Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Haakon Muggerud

Feed-forward neural networks and how to explain their predictions

Bachelor's project in Mathematical Studies
Supervisor: Mette Langaas

May 2021

NTNU
Norwegian University of
Science and Technology

Haakon Muggerud

Feed-forward neural networks and how to explain their predictions

Bachelor's project in Mathematical Studies
Supervisor: Mette Langaas
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Contents

Abstrakt Et kunstig feed-forward nevralt nettverk bruker lag med noder knyttet sammen av vekter til å kalkulere prediksjoner på basis av et input. Denne arkitekturer i kombinasjon med ikke lineære aktiverings funksjoner gjør at feed-forward modeller kan bli brukt til komplekse regresjons- og klassifikasjonsproblemer. Feed-forward modeller bruker observert informasjon til å optimalisere vektene i modellen. Dette gjør den ved å se på en tapsfunksjon og bruker gradienten sammen med kjerneregelen til å forandre på vektene.
Feed-forward modeller er en del av en modellfamilie kalt black-box modeller. I en black-box modell er kun inputtet og prediksjonen kjent, men hvordan modellen ender opp med denne prediksjonen er ukjent. Black-box modeller er mer brukt i dagens samfunn og det er derfor ikke bare en interesse å se hva som skjer på innsiden av modellen, men også regler for å sikre rettferdigheten ovenfor individene som er berørt av en black-box modell. Black-box modeller kan ofte være kompliserte og vanskelige å beskrive, men metoder som ICE og PD plot gjør det mulig å få en bedre forståelse av modellen ved å se på hvordan komponenter i data settet forandrer prediksjonen til modellen.

Keywords Feed-Forward Neural Network · Black-box · ICE plot · PD plot

Abstract An artificial feed-forward neural network uses layers with nodes linked together with weights to make predictions from an input. This architecture in combination with the non-linear activation function makes the feed-forward model able to be used to solve complex regression and classification problems. The model uses observed data to optimize its weights by looking at a loss function and uses the loss functions gradient and the chain rule to update its weights.
Feed-forward models are a part of a model family called black-box models. In a black-box model only the input and the prediction are known, but how the model comes up with these predictions is unknown. Because these models are more and more used in everyday life there is not only an interest to look inside the black-box, but also regulations in place to ensure fairness for the individual affected by these models. Explaining these models can be complicating, but methods like ICE and PD plots can make an individual get a better understanding of which features affect the model's prediction.

Keywords Feed-Forward Neural Network · Black-box · ICE plot · PD plot

1 Introduction

In this thesis, we will take a closer look at the mathematical foundation and methods used for estimating parameters in artificial neural networks. The models do not rely on complex mathematical ideas but can combine simple ideas in a way that makes you able to solve complex problems. We will look at many aspects of a feed-forward neural network model from architecture to optimization.

Today more and more decisions are automatically made by machine learning. Therefore, it is important to understand what the model does and how it makes its predictions. This is to ensure that both the people working with the model can further enhance it, and ensure that the people affected by the model also understand it. We are going to look at two methods that visualize how the model makes its prediction and how the desired feature affects the prediction of the model. These methods are called ICE and PD plots. We will study how they are calculated and how to interpret their outputs.

For the last part, we are going to use the methods discussed in the thesis on a data set and show how one can analyze a real data set with artificial neural networks and interpret the fitted model with ICE and PD plots.

2 Deep feed-forward artificial neural networks

Assume we have observed one data set consisting of N observations of p_0 features and one output.

2.1 Network architecture

A feed-forward neural network has a layered structure. The first layer is an input layer and the last an output layer. In between, we may have several hidden layers. Each layer consists of nodes. Each layer, except the output layer, also has a so-called bias node, which is equivalent to an intercept term in a regression model. The input layer consists of observed features. The hidden layer is where the model finds patterns to produce a prediction based on the input data. In the output layer is the predicted answer, the output from the model. The output can either consist of multiple nodes, for a classification model, or a single node, for a regression model. We will focus on a regression model.

In Figure 1 we see a feed-forward neural network with two hidden layers. We will use the following notation for the network parts.

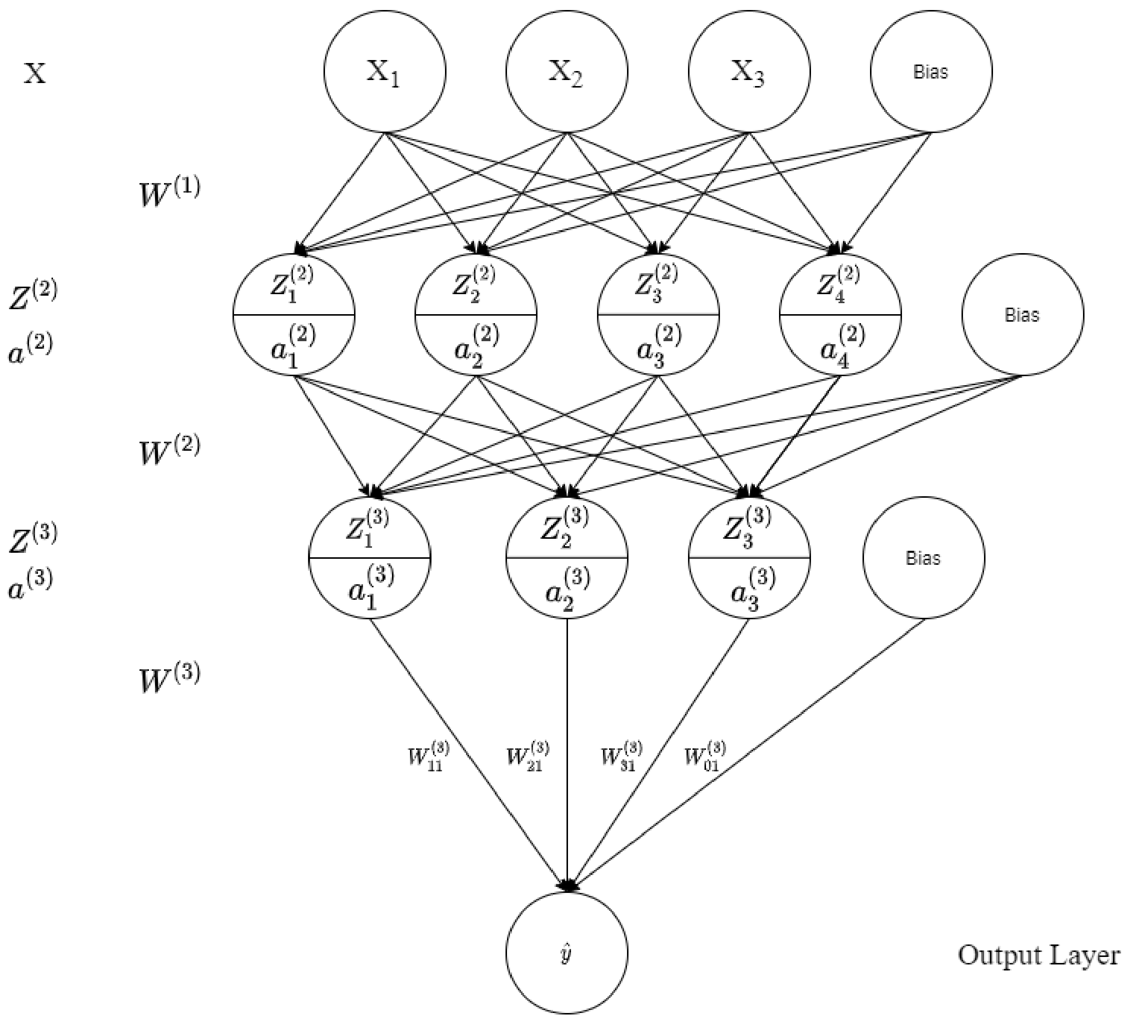


Figure 1: Visual representation of an artificial feed-forward neural network.

- X: A matrix of dimension $(N \times p_0)$ where $(X_1, .., X_{p_o})$ are column vectors. $X_1(N \times 1)$ is a vector with input features from a single observation. N is the number of observations, and p_0 is is the number of input features, which equals the number of nodes of the input layer.
- $W^{(l)}$: Weights going from layer l to layer $l+1$, $l = 1, 2, .., m-1$ and m is the number of layers in the whole network, including the input layer, all hidden layers and output layer.
- $W_{uv}^{(l)}$: The weight going from node u in layer l to node v in layer $l+1$. Here $u = 1, 2, ..., p_{m-1}$, $v = 1, 2, ..., p_m$, p_l is the number of nodes in layer l.
- $Z^{(l)}$: The hidden layer l, $l = 2, 3, .., l-1$.
- $Z_j^{(l)}$: The j'th node of the hidden layer $Z^{(l)}$, $j = 1, 2, .., p_{l-1}$.
- $f(Z^{(l)})$: Activation function on the nodes in layer $Z^{(l)}$.
- $a_j^{(l)}$: The nodes in layer l after applying an activation function.
- $\hat{y}$: The predicted output of the neural network.

An artificial neural network can be used for a vast variety of problems, but all with the same idea. The idea is to use observed data to make a model that predicts an output, as correct as possible, based on patterns found in the observed data.

2.2 Mathematical model

An artificial neural network always starts with an input matrix, X. All the data about the features are stored in the $(N \times p_0)$ dimensional input matrix. This data is then passed into the nodes in the first hidden layer using a set of weights. The first hidden layer is then a product of the input layer and the first set of weights.

$$Z^{(2)} = XW^{(1)} \tag{1}$$

To be able to model non-linear functions we use an activation function on equation (1), giving equation (2). More information about activation functions will be given in Section 2.3.

$$a^{(2)} = f(Z^{(2)}) \tag{2}$$

We now feed the results of equation (2) to the nodes in the next hidden layer, using a new set of weights.

$$Z^{(3)} = a^{(2)}W^{(2)} \tag{3}$$

$$a^{(3)} = f(Z^{(3)}) \tag{4}$$

Using the same formula, applying an activation function to the values of the hidden layer and feeding it to the next hidden layer using a set of weights, can be used several times, dependent on the number hidden layers in the model. If the feed-forward neural network is used for classification, an activation function will also be used for the output layer.

$$\hat{y} = a^{(3)}W^{(3)} \tag{5}$$

Comparing the output to the true response we asses how well our model performs. In Section 2.4 we look at how we use the residual, the difference between the networks predicted response and observed response, to train the model.

2.3 Activation function

Artificial neural networks are inspired by the neurons in the brain. Neurons are binary, meaning they are either on or off, or as we going to use, either firing or not firing. Neurons bases their decision on firing or not firing on information from previous neurons either firing or not firing. This decision is then passed onto several other neurons, which also has to decide on firing or not firing. This forwarding of information eventually leads our body to move in a specific way, based on the information given. We can mimic the choice of firing or not firing in an artificial neural network by using an activation function. Another reason to use an activation function is that it adds a non-linearity to the model, preventing the model from just having linear operations. We will look at two commonly used activation functions, the Sigmoid and the Rectified Linear Unit function (ReLU). There is also possible to use a combination of multiple activation functions in a model, for example using ReLU as an activation function for the hidden layers and Sigmoid as an activation function for the final layer, outputting something that can be interpreted as a probability (Goodfellow et al., 2016, page 63).

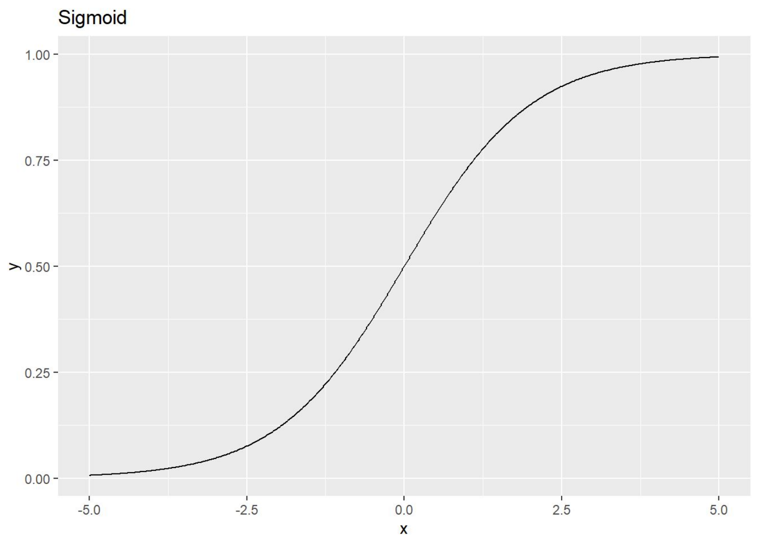


Figure 2: Visual representation of Sigmoid function

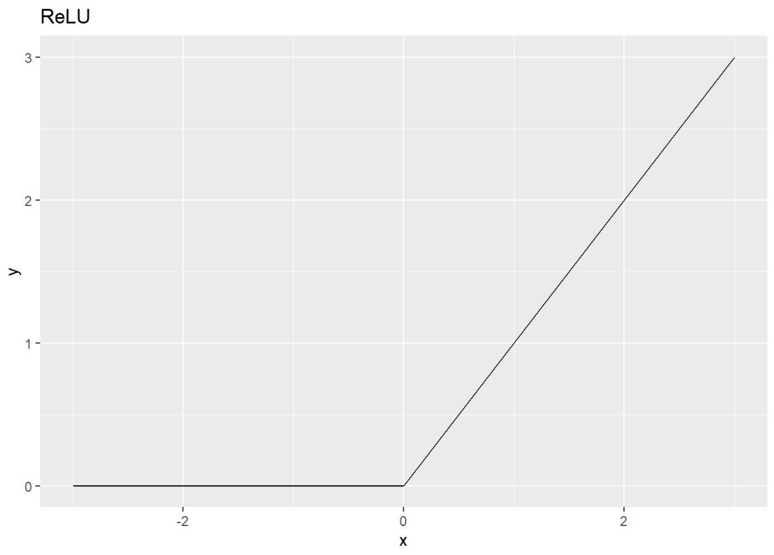


Figure 3: ReLU function

Sigmoid

$$f(z) = \frac{1}{1 + e^{-z}} \tag{6}$$

The Sigmoid function, equation (6) and figure 2, was the previously preferred activation function for the hidden layers. The function has two horizontal asymptotes, 0 when $z \to -\infty$ and 1 when $z \to \infty$. The potential problem with the Sigmoid is that the gradient is 0 at both tails, making the gradient descent method not efficient (Section 2.4 we talk about gradient descent method).

ReLU

$$f(z) = \max(0, x) \tag{7}$$

The ReLU function, equation (7) (figure 3), is now the most common activation function to use on the nodes in the hidden layers of a feed-forward neural network. Unlike the Sigmoid function, which gives some activation for negative values, ReLU always sets negative values to 0. The ReLU also has no upper limit to how "activated" a node can potentially be. So even though it will have a gradient equal to 0 for negative values, will it not have a gradient equal to 0 for positive values.

2.4 Optimization

We have in Section 2.2 presented the architecture of the network, and seen that the network model consists of unknown weights and biases that create an output. The goal of the optimization is to change the weights and biases in an effective way that makes the model as accurate as possible, in the sense that the output $\hat{y}$ is as close as possible to y, the observed response. We will use the feed-forward neural network to perform a regression task, with the observed data (X, Y). We next want to estimate values for the network weights. This is done by minimizing a loss function, equation (8).

$$L(y, \hat{y}) = \frac{1}{N}\sum_{l=1}^{N}(y_i - \hat{y}_i)^2 \tag{8}$$

A loss function compares the output to the observed response. Due to the non-linearity of an artificial neural network, most loss functions become non-convex. Therefore, rather than finding the global minimum an iterative gradient-based optimizer is used to lower the loss function (Goodfellow et al., 2016, page 86).

Gradient descent

From calculus it is known that the gradient of a function, $\nabla L(y, \hat{y})$, gives us the direction of the steepest ascent, so by taking the negative of the gradient, we get the direction of the steepest descent. Changing the weights in this direction will lower the loss function, and therefore indicates the directions we want the weights to change in. In combination with the direction, a *step-size* is often utilized to indicate how far in the given direction the weight should change. Too small of a step-size would mean the gradient method would take many iterations to converge and have the possibility to get stuck in a local minimum. On the other side, a large step-size could ensure ending up in random places in the graph.

Stochastic gradient descent
Using the gradient descent to find a minimum in a neural network with few features and observation is no problem. Unfortunately, neural networks often consist of thousands if not millions of features and observations, so the method becomes expensive. Therefore we need an alternative method for using gradient descent, this method is called stochastic gradient descent. Stochastic refers to the random choosing of points. True stochastic gradient descent takes a single random point and uses that points gradient to give a direction for the whole set. This offers a low cost but is inaccurate. Batch stochastic gradient descent uses all the points, and is very accurate, but as mention above, can be expensive. Between these two extremes, there is an efficient compromise called *mini-batch stochastic gradient descent* (Goodfellow et al., 2016, page 45).
Mini-batch stochastic gradient descent randomly draws a sample of the points and uses these points to compute a loss. Then calculating the gradient of the loss function from these points, and changes the weights based on the gradient.

Backpropagation
The core idea of backpropagation is to use the chain rule to find the partial derivative of each weight from the loss function. An artificial neural network is essentially a function consisting of nested functions, so by applying the chain rule, $\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$, we can find the partial derivative of every weight and use gradient descent to change them. The result of backpropagation is an optimal set of weights, $\hat{W}$.

2.5 Model assessment and model selection

One of the first things we do when training a model using a data set is to split the data set first into two parts, training data, and test data. The training data is needed for model selection and the training of the model. This is done by dividing the training data into two new sets, a training set, and a validation set. On the training set, we use the gradient descent method to change the weights, so the model performs as well as possible on the training set. The validation set is used for model selection, by avoiding *overfitting* and implementing *early stopping.* After the model is trained it is then exposed to the test data, data the model never has seen before. This is to evaluate how well the model will perform on new data.

Overfitting and early stopping

A common strategy when dealing with a neural network is choosing a model that is too big. The global optimum of the model will then overfit the training data. A validation set is then used to prevent overfitting, by implementing early stopping. It is important to keep in mind that the model is optimized on the training data, and the validation set is not used in the optimization. Overfitting is when the model is so tailored to the training data that it starts performing worse in the general case. A reason for overfitting can be that the model finds patterns that only exists in the training set, and not in the general case. A way to spot overfitting is to compare the error of the model on the training set to the error of the model on the validation set. If the model keeps performing better on the training data, but not on the validation set, this can imply overfitting. A method to prevent the model to overfit on the training data is early stopping. Early stopping looks at the performance of the model on the validation set and stops the training of the model when the model performs best on the validation set. "When" refers to how many epochs the model has gone through. An epoch is one cycle through the training set using gradient descent to optimize the model. Training a neural network often takes multiple epochs. The early stopping tells the model the optimal number of epochs to run through to perform best on the validation set.

K-fold cross-validation
We often have a limited amount of observed data, and we want to use the available data efficiently. One way to do this is the K-fold cross-validation. This method divides the training data into K different parts or folds. Each fold is then used as a validation set once, and as a part of the training set the other $K-1$ times. This way the model trains on K slightly different training sets. It then uses the average of the loss on the validation set as an estimate for how the model performs.

Hyperparameters
All parameters within the model that are not automatically changed or need to be manually computed are called hyperparameters. This includes the number of hidden layers, nodes within the layers, step-size, and the number of epochs the model should train on. A popular strategy is to select a set of values for each hyperparameter and then evaluate all combinations of these hyperparameters. This method is referred to as grid search. This is not the main topic of this article and we have only used the method of early stopping to choose the number of epochs to train the model on in Section 4.

3 Explainable AI

3.1 Black box and why we need to look inside it

Many machine-learning methods like Random Forest, Neural Networks, and other Ensemble methods are what we call black-box model. What identifies a black-box model is that it starts with a problem and finds a prediction, how it ends up with this prediction is unknown, and happens within the black box. If the model performs well and makes good predictions one might think there is no reason to looking inside the black box, but there are many reasons for why one want to look inside.

Black-box models are now used in many different fields. A doctor can use a black-box model to predict the relapse of a cancer patient, or a data scientist uses black box models to make complex models predicting everything from the stock market to insurance. For many of these users, it would be of interest to look at what is going on within the black box. For instance, the doctor may want to check what variables determine whether or not a patient is going to relapse to trust or learn from the model, or the data scientist wants to study which variables determine the predictions to improve the model.

Due to the rising popularity of using black-box models for predictions, a regulation took place in 2018 called General Data Protection Regulation (*2018 reform of EU data protection rules* 2018). This regulation states that an individual has the right to a "meaningful explanation of the logic involved" when an automatic decision has taken place. The reason for the regulation was to secure fairness for the individuals involved. For example, may a bank use a black-box model to determine if a person is eligible for a loan. If this person is rejected because of the model she has the right to know why the loan was rejected in a way that makes her understand the reason behind it. How a black-box model makes its predictions are hard enough for a data scientist to understand, so describing it in an understandable way such that everyone at least understands parts of is difficult. In this section, we are going to review some of the methods that are used to explain what happens inside the black box.

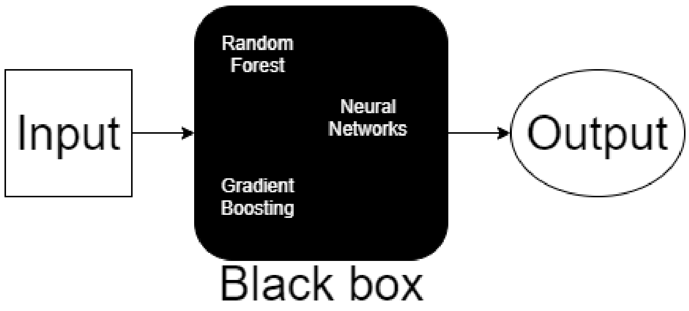


Figure 4: Visual representation of a black box model, with some model that is characterized as black box models inside.

3.2 How to look inside the black box

There are many methods to look at what is going on inside the black box. One method is to look at the importance of the features in the model. This indicated which features drives the prediction, but nothing about how the expected prediction varies depending on a single feature.

Some of the methods are global and others are local. A global method tries to explain the whole model and how the model predicts its output. For example, if a doctor wants to know how the model predicts a relapse, the doctor would want to use a global method, like PD or ICE plot. On the other side, if the doctor is interested in how a specific patient's set of genes affect the prediction of the model, he would use a local method like LIME or Shapley. The local methods aim to see how a specific input feature varies the prediction of the model.

Partial dependants plot and individual conditional expectation plot, which we will be looking at in Section 3.3 and Section 3.4 respectively, are examples of methods that are model agnostic. If a method is agnostic it means that it works on any black-box model. The opposite of this is a model-specific method, that only works on a specific black-box model.

3.3 Partial dependence plot

Partial dependence (PD) plot is an global model agnostic method that aims to show the marginal effect of a feature on the prediction of the model. The idea is to see how the prediction changes when a set of features are varied and the others are kept fixed. As an example, PD plots can be used to see how the weather effects the number of bikes rented, if you have a model that predicts the number of bikes rented based on many features including the weather.
We need to define PD plot mathematically. Let $S \subset \{1, ..., p\}$ and C be the complement set of S. S and C are here index subsets of predictors. Example if $S = \{1, 2\}$, then x_S refers to a 2×1 vector containing the first and second coordinates of x. We define the partial dependence function f on x_S as:

$$f_S(x_S, x_C) = E_{x_C}[f(x_S, x_C)] = \int f(x_S, x_C)dP(x_C) \tag{9}$$

Each subset S has its own partial dependence function f_S. This function gives the average value of f when x_S is fixed and x_C varies as explained by the marginal distribution $dP(x_C)$. We need make an estimate of equation (9), as neither the true f nor the marginal distribution $dP(x_C)$ are known. This is done in equation (10), where $x_{C1}, ..., x_{CN}$ are the different values of x_C observed in the data.

$$\hat{f}_S(x_S, x_{C_i}) = \frac{1}{N}\sum_{i=1}^{N} \hat{f}(x_S, x_{Ci}) \tag{10}$$

For each observation x_{Ci} is kept and x_{Si} is switched with the variable we want to study. Here $\hat{f}_S$ is an estimate of the true model, and is the function we plot (Goldstein et al., 2015).

3.4 Individual conditional expectation plot

Individual conditional expectation (ICE) plot is an extension of the PD plot. The ICE plots the predicted response as a function of feature x_S conditional on an observed x_C, rather than plotting the average partial effect. This results in N estimated conditional expectation curves. A PD plot is the average of all the ICE plots. ICE plots can give interesting plots and indications that are not visible through PD plot.

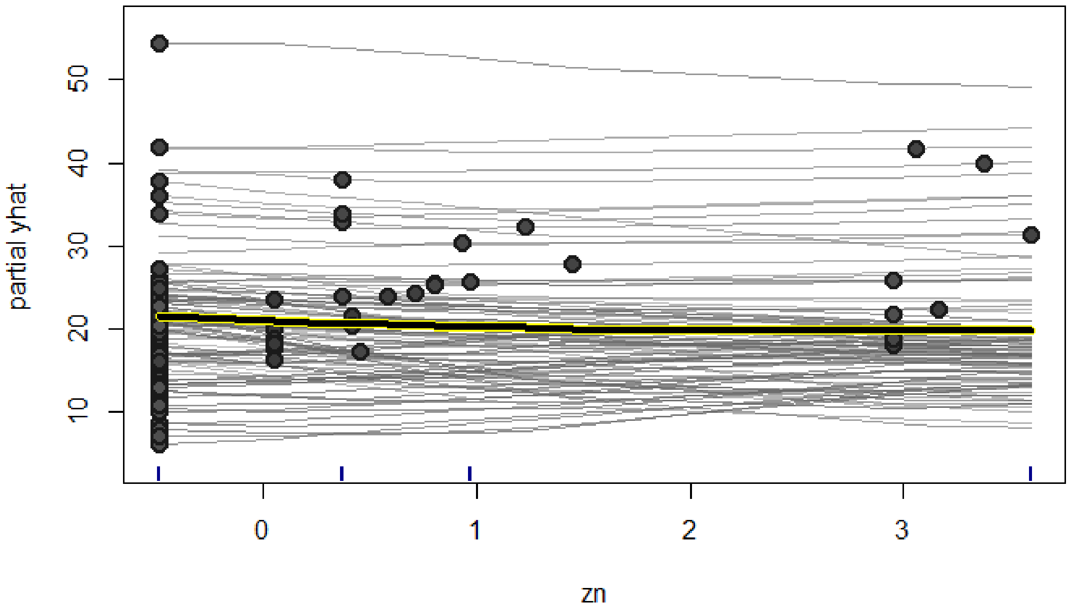


Figure 5: An example of an ICE plot.

Figure 5 is an example of an ICE plot. The plot shows how the proportion of residential land zoned for lots over $25,000$ sq.ft (Zn) affects the output of the model used in the Section 4. The bold line with yellow border around it is the PD plot, while all the other lines are ICE plots. Here we see that the PD plot is the average of all the individual ICE plots. From the plot we can see that the changes in this variable almost have no influence on the prediction of the model.

4 Data analysis

In this section we will study how to use the methods presented in this thesis on a real life problem using a popular benchmark data set.

4.1 Boston Housing

We will use the Boston Housing data set to build a feed-forward neural network as well as show PD and ICE plots. The Boston Housing data set contains 13 feature variables and 506 observations and is collected in 1978. Description is from *The Boston Housing Dataset* 1996, and the data set is in the MASS package in R (Venables and Ripley, 2002) . The following are the features presented in the data set:

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town.
- CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per \$10,000
- PTRATIO - pupil-teacher ratio by town
- B - $1000(Bk - 0.63)^2$ where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in \$1000's

The goal is to predict the median value of owner-occupied homes in \$1000's.

4.2 Descriptive statistics

Understanding the data is important when using machine learning models. Information about the data is the basis for both the decision of which model to use and how to use the data in the model. It is also important to address missing data. When the data set is missing data points there are many different methods to handle this. In the Boston Housing data set there are no missing data, so these methods will not be discussed in this thesis.

We can look at the data by taking a summary in R. This way we can look at the minimum (min.), median, mean, maximum (max.) and the standard deviation for each feature. From this, we see that some of the features have a higher variance than others and a wider range. All these implications can affect the model and we need to handle the data.

	crim	zn	indus	nox	black	lstat
Min	0.01	0.00	0.46	0.39	0.32	1.73
Median :	0.26	0.00	9.69	0.54	391.44	11.36
Mean :	3.61	11.36	11.14	0.55	356.67	12.6
Max.:	88.98	100.00	27.74	0.87	396.90	37.97
Sd.:	8.60	23.32	6.86	0.12	91.29	7.14

	rm	age	dis	rad	tax	ptratio
Min.:	3.56	2.90	1.13	1.00	187.0	12.60
Median :	6.21	77.50	3.21	5.00	330.0	19.05
Mean :	6.29	68.57	3.80	9.55	408.2	18.46
Max.:	8.78	100.00	12.13	24.00	711.0	22.00
Sd.:	0.70	28.15	2.11	8.70	168.54	2.16

Table 1: The minimum, median, mean, maximum and standard deviation for every feature in the data set.

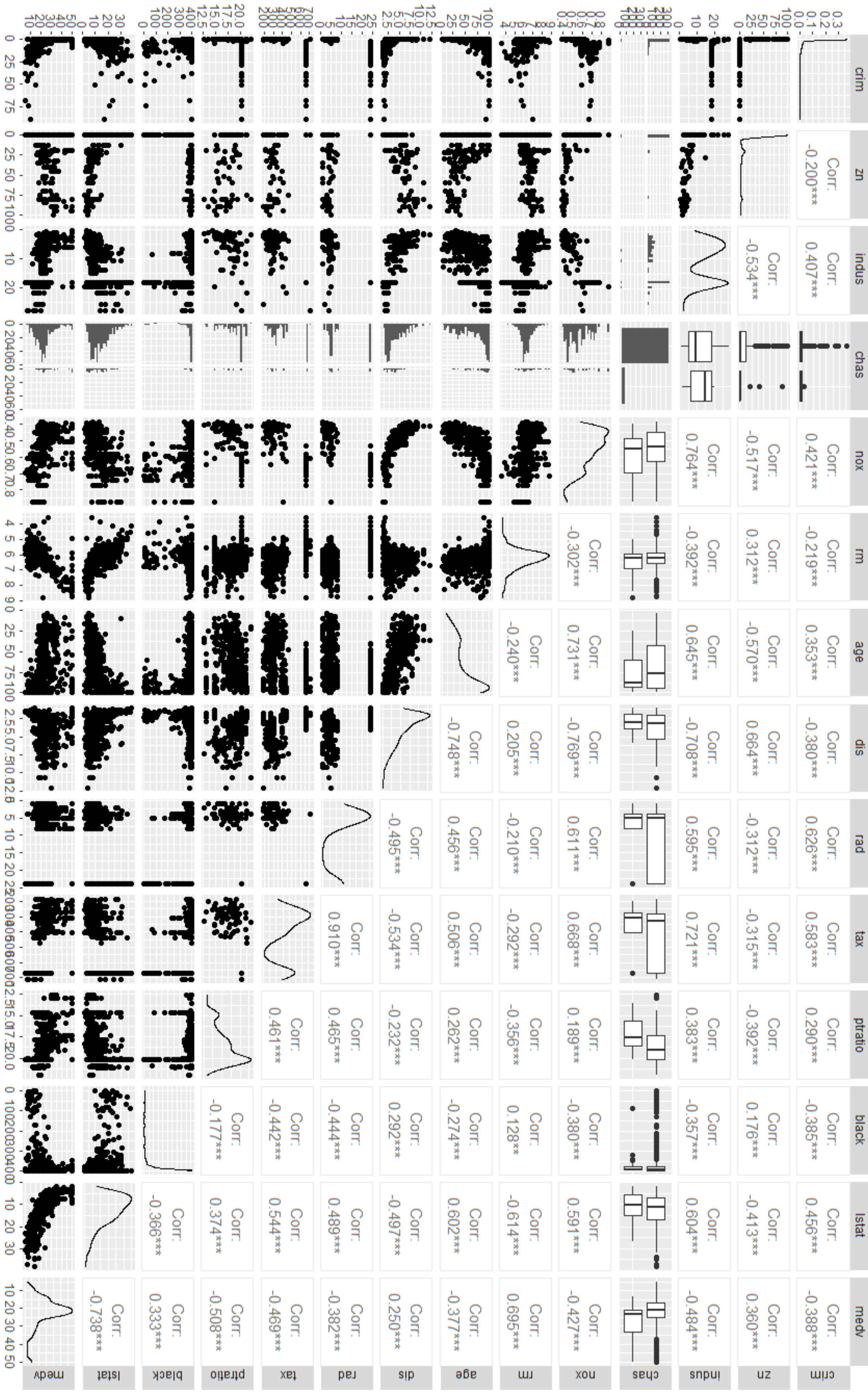


Figure 6: Pairs plot of the features in the Boston Housing set.

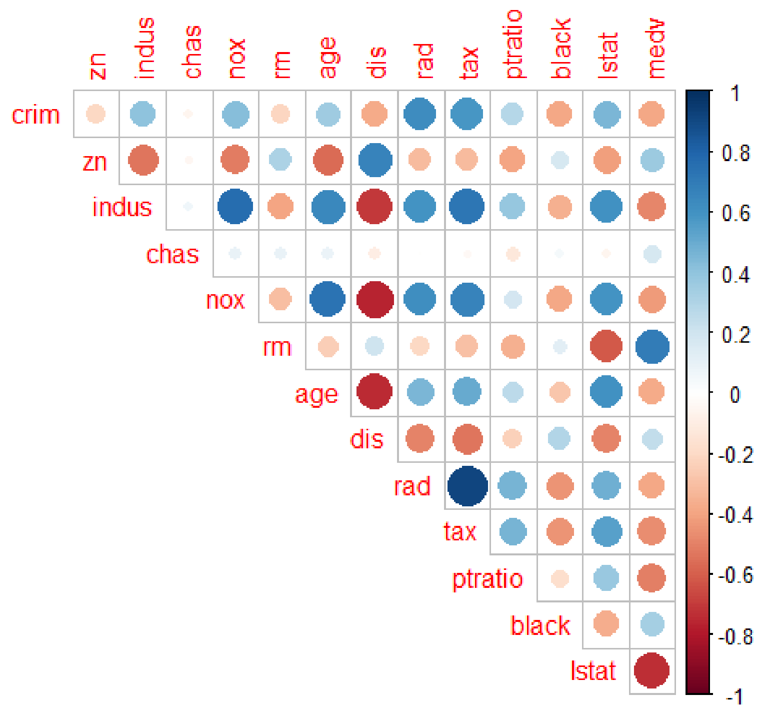


Figure 7: Chart that shows the correlation of the features in the Boston Housing set.

Figure 7 shows the correlation between the features in the Boston Housing set. From the plot, we are most interested in the correlation between the median value of owner-occupied homes in \$1000's (MEDV) and the other features. Here two correlations stand out. The negative correlation with the % lower status of the population(LSTAT) and the positive correlation with the average number of rooms per dwelling(RM). This correlation is also visible in the bottom row of the pairs plot, Figure 6.

Standardization

Depending on the method to be used, data standardization is an important aspect. For models using the black box methods random forest and gradient boosting standardization is less important, especially when the sample size is large (Shanker et al., 1996).

Standardization is used when there is a large difference in the range of the features, and when the features use different measurement units. If a feature has a wide range it may dominant the model, standardization makes sure this does not happen. For Boston Housing we have features measured in dollar, square feet, and other measurement units as well as wide ranges for certain features, so it is important to standardize the data set.

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i \tag{11}$$

$$S = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_i - \bar{X})^2} \tag{12}$$

Standardization consists of first centering data to have a mean 0 and then scaling to have a standard deviation of 1. The mean is calculated by equation (11) and the standard deviation by equation (12). Each value is then subtracted by the mean, and divided by the standard deviation to get the standardization. In the model, we use the standardization within the cross-validation. Doing it within the cross-validation ensures a better representative for the training data. In the cross-validation, you both standardizes the partial training data and the validation data separately.

4.3 Fitting an artificial feed-forward neural network with Keras

4.3.1 Training and test data

The first thing we do when building a model is to divide the data set into a training data set and test data set. A common split is to have 80% of the data being used as training data and the remaining 20% be test data. This makes sure that we have enough training data to run a k-fold cross-validation for model selection in the training data and enough to get a representative result from the test data. We calculate the feature means and standard deviation on the training data and use that to standardize the test data now. The reason we want to standardize on the mean and standard deviation of the training data is that we expect observations in the future to not come in a data set and that the training data is a better representation for the mean and the standard deviation of the data set.
Next is to prepare for the k-fold cross-validation by splitting the training data into k different parts. Practically this is done by giving each observation an index between 1 and k at random, making sure that each split has more or less the same number of observations. We chose to go with a 5-fold cross-validation, this ensures that neither the validation set nor the training set is too small.

4.3.2 Model setup

Now we start the set up the architecture of the model. We chose the R package Keras (Chollet and J. J. Allaire, 2018) to build the model. To get a more optimal model one could use grid search to get the best possible architecture. This is not the topic of this thesis and we went for a small model containing two hidden layers with 16 nodes in the first and 8 nodes in the second one. This is a regression problem and we use the squared loss of Section 2.4. The ReLU, discussed in 2.3, is used as the activation function.

4.3.3 Hyperparameter tuning: early stopping

The only hyperparameter we calculate in this thesis is the number of epochs we use to train the model, this method is called early stopping and was discussed in Section 2.5. Before starting we assign an upper limit on how many epochs to run through. We choose this to $n = 500$. After this, we used the 5-fold cross-validation on the model. Training on four parts and validating on one part. For every new validation set, we standardize the partial training set and the validation set. The validation set is standardized by the mean and standard deviation of the partial training set, as this is a larger set and is expected to be a better estimate for the mean and standard deviation of the data set. We train the model on each cross-validation set n times, after each epoch saving the score on the training set and the validation set. When the model has been trained on all 5 different sets we take the mean result for epoch and calculate the standard deviance and plot these in a graph.
Figure 8[1] shows how the loss on the validation set is affected by the number of epochs. Here we see that it starts by decreasing before it eventually starts to increase. The increase in the loss of the validation set suggests that the model starts to overfit. The optimal number of epochs to use is suggested by the lowest loss on the validation test, which in our case was 72 epochs.

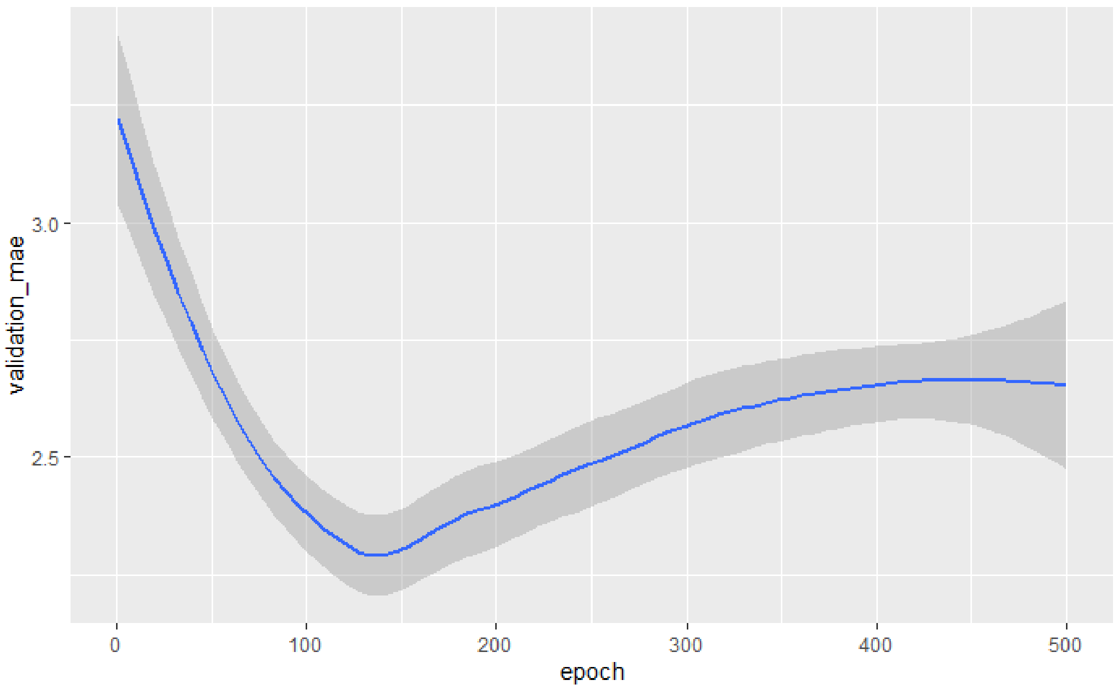


Figure 8: How the mean absolute error on the validation is affected by the number of epochs.

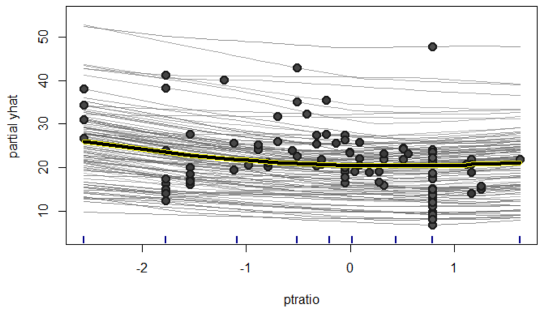
[1]Figure 8 is created using the geom_smooth() in ggplot2 and does not display the precise value of the validation mae, but how they are predicted to look like (Wickham, 2016).

4.3.4 Final optimization

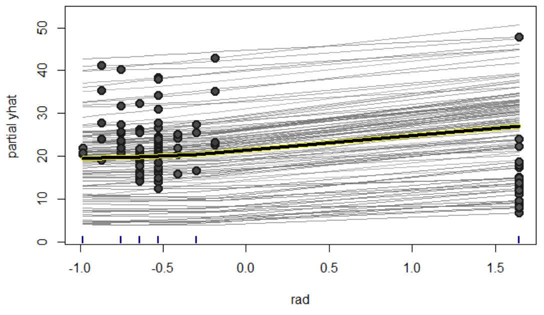
After we are finished looking at the data, finding the architecture and the hyper-parameters of the model, we train our final model. The final model is trained on all available training data in 72 epochs. The final model resulted in a loss of 15.76 and a mean absolute error of 2.37 on the test data.

4.4 ICE and PD plot

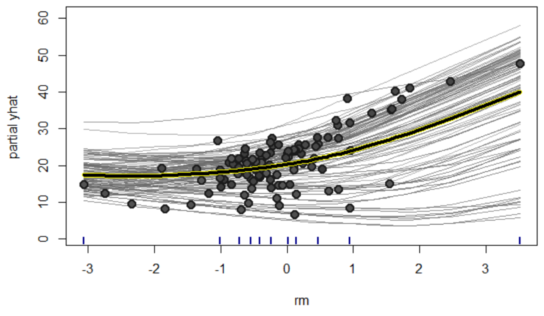
We now present the PD plot and ICE plots for all the features in Boston Housing set fitted by the artificial neural network. As in Section 3.4 the line with the yellow border is the PD plot and all the others are the ICE plots.
A interested thing to notice from the plots in Figure 10, 9 and 5 is that most of the features looks linear. This may indicate that we can use linear regression to get similar prediction results as for the artificial neural network.
The package used to make the ICE plots(Goldstein et al., 2015) did not handle predictions from Keras automatically. Therefore, we had to modify parts of the code, see Appendix A.5.



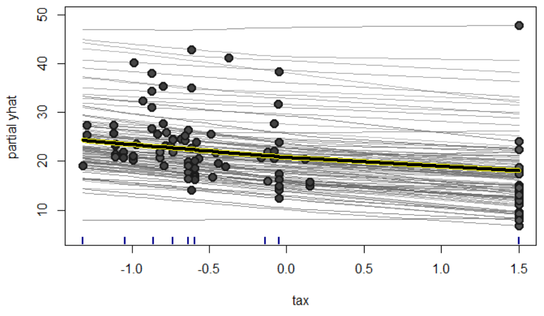
ptratio



rad

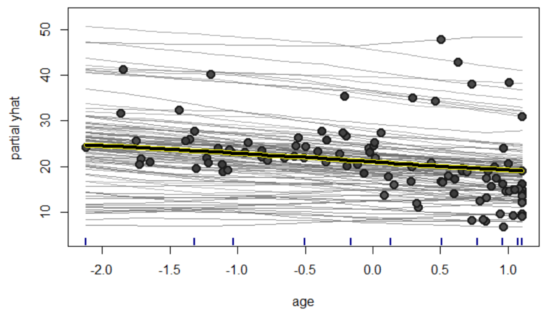


rm

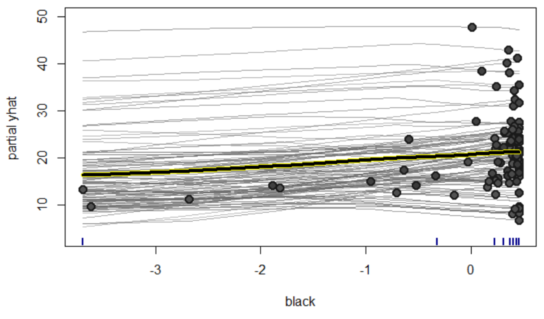


tax

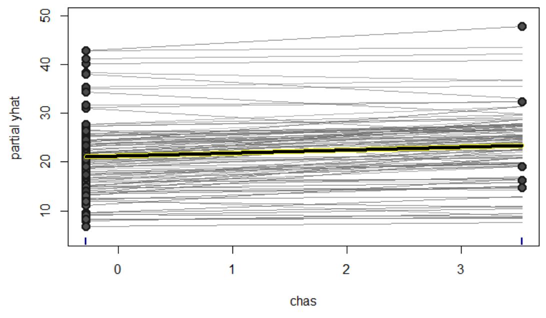
Figure 9: ICE and PD plots of 4 out of 13 features.



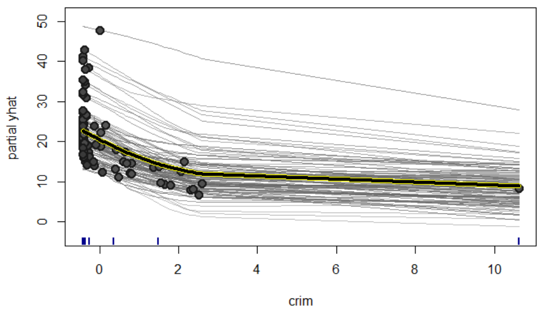
age



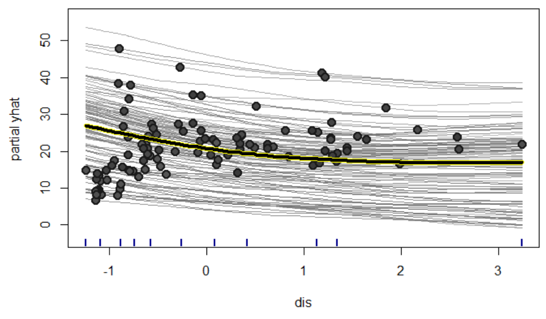
black



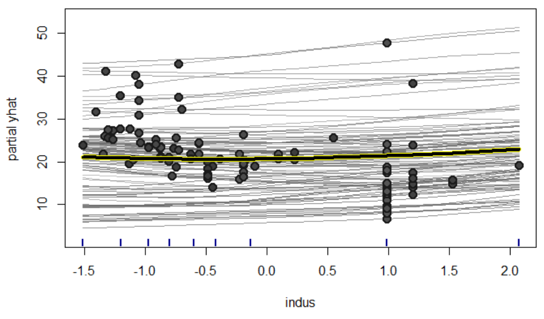
chas



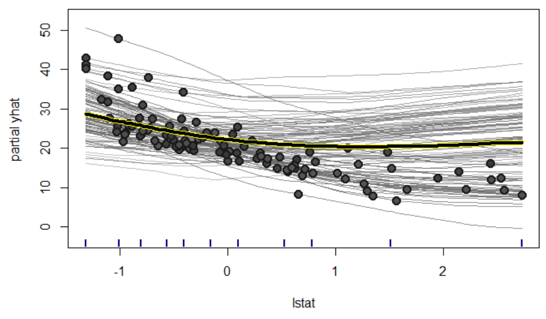
crim



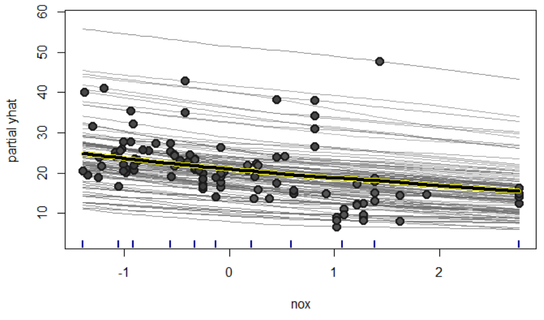
dis



indus



istat



nox

Figure 10: ICE and PD plots of 8 out of the 13 features.

5 Discussion and conclusion

A feed-forward neural network can be used to fit complex data set, using non-linear functions, but does not seem to be very accurate on the Boston Housing data set, considering the mean value of the houses in the data set is $\$22,532$ and the mean absolute error for the model is $\$2,370$. There may be several reasons for this, the first one being a sub-optimal model. The final model is just a regular feed-forward with very little hyperparameter calculations around it, so there may be a model that is more fit for this set. The second reason can be the randomness of the pricing. If the price is determined by a person a lot more features may have been used to determine the value of a house. A third reason is the size of the data set. The Boston Housing set only has 506 observation. From Figure 6 we can see the variable we want to predict MEDV is unevenly spread, giving the model few training observation for the houses that are not valued around $\$20,000$.

Looking at the ICE plots it looks like most of the features are more or less linear. The reason this is interesting is that artificial neural networks are a lot more expensive to build compared to regular linear regression. So if a linear regression model performs just as well as the feed-forward model, it is a lot more cost-efficient to use the linear regression model. We test this out by making a linear regression model and looking at the results of the loss and mean absolute error on the test data. The linear regression model ended up with a loss of $\$478,244,000$ and a mean absolute error of $\$543,000$ on the test data. This implies that the model is much more complex than the ICE and PD plots suggest and that a linear model performs much worse on the data. So even though most of the feature seems to have a linear correlation with the prediction, we can not use a linear regression model to yield the same results.

If we wanted to expand on this thesis it would be around optimization of the hyper-parameters in the model. We would also look at more local explainable AI methods to better understand and explain the model.

References

2018 reform of EU data protection rules (25th May 2018). European Commission. URL: https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf (visited on 17th June 2019).

Allaire, JJ and François Chollet (2021). *keras: R Interface to 'Keras'*. R package version 2.4.0. URL: https://CRAN.R-project.org/package=keras.

Chollet, François and J. J. Allaire (2018). *Deep learning with R*. Manning Press.

Dahl, David B. et al. (2019). *xtable: Export Tables to LaTeX or HTML*. R package version 1.8-4. URL: https://CRAN.R-project.org/package=xtable.

Goldstein, Alex et al. (2015). 'Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation'. In: *Journal of Computational and Graphical Statistics* 24.1, pp. 44–65. DOI: 10.1080/10618600.2014.907095.

Goodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep learning*. MIT Press.

Molnar, Christoph (2019). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable.* https://christophm.github.io/interpretable-ml-book/.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Schloerke, Barret et al. (2021). *GGally: Extension to 'ggplot2'*. R package version 2.1.1. URL: https://CRAN.R-project.org/package=GGally.

Shanker, M., M.Y. Hu and M.S. Hung (1996). 'Effect of data standardization on neural network training'. In: *Omega* 24.4, pp. 385–397. ISSN: 0305-0483. DOI: https://doi.org/10.1016/0305-0483(96)00010-2. URL: https://www.sciencedirect.com/science/article/pii/0305048396000102.

The Boston Housing Dataset (10th Oct. 1996). URL: https://www.cs.toronto.edu/~delve/data/boston/bostonDetail.html (visited on 13th May 2021).

Venables, W. N. and B. D. Ripley (2002). *Modern Applied Statistics with S*. Fourth edition. ISBN 0-387-95457-0. New York: Springer. URL: https://www.stats.ox.ac.uk/pub/MASS4/.

Wickham, Hadley (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

A R-code

A.1 Dividing the data

```
set.seed(1)
train = sample(1:nrow(Boston), 0.8*nrow(Boston))
colnames(Boston)
head(Boston)
train_data=Boston[train,-14]
train_targets=Boston[train,14]
test_data=Boston[-train,-14]
test_targets=Boston[-train,14]
head(test_data)


k <- 5
indices <- sample(1:nrow(org_train))
folds <- cut(indices, breaks = k    , labels = FALSE)
```

A.2 Building the model

```
build_model <- function() {
    model <- keras_model_sequential() %>%
        layer_dense(units = 16, activation = "relu",
            input_shape = dim(train_data)[[2]]) %>%
        layer_dense(units = 8, activation = "relu") %>%
        layer_dense(units = 1)
    model %>% compile(
        optimizer = "rmsprop",
        loss = "mse",
        metrics = c("mae")
    )
}
```

A.3 Select early stopping

```
num_epochs <- 500
all_mae_histories <- NULL
for (i in 1:k) {
  cat("processing fold #", i, "\n")

  val_indices <- which(folds == i, arr.ind = TRUE)
  val_data <- train_data[val_indices,]
  val_targets <- train_targets[val_indices]
  partial_train_data <- train_data[-val_indices ,]
  partial_train_targets <- train_targets[-val_indices]

  model <- build_model()

  history <- model %>% fit(
    partial_train_data, partial_train_targets,
    validation_data = list(val_data, val_targets),
    epochs = num_epochs, batch_size = 1, verbose = 0
  )
  mae_history <- history$metrics$val_mae
  all_mae_histories <- rbind(all_mae_histories , mae_history)
}
dput(all_mae_histories ,"all_mae_histories.dd")

all_mae_histories=dget("all_mae_histories.dd")
average_mae_history <- data.frame(
  epoch = seq(1:ncol(all_mae_histories)),
  validation_mae = apply(all_mae_histories , 2, mean)
)

ggplot(average_mae_history , aes(x = epoch, y = validation_mae))
    + geom_line()
ggplot(average_mae_history , aes(x = epoch, y = validation_mae))
    + geom_smooth()
```

A.4 Train final model

```
thisepochs=which.min(average_mae_history$validation_mae)
model <- build_model()
model %>% fit(train_data, train_targets,
            epochs = thisepochs, batch_size = 1, verbose = 0)
result <- model %>% evaluate(test_data, test_targets)
```

A.5 Make ICE and PD plots

```
icekerasregression=function(object ,X, predictor ,y)
{
  N = nrow(X)
  xj = X[, predictor]
  grid_pts = sort(X[, predictor])
  grid_pts = unique(grid_pts)
  num_unique_pts = length(grid_pts)
  ice_curves = matrix(NA, nrow = nrow(X), ncol = length(grid_pts))
  colnames(ice_curves) = round(grid_pts ,2)
  xvec_temp = X[, predictor]
  for (t in 1:length(grid_pts)) {
          X[, predictor] = grid_pts[t]
          out=object %>% predict(as.matrix(X))
          ice_curves[, t] =out
  }
  X[, predictor] = xvec_temp

  actual_prediction = object%>% predict(as.matrix(X))

  xlab = colnames(X)[predictor]
    range_y = NULL
  sd_y = NULL
  if (!missing(y)) {
      range_y = max(y) - min(y)
      sd_y = sd(y)
  }
  pdp = apply(ice_curves, 2, mean)


   ice_obj = list(ice_curves = ice_curves, gridpts = grid_pts,
        predictor = predictor, xj = xj,
        actual_prediction = actual_prediction,
        logodds = FALSE, probit = FALSE, xlab = xlab,
        nominal_axis = FALSE,
        range_y = range_y, sd_y = sd_y, Xice = X, pdp = pdp,
        indices_to_build = NULL, frac_to_build = 1,
        predictfcn = NULL)
    class(ice_obj) = "ice"
    invisible(ice_obj)
}

kice=icekerasregression(object=model,X=as.data.frame(train_data),
    predictor=6)
plot(kice)

par(ask=TRUE)
for (k in 1:13)
{
  this=icekerasregression(object=model,X=as.data.frame(train_data),
    predictor=k)
  assign(paste("kice",k,sep=""),this)
  plot(this,centered=FALSE,xlab=colnames(train_data)[k],
    point_labels_size=0.5)
}

for (k in 1:13)
{
  this=icekerasregression(object=model,X=as.data.frame(test_data),
    predictor=k)
  assign(paste("kicetest",k,sep=""),this)
  plot(this,centered=FALSE,xlab=colnames(test_data)[k],
    point_labels_size=0.5)
}
```