[0,0]$	0	-3.6	-3.6	-2.1	-2.1	-19	$4.2 \cdot 10^{-9}$
$[0,1]$	0	-0.9	-0.9	-0.6	-0.6	-6.5	$1.5 \cdot 10^{-3}$
$[1,0]$	0	-0.9	-0.9	-0.5	-0.5	-6.5	$1.4 \cdot 10^{-3}$
$[1,1]$	1	1.7	1.7	1.0	1.0	6.2	1.0

Table 2: Network output and activations to approximate a logical AND

$$\mathbf{w}_0^{(0)} = [2.6, 2.7] \quad b_0^{(0)} = -3.6 \quad \sigma_0^{(0)} = \text{linear} \tag{20}$$

$$\mathbf{w}_1^{(0)} = [1.6, 1.5] \quad b_1^{(0)} = -2.1 \quad \sigma_1^{(0)} = \text{linear} \tag{21}$$

$$\mathbf{w}_0^{(1)} = [3.6, 2.1] \quad b_0^{(1)} = -2.1 \quad \sigma_0^{(1)} = \text{sigmoid} \tag{22}$$

Problem 3.3

Number of trainable parameters

The summary of the baseline model can be seen below. For a dense layer with input size n and output size of m the number of parameters can be calculated by first considering the interconnections basically acting like a matrix of size $m \times n$. Additionally there are m bias terms adding up to a total number of $(n+1)m$ parameters. For `Layer1` this means $11 \cdot 128 = 1408$ parameters, because the input size is 10. For the `Output` layer it can be calculated by $129 \cdot 3 = 387$.

Listing 1: Summary of baseline model

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 128)	1408
Layer2 (Dense)	(None, 128)	16512
Layer3 (Dense)	(None, 128)	16512
Output (Dense)	(None, 3)	387

Total params: 34,819
Trainable params: 34,819
Non-trainable params: 0

The summary of the dropout model below shows that the dropout has exactly the same number of total parameters. The added dropout layers have no parameters, a certain percentage of the connections is just dropped at random during training.

Listing 2: Summary of dropout model

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 128)	1408

Drop1 (Dropout)	(None, 128)	0
Layer2 (Dense)	(None, 128)	16512
Drop2 (Dropout)	(None, 128)	0
Layer3 (Dense)	(None, 128)	16512
Drop3 (Dropout)	(None, 128)	0
Output (Dense)	(None, 3)	387

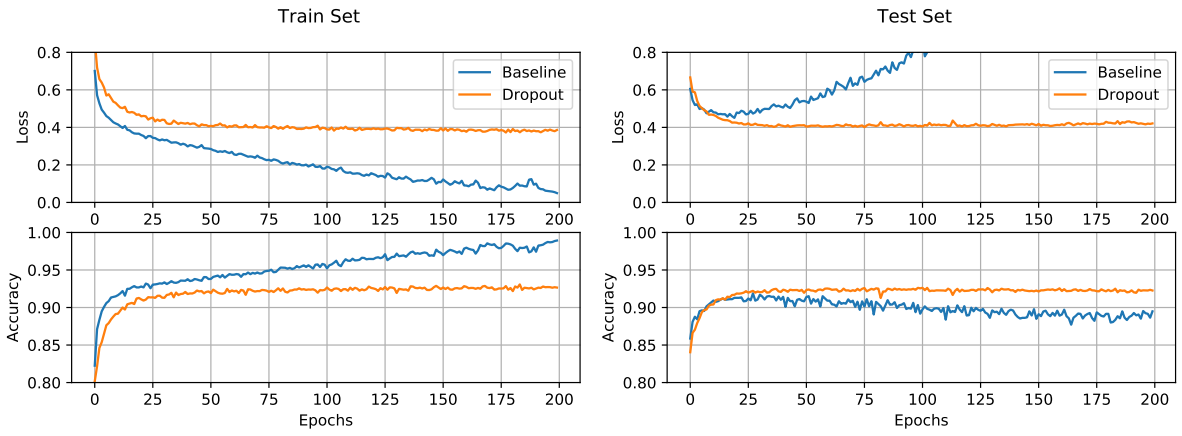
Total params: 34,819
Trainable params: 34,819
Non-trainable params: 0

Comparison of the models

In Figure 5 the learning curves for both models on both datasets are shown.

On the train dataset the baseline model can decrease the loss further achieving a better accuracy than the dropout model. The loss of the dropout model stays relatively constant from epoch 50 on. On the test set it can be seen that the baseline model is already in an overfitting regime from iteration 25 on, because the loss on the test set increases again and the accuracy gets worse. The dropout model does not suffer from this effect, its performance stays constant from epoch 50 on.

The final accuracy for the baseline model was 89.5 %, for the dropout model it was 92.3 % on the test set.



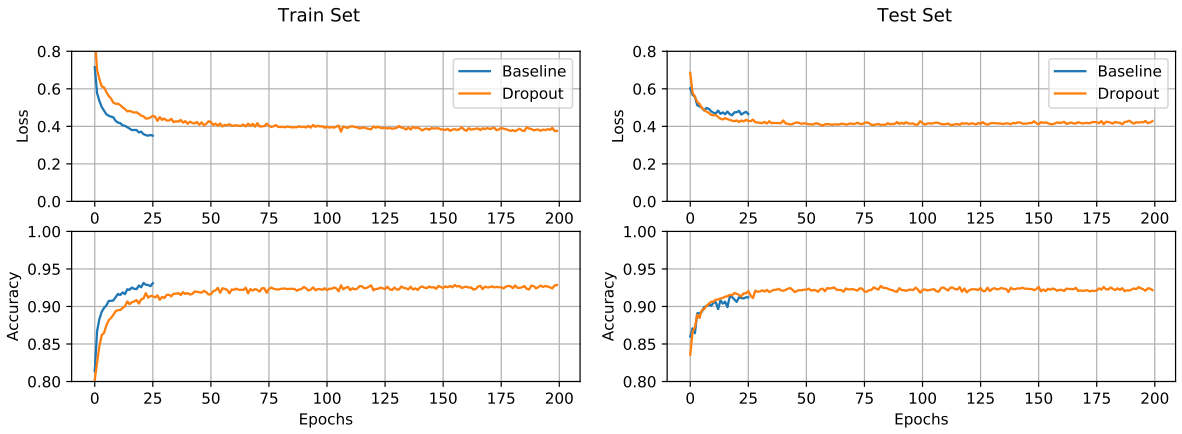
(a) Train dataset (b) Test dataset

Figure 5: Learning curves on the two datasets for both models

Early Stopping

Another approach to regularization and avoiding overfitting is early stopping. The corresponding plots are shown in Figure 6. The training process is stopped earlier, when for a certain number of epochs the loss decreases less than a specified amount. The training process is stopped after iteration 25. As can be seen this is before the loss on the test set starts increasing.

The final accuracy for the baseline model was 90.8 %, for the dropout model it was 92.3 % on the test set. The baseline model thus performs better with early stopping than when optimizing for the whole 200 epochs, because overfitting is avoided.



(a) Train dataset (b) Test dataset

Figure 6: Learning curves on the two datasets for both models

Housing dataset

A neural network was then used to perform regression on the housing dataset. As the loss the mean absolute error (MAE) was choosen. The network structure is summarized below. A graph is shown in Figure 7. For the hidden layers a relu activation function has been selected. For the output layer a linear layer was selected. For the dropout layers a rate of 0.15 was selected. The batch size was set to 64. The learning curve is shown in Figure 8. The final MAE on the test set was 0.312. In the shown number of epochs the loss on both datasets decreases, indicating that the model is not yet overfitting.

Listing 3: Neural network for the housing dataset

Layer (type)	Output Shape	Param #
Layer1 (Dense)	(None, 128)	1152
Drop1 (Dropout)	(None, 128)	0

Layer2 (Dense)	(None, 128)	16512
Drop2 (Dropout)	(None, 128)	0
Layer3 (Dense)	(None, 64)	8256
Drop3 (Dropout)	(None, 64)	0
Output (Dense)	(None, 1)	65

Total params: 25,985
Trainable params: 25,985
Non-trainable params: 0

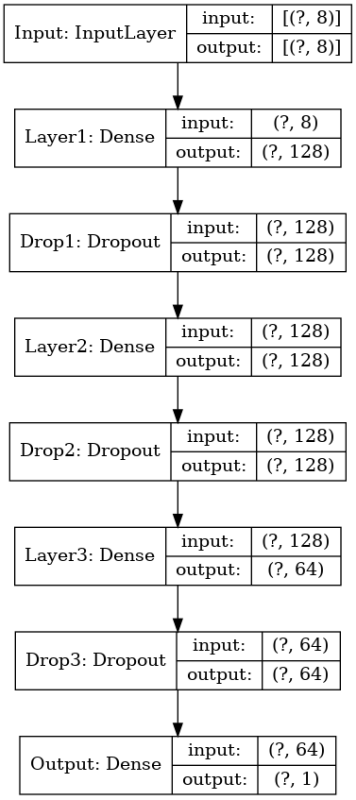


Figure 7: Graphical representation of the network

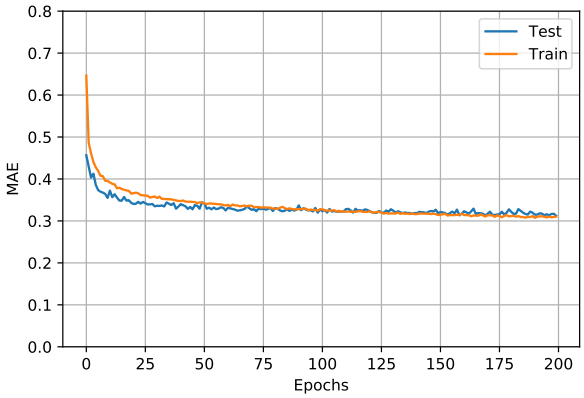


Figure 8: Learning curve for the network on the housing dataset