Deep Learning for Artificial Intelligence Engineering

Assignment 1

Training Simpe Neural Networks

Student:
Amer Delić (01331672)

Lecturers:
Ozan Özdenizci
Simon Hitzginger

Institute of Machine Learning and Neural Computation
Graz University of Technology

Graz, November 2025

Contents

1 Training simple neural networks **3**
1.1 Dataset exploration and preprocessing 3
1.2 Neural network design 4
1.3 Final model and evaluation 5
1.4 Binary classification extension 6

Appendix **8**

List of Figures

1.1 Feature distributions 3
1.2 Training and validation loss evolution of the final model 5
1.3 Predicted vs. true target values on the test set 6
1.4 Predicted probability distributions for the two classes 7

1 Training simple neural networks

The main goal and detailed description of this assignment are provided in the attached assignment sheet. This report focuses on presenting the implementation, experimentation, and results corresponding to those tasks.

1.1 Dataset exploration and preprocessing

The original training set was further split into a training subset and a validation subset using an 80/20 split. The validation set was then used for model selection, while the test set was held out for final evaluation.

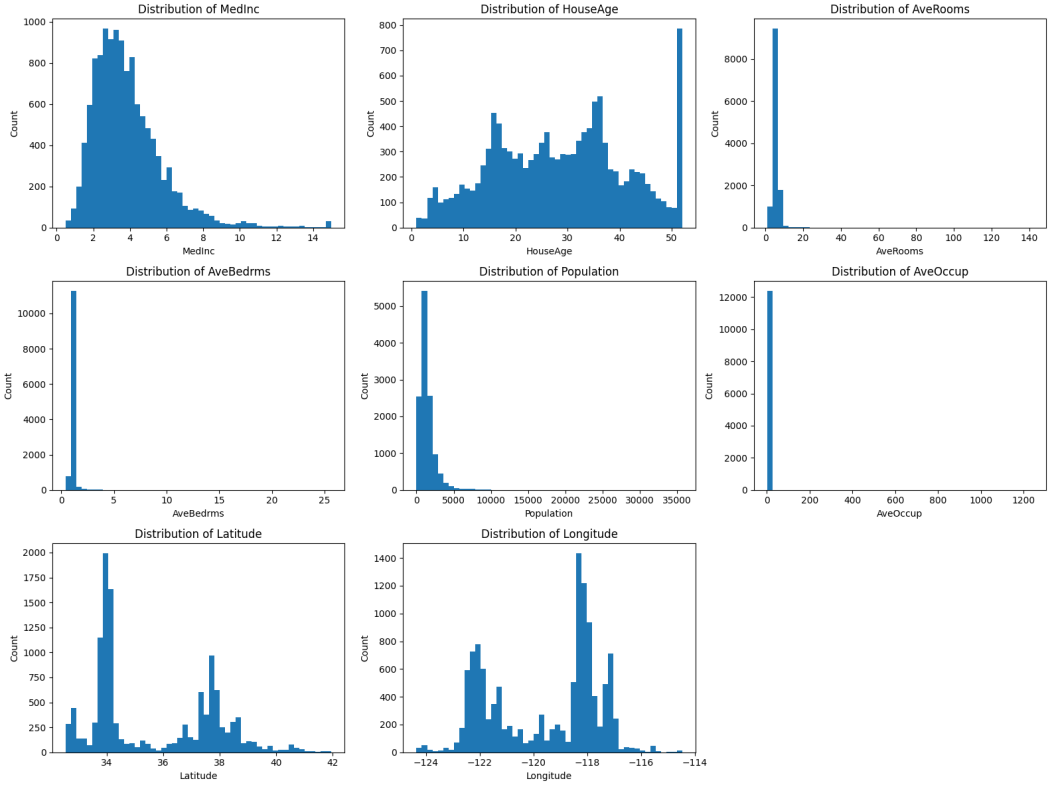

Figure 1.1: Feature distributions

To ensure that all features contribute equally to the learning process and to improve training stability and convergence, the data was normalized. For this purpose, the `StandardScaler` from the *scikit-learn* package was used, which transforms features to have zero mean and

unit variance. Normalization of the training data was performed using the `fit_transform()` method, which computes the mean and standard deviation of the given dataset and scales the data accordingly. The validation set was normalized using the `transform()` method, which applies the scaling based on the previously computed statistics from the training set. This procedure prevents information leakage from the validation or test sets into the training process.

To investigate the feature distributions, histogram plots of individual features were created and are shown in Figure 1.1.

The histograms show the distributions of the normalized input features. As expected, most features are centered around zero with unit variance, confirming that normalization was correctly applied. Nevertheless, the original distributional shapes remain visible. *MedInc*, *AveRooms*, *AveBedrms*, and Population show (right-)skewed patterns, indicating long-tailed distributions and the presence of outliers, while *Latitude* and *Longitude* display non-uniform distributions. Such characteristics — non-uniform distributions, skewed patterns and outliers — can affect model training by introducing uneven learning dynamics, gradient imbalance, and potentially leading to biased weight updates or overfitting. In contrast, *AveOccup* shows a concentration near the mean with a few extreme values, while *HouseAge* appears more uniformly distributed, which helps reduce bias in training but may not be very informative unless combined with other features.

1.2 Neural network design

For the regression task, a feedforward neural network was designed and trained using stochastic gradient descent (SGD). The architecture consists of an input layer matching the number of the features, followed by one or more hidden layers with ReLU activation, and an output layer with a single neuron - since the goal is to predict a single target variable - and no activation function - allowing the model to produce unbounded continuous outputs. Mean Squared Error (MSE) was chosen as the loss function since it is mathematically convenient and easy to interpret. Furthermore, it directly measures the average squared difference between predicted and actual values, penalizing larger errors more heavily, making it sensitive to the outliers.

To identify an effective neural network setup for this regression task, 10 different configurations of hidden layer structures were trained and evaluated. Starting with simple networks containing a single hidden layer, the depth and width of the architecture were systematically increased to observe their impact. Optimization hyperparameters — including the SGD learning rate, number of epochs and batch size — were adjusted accordingly. Finally, training and validation losses for each configuration were compared, and the results are summarized in Table 1.1.

Configuration	Hidden Layers	Neurons per Layer	Learning Rate	Epochs	Batch Size	Train Loss	Validation Loss
1	1	32	0.001	100	32	0.403	0.440
2	1	64	0.001	100	32	0.395	0.433
3	1	128	0.001	200	128	0.422	0.463
4	2	32,16	0.001	100	32	0.372	0.400
5	2	64,32	0.001	100	32	0.347	0.374
6	2	128,64	0.01	100	64	0.260	0.329
7	3	64,32,16	0.001	150	32	0.315	0.344
8	3	128,64,32	0.01	100	64	0.251	0.295
10	3	64,64,64	0.001	150	32	0.292	0.325
11	4	128,64,32,16	0.001	200	32	0.260	0.303

Table 1.1: Comparison of different configurations

1.3 Final model and evaluation

After comparing the different architectures in the model selection phase, the configuration with three hidden layers of sizes 128, 64 and 32 was chosen as the final model. This architecture provided the lowest validation loss among the tested setups.

To better understand how this final model learns, the training and validation losses recorded during its training process are shown in Figure 1.2. The losses decrease smoothly, and the gap between them remains relatively small, indicating that the model generalizes well and does not strongly overfit.

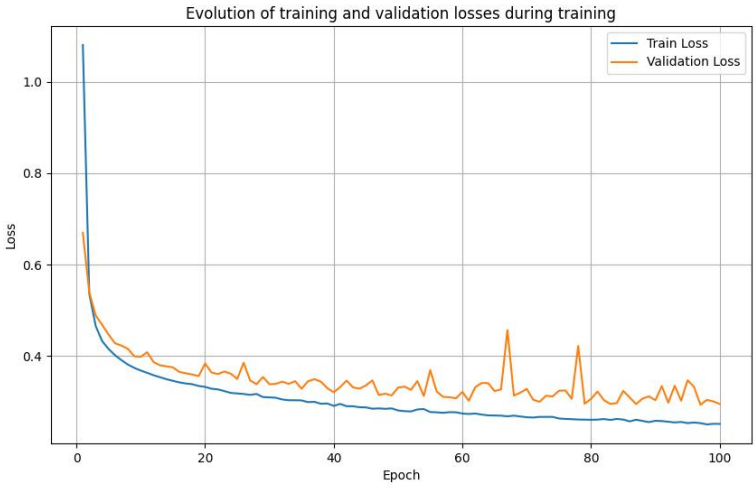

Figure 1.2: Training and validation loss evolution of the final model

Once the architecture was fixed, the model was retrained on the full training set. This ensures that the model benefits from all available training samples before the final evaluation. The same preprocessing and scaling rules were applied as before.

The test loss obtained with the final model shows that it performs well on unseen data (train loss: 0.254, test loss: 0.285). To visually inspect how well the model fits the regression task, a scatter plot comparing the predicted values with the true target values is shown in Figure 1.3. Ideally, the points should lie close to the diagonal line, which represents perfect predictions. The results show that most points follow this trend, although some variance remains.

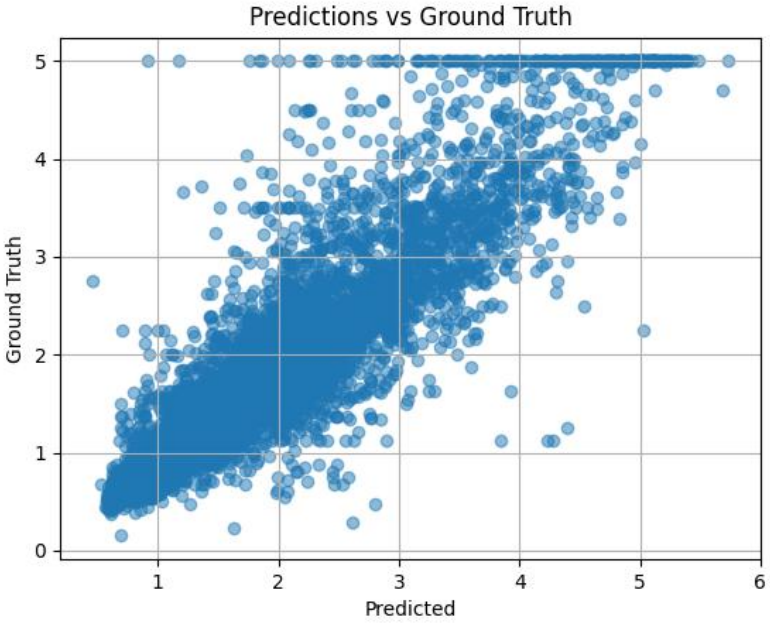

Figure 1.3: Predicted vs. true target values on the test set

Overall, the final model provides a good balance between complexity and generalization.

1.4 Binary classification extension

In the last part of the assignment, the regression problem was converted into a binary classification task. The goal was to predict whether the median house value is below or above \$200,000. For this purpose, the target values were transformed into two classes: **0** for values below 2.0 and **1** for values equal to or above 2.0.

The same neural network architecture used in the regression task was reused here. However, the training pipeline was adjusted to match a classification setting. A sigmoid activation was applied to the output so that the model produces probabilities between 0 and 1. Instead of the MSE loss, binary cross-entropy (BCE) was used, which is more suitable for evaluating how well the predicted probabilities match the two target classes.

The model was trained on the full training set for 100 epochs. During training, the BCE loss steadily decreased, showing that the network was learning to separate the two classes. After training, the model was evaluated on the test set. It achieved a test BCE loss of **0.4069** (while BCE train loss: 0.398) and a classification accuracy of **81.12%**. These results indicate that the model can correctly classify most samples.

A visualization of the predicted probability distributions for the two classes is shown in Figure 1.4. The plot illustrates that the model generally assigns low probabilities to class 0 samples and high probabilities to class 1 samples, but the distributions are not perfectly separated, which explains the remaining classification errors.

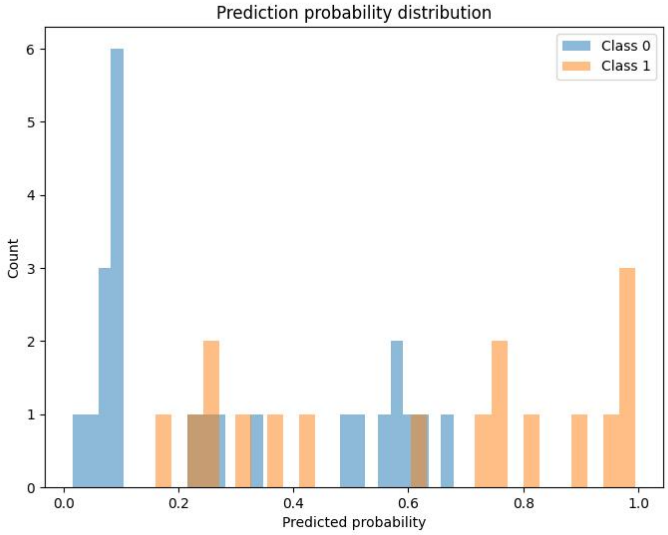

Figure 1.4: Predicted probability distributions for the two classes

Appendix

Deep Learning for AIE (708.002) WS25
Assignment 1
Training Simple Neural Networks

Ozan Özdenizci, Simon Hitzginger
Institute of Machine Learning and Neural Computation, TU Graz
oezdenizci@tugraz.at, simon.hitzginger@tugraz.at

Points to achieve:	15 pts
Assignment Issued:	30.10.2025 16:00
Deadline:	20.11.2025 16:00 (**no extensions**)
Hand-in procedure:	You **can** work in groups of **at most two people**. **Exactly one** team member uploads two files to TeachCenter: **The report (.pdf)** and **the Python code as .py or .ipynb**. **Do not upload a folder. Do not zip the files.** The first page of the report **must** indicate the group partner.
Assignment interviews:	You might be invited for an assignment interview.
Plagiarism:	If detected, 0 points for all parties involved.

We will use PyTorch to train a neural network for a regression task on the California Housing Dataset. Our goal is to predict the values of the houses based on various predictive variables. Use the following code to fetch the dataset and split it into a training and test dataset:

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
X, y = fetch_california_housing(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=302)
```

There are 15,480 training and 5,160 test examples with 8 predictive input features and 1 target variable. Input features consist of the spatial locations of the districts that data is collected from (latitude, longitude), demographic information in the districts (income, population, house occupancy), and general information regarding the houses (number of rooms, number of bedrooms, age of the house). Since these statistics are measured per district, the features correspond to averages or medians across "block groups" (a geographical unit with a population of 600 to 3,000 people).

The input features (in `X_train` and `X_test`) are provided in the following order:

- `MedInc` : median income in block group
- `HouseAge`: median age of a house within a block
- `AveRooms`: average number of rooms per household
- `AveBedrms`: average number of bedrooms per household
- `Population`: block group population
- `AveOccup`: average number of household members
- `Latitude`: a measure of how far north a house is
- `Longitude`: a measure of how far west a house is

The target variable is the *median house value* for California districts, expressed in hundreds of thousands of dollars ($\$100,000$).

Task details:

For all the tasks below, create the appropriate code and discuss your experimentation and reasoning process, findings and choices in your report. **All results should be shown in the PDF report. The submitted code should be able to reproduce your results**.

a) (3 pts) : Get familiar with the dataset. Split `X_train` into an actual *training set* and a *validation set*. We will use this validation set during the model selection process. We will use the *test set* only to evaluate the final model (i.e., *after* model selection). Investigate the feature distributions, include your plots in your report, and discuss your main observations.

You will have to normalize the features of this dataset. When you do that, make sure you do not compute statistics using the validation or test set. Have a look at the *hint* below, and explain in your report what is the difference when using `fit_transform` and `transform`.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
your_train_split = scaler.fit_transform(your_train_split)
your_val_split = scaler.transform(your_val_split)
```

b) (6 pts) : Design your feedforward neural network architecture for the regression task. Explain your design choices for the output layer and the loss function that you will use during training. Minimize the training loss using stochastic gradient descent (SGD) with a suitable mini-batch size. Try at least 10 different configurations by varying numbers of hidden units and hidden layers. Provide a table where training and validation losses of various architectures and optimization hyper-parameters (e.g., SGD learning rate, number of epochs) are compared.

c) (2 pts) : Clearly summarize your final model once your architecture choices are fixed. Provide a plot where the evolution of the training and validation loss during training are shown throughout iterations. Perform a final training with this model on the whole training set (i.e., combine the proper training set and the validation set). Report and comment on the final test loss. Provide a scatter plot in which you compare model predictions (x-axis) with their ground truth values (y-axis) on the test set.

d) (4 pts) : Now assume that we want to use a similar architecture for the binary classification problem of determining if the median house value is below or over $\$200,000$. Explain which parts of the architecture design and the training pipeline you would have to change, and which test set evaluation metrics should be investigated in this case. Explain in detail the reasons for these differences.

Implement these changes to the final architecture and the training pipeline you had in Task (c), and train a single model for this binary classification task using the whole training set. Note that you will also need to redefine your target variables by executing the lines:

```
y_train[y_train < 2], y_test[y_test < 2] = 0, 0
y_train[y_train >= 2], y_test[y_test >= 2] = 1, 1
```

Evaluate your model on the test set, report and comment on its performance.