Assignment 2

Deep Learning KU, WS 2024/25

Team Members		
Last name	First name	Matriculation Number
Stanić	Milan	12011449
Höber	Laura Maria	12009439

a) Data analysis

The aim of this exercise is to train a neural network for regression on the openly available California housing dataset. It eventually should be able to predict the target of median housing prices in California based on given feature values.
The first step to train a neural network like this, is to understand the data that we are working with. The input data consists of 8 features, measured relative to a block group that a house is located in, as defined in the following list:

Input features

- median income, in tens of thousands of dollars $ ("MedInc")
- median house age, in years ("HouseAge")
- average number of rooms, ("AveRooms")
- average number of bedrooms ("AveBdrms")
- population size ("Population")
- average number of occupants ("AveOccup")
- latidudal position, in degrees ° ("Latidue")
- longitudinal position, in degrees ° ("Longitude").

The next section tries to visualize the distribution of those features, to determine if any normalization is necessary before using the data is used for training.

Statistics of feature distributions

From the visualizations of the input feature distributions in figure 2 and the corresponding statistics in table 1 it is clearly visible, that their values have vastly different ranges, variances and means. In model training this can lead to biases, i.e. some features will have a greater but unjustified influence on the predictions than others, simply based on the nature of the distribution or higher magnitude of their values.
To mitigate this influence and also make training more efficient, normalization can be applied in the sense of transforming the distributions to be centered around 0 and take on a unity-variance.

$$\vec{x}_{normalized} = \frac{\vec{x} - mean(\vec{x})}{std(\vec{x})} \tag{1}$$

To solve the problem with differing value ranges we could also apply "min-max-scaling", as defined in the following equation, where the values get scaled into a range of 0, +1]. However, some features have a few strong outliers, so this method isn't appropriate, because it would scale down the non-outlier values into a very small interval.

$$\vec{x}_{scaled} = \frac{\vec{x} - min(\vec{x})}{max(\vec{x}) - min(\vec{x})} \tag{2}$$

Table 1 shows the statistics of all the features for the training dataset before normalization. The corresponding distributions are visualized in figure 2.

Normalization is applied to the training set with the "StandardScaler" of sklearn. It was only fitted on the feature values of the training data and then applied to training, validation and test set, which is the reason why mean and variance of validation and test set don't necessarily become 0 and 1 respectively. The transformed feature distributions then take on the shapes shown in figures 4, 6 and 8 with the corresponding statistics shown in tables 3, 5 and 7.

Training set

size: (14448, 8)

	MedInc	HouseAge	AveRooms	AveBdrms	Population	AveOccup	Latidue	Longitude
mean	3.86	28.65	5.43	1.1	1421.79	3.13	35.64	-119.58
std	1.89	12.58	2.41	0.43	1110.97	12.39	2.14	2.01
min	0.5	1.0	0.85	0.33	6.0	0.69	32.54	-124.35
max	15.0	52.0	141.91	25.64	35682.0	1243.33	41.95	-114.47

Figure 1: Statistics of the original data

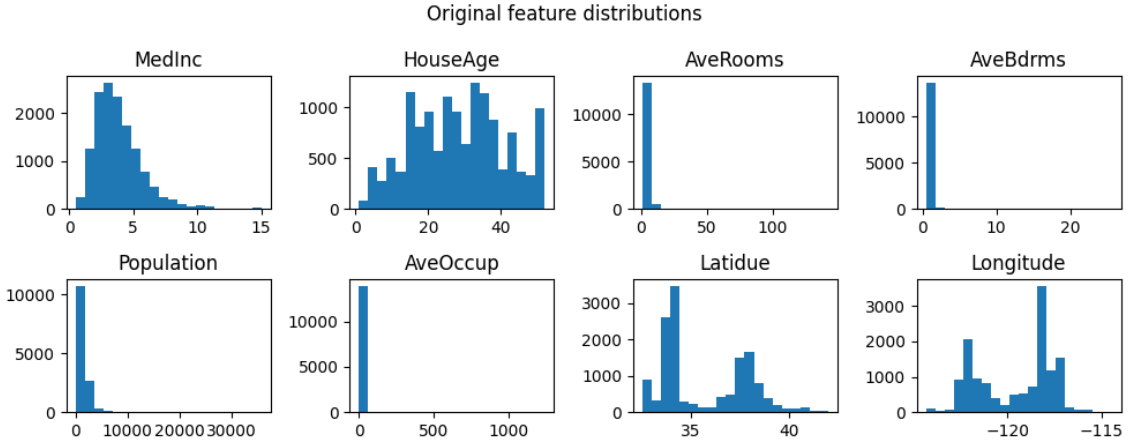

Figure 2: Original feature distributions of the training data

Training set

size: (14448, 8)

	MedInc	HouseAge	AveRooms	AveBdrms	Population	AveOccup	Latidue	Longitude
mean	-0.0	-0.0	-0.0	-0.0	-0.0	0.0	0.0	-0.0
std	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
min	-1.78	-2.2	-1.9	-1.76	-1.27	-0.2	-1.45	-2.37
max	5.9	1.86	56.72	56.66	30.84	100.1	2.95	2.54

Figure 3: Statistics of the normalized training data

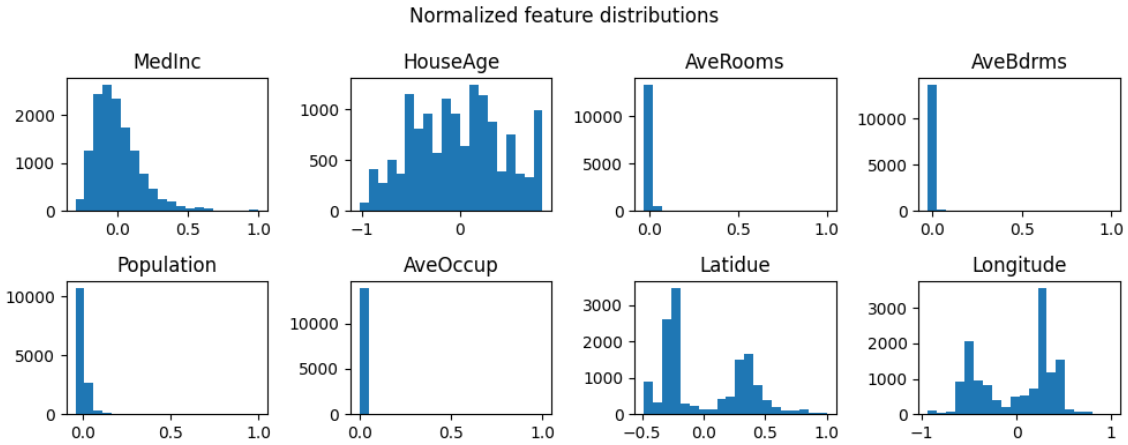

Figure 4: Normalized feature distributions of training data

Validation set

size: (3612, 8)

	MedInc	HouseAge	AveRooms	AveBdrms	Population	AveOccup	Latidue	Longitude
mean	0.02	-0.01	-0.01	-0.03	0.01	-0.02	-0.02	0.02
std	1.03	1.01	0.76	0.66	1.03	0.08	0.99	0.98
min	-1.78	-2.2	-1.68	-1.67	-1.28	-0.19	-1.44	-2.32
max	5.9	1.86	12.9	15.87	12.26	3.07	2.95	2.62

Figure 5: Statistics of the normalized validation data

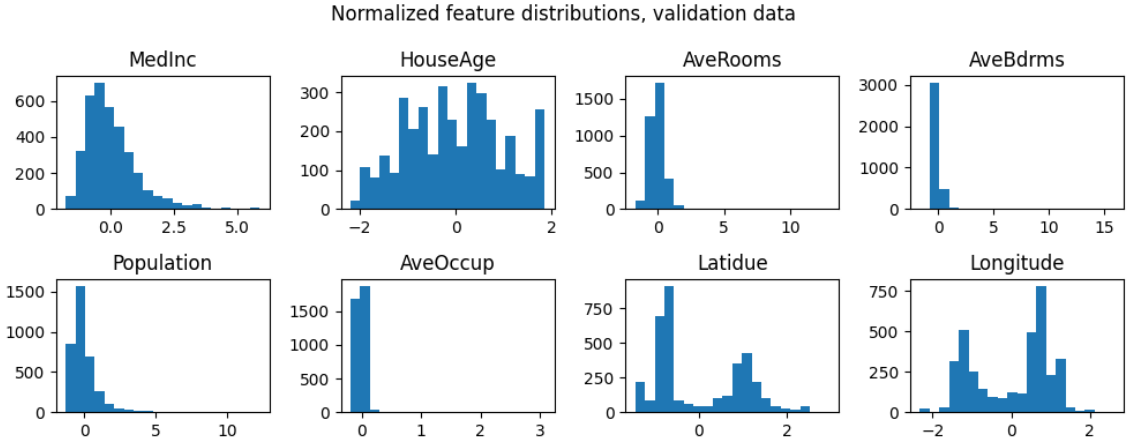

Figure 6: Normalized feature distributions of validation data

Test set

size: (2580, 8)

	MedInc	HouseAge	AveRooms	AveBdrms	Population	AveOccup	Latidue	Longitude
mean	0.01	0.01	0.02	0.03	0.01	-0.02	-0.01	0.0
std	1.01	0.99	1.43	1.84	1.11	0.1	1.0	0.99
min	-1.78	-2.12	-1.89	-1.38	-1.27	-0.15	-1.45	-2.29
max	5.9	1.86	52.82	76.13	24.43	3.9	2.92	2.53

Figure 7: Statistics of the normalized test data

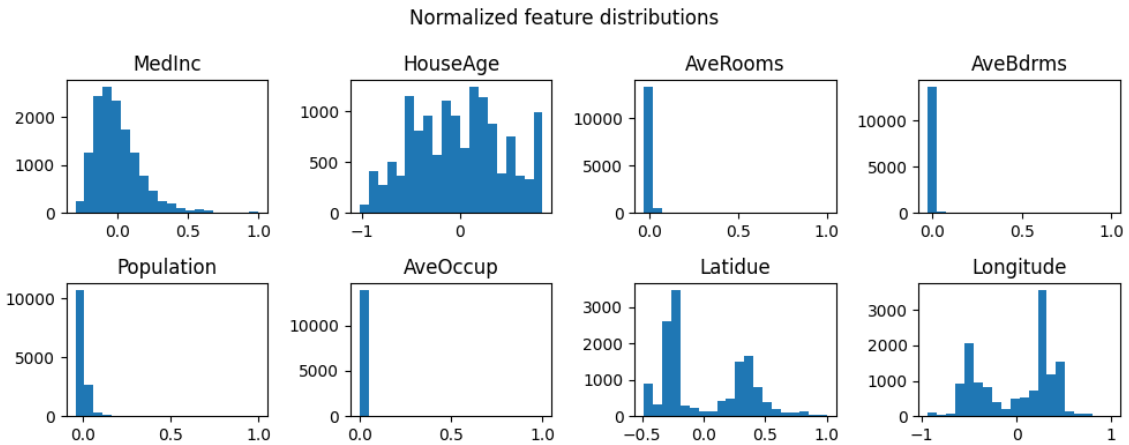

Figure 8: Normalized feature distributions of test data

b) Designing a feed forward neural network

This part of the task aims to compare in total 5 different variations of a feed forward neural network, mainly varying in the number and dimensions of hidden layers.
What they all have in common, is that the first layer takes in the values of the 8 input features, the activation function of the hidden layers is always a rectified linear unit (ReLu) defined as $f(x) = max(0, x)$ and the activation function of the output layer is always the unity function $f(x) = x$. The activation function of the output layer differs from the other layers, since it depends on the underlying task, in this case an 8-dimensional linear regression. We chose a simple linear function, as this doesn't transform our output in any way and keeps the calculation of the derivative easy in the training process.

Network Architectures

Table 1: Layer composition and dimensions of the 5 chosen neural network models. HL... hidden layer, ReLu... rectified linear unit

name	**input layer**	**ReLu**	**HL1**	**ReLu**	**HL2**	**ReLu**	**HL3**	**ReLu**	**output layer**
NeuralNet _default	8×64	-	64×64	-					64×1
NeuralNet _deep	8×64	-	64×128	-	128×64	-			64×1
NeuralNet _wide	8×128	-	128×128	-					128×1
NeuralNet _deeper_wide	8×128	-	128×256	-	256×128	-	128×64	-	64×1
NeuralNet _deep_wider	8×128	-	128×256	-	256×128	-			128×1

Loss Function

As mentioned, we deal with an 8-dimensional regression task with input features $\mathbf{x} \in \mathbf{R}^8$ and the predicted output variable $y(\mathbf{x}) \in \mathbf{R}$. The loss function is therefore chosen as the mean squared error (MSE). It is defined as the squared Euclidean distance between the model predictions and the actual target values:

$$E(\mathbf{w}) = \frac{1}{M} \sum_m ||y(\mathbf{x}^{(m)}) - \mathbf{t}^{(m)}||^2 \quad m = 0, ..., M-1 \tag{3}$$

Evaluation

To compare the performance of the 5 networks, the training and evaluation set loss is calculated for each model. Beforehand, the data was split into training, validation and test data by first splitting off 12,5% of the data as test set (2580 samples), and then splitting off 20% of the remaining samples as validation set (3612 samples), which leaves the rest as training set (14448 samples). Then the data got normalized, as described in section a) and also separated in mini-batches, with additional shuffling, which means, that with every training epoch (one iteration over the whole training/validation set) the samples are assigned to different batches.
The batch size was chosen relatively small, because a smaller batch size generally leads to a better generalization performance [1], although at the same time it increases the training time. A size of 32 was found to be a good trade-off. The number of epochs was fixed to 30 as this has shown to be a point where most models converged quite well. Table 2 compares the final training and validation losses of all models. The best results are highlighted in bold.

Table 2: Comparison of training and test loss between the different models after 30 epochs.

name	**Final training set loss**	**Final validation set loss**
NeuralNet_default	0.2335	0.2826
NeuralNet_deep	0.2461	0.2816
NeuralNet_wide	0.2613	0.2878
NeuralNet_deeper_wide	**0.228**	**0.278**
NeuralNet_deep_wider	0.23	0.2801

c) Comparing Optimizers and Schedulers

As shown in point b), the best network is the *NeuralNet_deeper_wide*. We therefore set it as the default network on the following tasks to be able to compare different learning rates and optimizers.

We use a mix of different schedulers, learning rates and optimizers to find the best hyperparameters for our model and dataset. The search for the best hyperparameters has been reduced in scope to have a computable number of combinations of different hyperparameters. Again, the best results are highlighted in bold.

The learning rates are:

```
lr = [0.001, 0.005, 0.01 ]
```

The Optimizers which are compared are:

```
optimizers = {
        torch.optim.SGD(params, lr=lr, momentum=0),
        torch.optim.SGD(params, lr=lr, momentum=0.9),
        torch.optim.SGD(params, lr=lr, momentum=0.9, nesterov=True),
        torch.optim.RMSprop(params, lr=lr),
        torch.optim.Adam(params, lr=lr)
    }
```

The schedulers are:

```
schedulers = {
    StepLR(optimizer, step_size=15, gamma=0.5),
    MultiStepLR(optimizer, milestones=[5, 10, 20], gamma=0.5),
    CosineAnnealingLR(optimizer, T_max=30),
    CosineAnnealingWarmRestarts(optimizer, T_0=15),
}
```

Table 3: Comparison of training and validation losses of different hyperparameters for learning rate 0.001. The left column being the training and the right column the validation loss for each optimizer/scheduler combination.

LR = 0.001	SGD	SGD Moment.	Nesterov	RMSProp	Adam
StepLR	0.387 | 0.420	0.265 | 0.300	0.261 | 0.302	0.200 | 0.298	0.216 | 0.271
MultiStepLR	0.451 | 0.476	0.288 | 0.321	0.286 | 0.320	0.211 | 0.288	0.227 | 0.280
CosineAnnealingLR	0.420 | 0.448	0.267 | 0.309	0.264 | 0.306	0.191 | 0.276	0.212 | 0.276
CosineAnnealingW.Rest.	0.417 | 0.446	0.266 | 0.308	0.264 | 0.307	0.194 | 0.274	0.216 | 0.275

Table 4: Comparison of training and validation losses of different hyperparameters for learning rate 0.005. The left column being the training and the right column the validation loss for each optimizer/scheduler combination.

LR = 0.005	SGD	SGD Moment.	Nesterov	RMSProp	Adam
StepLR	0.287 | 0.321	0.241 | 0.284	0.238 | 0.298	0.218 | 0.279	0.214 | 0.273
MultiStepLR	0.328 | 0.360	0.241 | 0.284	0.239 | 0.285	0.194 | 0.271	**0.193** | **0.262**
CosineAnnealingLR	0.301 | 0.337	0.226 | 0.277	0.227 | 0.280	0.180 | 0.268	0.193 | 0.263
CosineAnnealingW.Rest.	0.299 | 0.336	0.231 | 0.278	0.228 | 0.280	0.191 | 0.270	0.198 | 0.263

Table 5: Comparison of training and validation losses of different hyperparameters for learning rate 0.01. The left column being the training and the right column the validation loss for each optimizer/scheduler combination.

LR = 0.01	SGD	SGD Moment.	Nesterov	RMSProp	Adam
StepLR	0.267 | 0.306	0.229 | 0.271	0.223 | 0.287	0.233 | 0.284	0.233 | 0.276
MultiStepLR	0.285 | 0.320	0.227 | 0.273	0.226 | 0.275	0.198 | 0.276	0.206 | 0.265
CosineAnnealingLR	0.266 | 0.307	0.209 | 0.264	0.207 | 0.266	0.194 | 0.264	0.196 | 0.269
CosineAnnealingW.Rest.	0.266 | 0.307	0.213 | 0.266	0.213 | 0.269	0.207 | 0.265	0.216 | 0.269

The best performance is given with a learning rate of 0.005. A larger value like 0.01 leads to faster training, but it also causes jumps between low and high loss values during training and can even lead to instabilities. A smaller value like 0.001 would slow down the training process, so that the number of epochs would need to be increased to let the loss converge for all possible model combinations. Another reason, why in this case a learning rate of 0.005 works better than the rate 0.001 used in section b), is that the additional scheduling also slows down the training process over time, so starting off with a slightly higher rate than in b) gives us the best results here.

The best performing optimizer turned out to be Adam in combination with the MultiStep scheduler.

A combination of all the optimal choices resulted in a final training and validation loss of 0.193 and 0.262 respectively. It can also be seen, that CosineAnnealingLR and CosineAnnealingWarmRestart are very similar and only marginally worse performing.

d) Final Training and Results

From the table 5 we can see, that our best combination of parameters is the Adam optimizer with the MultiStep schedular and parameters as given above. The best learning rate, as it is a hyperparameter, was determined as 0.005. A closer view of the loss throughout the final training (with still separated training and validation sets) is shown in figure 9. The epochs are set to 30 as before, as this resulted in the best loss.

Final training loss: 0.193
Final validation loss: 0.262

The final training loss in figure 10 of the best model with the whole combined training and validation dataset doesn't look much different to the previous training loss in figure 9, with the exception of a spike in the early training phase.

Final training loss: 0.202

Afterwards, the model is applied for a last time on the test set with so far unseen data, that hasn't been used in any way by the model yet. This leads to the following test loss:

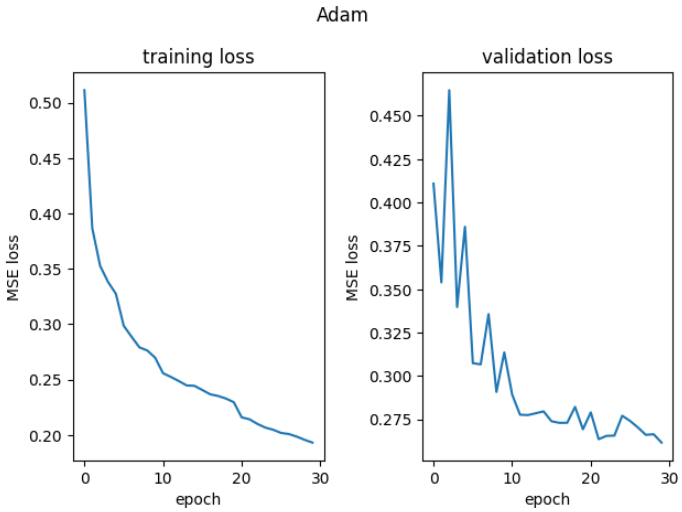

Figure 9: Training and validation loss of of the best model over all epochs.

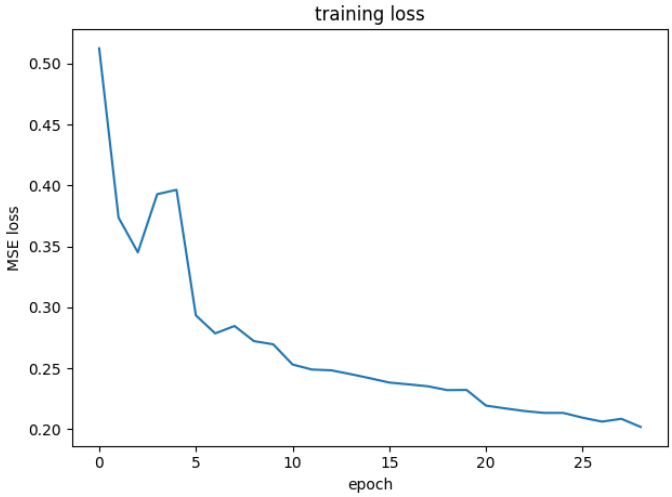

Figure 10: Final test loss of the whole validation and training dataset combined with the final model and hyperparameters shown over the epochs.

Test loss: 0.2617

The test loss is slightly higher than the final training loss. But the difference is not that significant, which indicates that the network was not overfitted, i.e. it didn't adapt itself too closely to the training data. The fact, that the test loss comes close to the final validation loss also indicates, that our neural network is neither influenced too much by underfitting, nor by overfitting.

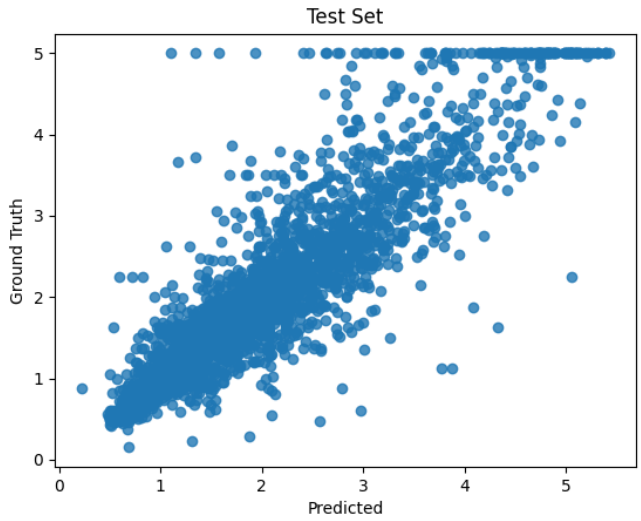

Figure 11: Comparison between prediction values and ground truth for the data points of the test set.

The scatter plot that compares the true and predicted labels (median house price in $100.000) also shows a good correlation, because the points approximately form a straight line with a slope of 1 and no offset. The points on the ceiling of the plot show our outliers and as expected, outliers are hardly trainable and predictable as they lie outside the learned distribution. Especially with such simple neural networks.

e) Equivalence of minimizing Mean Square Error (MSE) and Maximum Likelihood estimator

Task:

Denote the training dataset with $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^d$, and $y_i \in \mathbb{R}$. Let $f_\theta : \mathbb{R}^d \to \mathbb{R}$ denote the neural network. Show that finding the minimizer for the MSE Loss is equivalent to finding the *maximum likelihood estimate*, i.e.,

$$\arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} (f_\theta(x_i) - y_i)^2 = \arg\max_\theta p_\theta(y_1, \ldots, y_n \mid x_1, \ldots, x_n) \tag{4}$$

with the following assumptions: all $y_i \sim p^*(y_i \mid x_i)$ are independently sampled from their corresponding conditional distributions and $p_\theta(y_i \mid x_i) = \mathcal{N}(y_i; f_\theta(x_i), \sigma^2)$ is Gaussian with mean $f_\theta(x_i)$ and fixed variance σ^2 (independent of x_i). Clearly show all steps of your derivation in your report.

Derivation

Starting with the Gaussian definition of $\mathcal{N}(y_i; f_\theta(x_i), \sigma^2)$ for independently distributed samples, we can rewrite the joint probability of the whole dataset as the product of single Gaussians:

$$\mathcal{N}(\boldsymbol{y}; f_\theta(\boldsymbol{x}), \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} exp\left(-\frac{(y_i - f_\theta(x_i))^2}{2\sigma^2}\right) \tag{5}$$

Taking the logarithm of this leads to the following equation:

$$\sum_{i=1}^{n} \left(-\frac{1}{2\sigma^2}(y_i - f_\theta(x_i))^2 - \frac{1}{2}\log(2\pi\sigma^2) \right). \tag{6}$$

As we want $\arg\max_\theta p_\theta(y_1, \ldots, y_n \mid x_1, \ldots, x_n)$ we can ignore the terms that are independent from θ and the constant factor(while still keeping the sign), which leads to the simplified term:

$$\arg\max_\theta p_\theta(y_1, \ldots, y_n \mid x_1, \ldots, x_n) = \arg\max_\theta \sum_{i=1}^{n} -(y_i - f_\theta(x_i))^2 \tag{7}$$

This maximization problem with $\arg\max_\theta$ can be easily transformed to a minimization problem with $\arg\min_\theta$ by inverting the sign:

$$\arg\max_\theta \sum_{i=1}^{n} -(y_i - f_\theta(x_i))^2 = \arg\min_\theta \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2. \tag{8}$$

Compared to the left term of equation 5, the only remaining difference is the constant factor $\frac{1}{n}$, which can be ignored, because it doesn't influence the $\arg\min_\theta$ operation. This shows that finding the minimizer for the MSE Loss is equivalent to the maximum likelihood estimate for an independently sampled Gaussian distribution with fixed variance σ^2.

f) Binary Classification

Changes to the existing structure

Before we can use the existing structure of the so far best performing neural network, first some changes are necessary in order to make it appropriate for solving binary classification problems instead of regression tasks. The input and hidden layers can stay the same, but in the output layer we need a different activation function. The Softmax activation function is appropriate, because it transforms the last layer's outputs to take on values between 0 and 1, so they represent valid probabilities. In our case the adapted form of the log-Softmax function is used instead, to directly output the log probabilities of the output values in a range of $[-\infty, +\infty]$.

Furthermore, the output layer is adapted to produce two output features instead of one, i.e. we eventually get one log probability for the class "median house value larger or equal $200.000" and one for the class "median house value lower than $200.000". The model eventually takes on the structure shown in table 6.

Table 6: Layer composition and dimensions of model 5 , adapted to binary classification. HL... hidden layer, ReLu... rectified linear unit, Log-Softmax... logarithmic Softmax function

name	**Input**	**ReLu**	**HL1**	**ReLu**	**HL2**	**ReLu**	**HL3**	**ReLu**	**Output**	**Log-Softmax**
NeuralNet _deeper_wide _classification	8×128	-	128×256	-	256×128	-	128×64	-	64×2	

Changes to the training pipeline

The training loop also needs a different loss function, as the MSE loss is not sufficient for a binary classification problem. The use of Negative Log Likelihood (NLL) loss is a good replacement.

Lastly, in the model evaluation after training it is now possible to use the accuracy metric to check the network's performance in addition to the test and training loss.

Model performance

Here, the model is immediately trained with the combined training and validation set from before and then evaluated on the test set. It resulted in the training and test losses shown in figure 12.

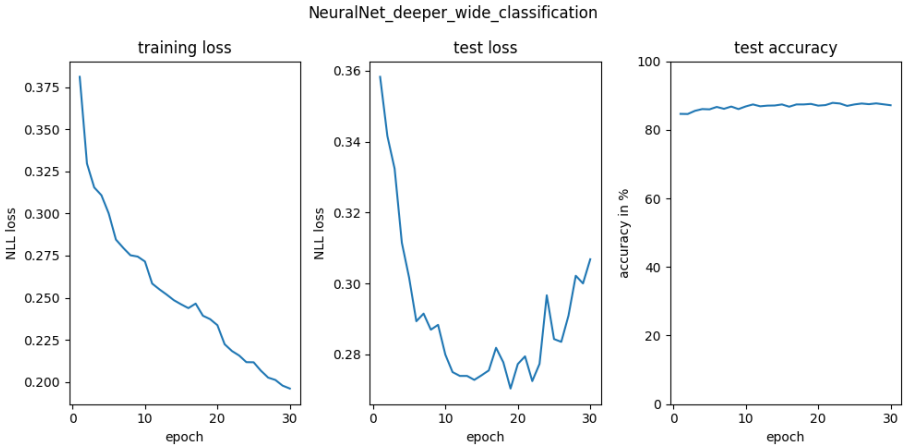

Figure 12: Training loss, test loss and test accuracy for the final neuronal network, adapted to binary classification

Final training loss: 0.1961
Final test loss: 0.3068
Final accuracy: 87.17 %

Although the model was initially intended to solve regression tasks, this adaptation works relatively well for classification, with a final accuracy of 87.17% that is far above the chance level of 50%. Seeing the test loss we can see that the network is starting to overfit during training and loses generalization capabilities as seen on the test set loss. Therefore one could examine the hyperparameters for the classification task separately and would probably find a different hyperparameter-set better fitting for this classification task. For example, either lowering the number of epochs to about 20 or decreasing the learning rate could help. Still, the losses aren't as far apart and this is still a good result considering the simplicity of our training network.

But one needs to be cautious, as including the test set in any kind of training procedure and any knowledge transfer is strictly forbidden and introduces bias!

References

[1] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2017.