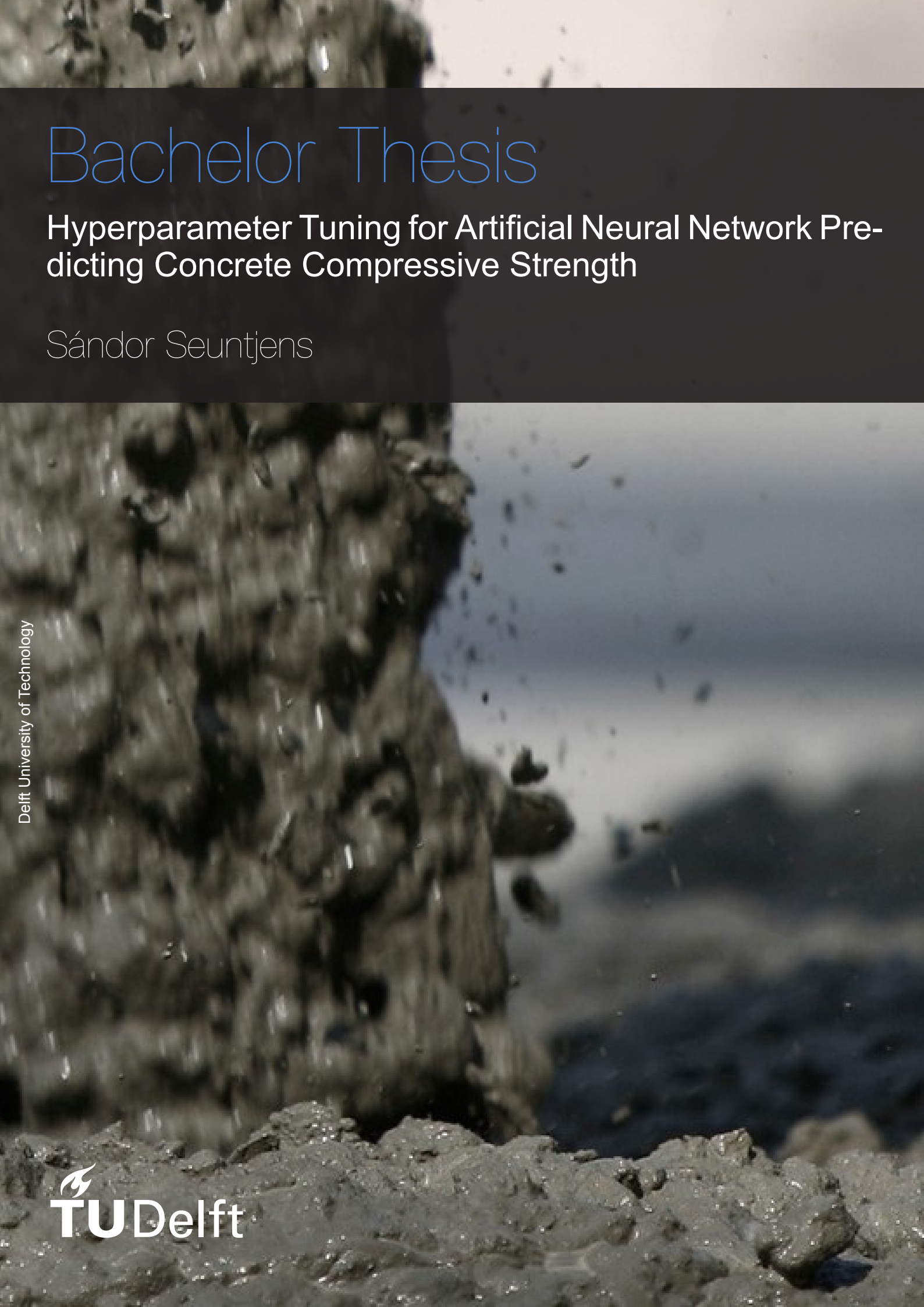


Bachelor Thesis

Hyperparameter Tuning for Artificial Neural Network Predicting Concrete Compressive Strength

Sándor Seuntjens

Delft University of Technology

TU Delft

Hyperparameter Tuning for Artificial Neural Network Predicting Concrete Compressive Strength

by

Sándor Seuntjens

Student Number: 4353609

Supervisors: Zhi Wan
Branko Šavija
Institution: Delft University of Technology
Faculty: Civil Engineering and Geosciences
Thesis Duration: April, 2021 - June, 2021

Cover Image: Pouring of Concrete - https://www.mc-bauchemie.com/market-segments/concrete-industry/

TU Delft

Preface

This thesis is written as partial fulfillment for the degree of Bachelor of Science in the Faculty of Civil Engineering and Geosciences at the Delft University of Technology in the Netherlands.

During my bachelor I did my minor in computer science. Later, Branko Šavija proposed a thesis topic that would combine civil engineering and computer science. I gladly accepted the thesis subject.

If, while reading this report, you the reader have questions or suggestions, don't hesitate to send me an e-mail at s.h.seuntjens@student.tudelft.nl.

The Jupyter Notebooks used in this thesis can be downloaded from my GitHub page: `https://github.com/sundurke`

I want to specifically thank Branko and Zhi for being my supervisors. They helped me to learn many new things in the field of machine learning and particularly deep learning. They taught me how to have meetings and work with supervisors, and helped me with difficult pieces of code. I also want to thank my friends Ignéczi Marcell and Jeroen Zwanepol for helping me develop deep learning insight and helping me with coding.

Sándor Seuntjens
Delft, June 2021

Figure 1: Pouring concrete on site (GCP, 2018)

Abstract

Concrete compressive strength is the most frequently used and most important mechanical property of concrete. National and international building codes (such as the Eurocode) frequently use the compressive strength for design with concrete.

In some cases, instead of testing the concrete specimens under compressive loading in laboratories, predicting the compressive strength using machine learning predictions could be a good alternative. The ingredients making up the concrete mix and the curing age can be used as predictors for the compressive strenght.

Artificial Neural Networks (ANNs) are machine learning algorithms that have been used since the nineteen sixties and were inspired by the way neurons work in the brain. Previous researches suggest that ANNs have great potential to predict concrete compressive strength.

In this research, an Artificial Neural Network was set up using the Keras framework and was trained with concrete composition data consisting of examples of concrete recipes and their respective concrete compressive strenghts. Then, three hyperparameter optimization methods (for loop, grid search and Bayesian optimization) were implemented in several runs. The resulting hyperparameters were used to create ANNs. Afterwards, the three methods were compared with respect to several metrics (R-squared score, root mean square error and running time) to see which one is the relatively best method to provide hyperparameters for the predefined ANN that learns from concrete composition data.

The best runs of the three hyperparameter optimization methods show similar goodness-of-fit (with negligible difference). Among these best runs, grid search has the shortest running time. Bayesian optimization provides the highest R-squared score, and has root mean square error of 4.45 [MPa] and a reasonable running time of 73 minutes. Therefore, Bayesian optimization is considered the preferable hyperparameter optimization algorithm for the concrete compressive data used as goodness-of-fit is in most cases to be considered the decisive metric for this application. Further research trying runs with larger parameter grids and spaces, or using other tuning methods could result in even better goodness-of-fit results.

Contents

Preface i

Abstract ii

1 Introduction 1
- 1.1 Background information . . . 1
  - 1.1.1 Neural networks . . . 2
  - 1.1.2 Hyperparameters . . . 3
- 1.2 Problem statement . . . 3
- 1.3 Research question . . . 3
- 1.4 Scope . . . 3
- 1.5 Report structure . . . 3

2 Methodology 4
- 2.1 Setup . . . 4
- 2.2 The fixed hyperparameters . . . 5
  - 2.2.1 Number of layers . . . 5
  - 2.2.2 Activation function . . . 6
  - 2.2.3 Loss function . . . 6
  - 2.2.4 Optimization algorithms . . . 6
  - 2.2.5 Number of epochs . . . 6
- 2.3 The Hyperparameters that were optimized . . . 6
  - 2.3.1 Number of neurons . . . 7
  - 2.3.2 Learning rate . . . 7
  - 2.3.3 Batch size . . . 7
- 2.4 Optimization with for loop . . . 7
- 2.5 Optimization with grid search . . . 7
- 2.6 Optimization with Bayesian optimization . . . 8

3 Results and Discussion 9
- 3.1 Results . . . 9
  - 3.1.1 Results for loop . . . 9
  - 3.1.2 Results grid search algorithm . . . 10
  - 3.1.3 Results Bayesian optimization algorithm . . . 12
- 3.2 Discussions . . . 13
  - 3.2.1 Comparing goodness-of-fit of the ANNs . . . 13
  - 3.2.2 Comparing run times . . . 14
  - 3.2.3 Comparing with multiple linear regression (LR) . . . 14

4 Conclusions 16

5 Recommendations 17

1 Introduction

1.1. Background information

Concrete compressive strength is the resistance of the material under compressive forces and is the most frequently used mechanical property when designing with concrete. National or international building codes frequently use the compressive strength parameter. Besides from the four main components (coarse aggregate, fine aggregate, cement and water) there is a wide variety of ingredients used in concrete mixes, called admixtures and additives that change the properties of the concrete or the cement. Also cement supplements can be added to reduce the usage of ordinary portland cement.

Concrete compressive strength is usually obtained through experiments in the laboratory where the concrete specimen is tested under compressive loading. When designing a concrete mix for a specific purpose, it can be expensive and time consuming to test all the mixes of the design iterations in the laboratory. This is where machine learning predictions can be useful as an economical alternative as the ingredients making up concrete and the curing age can be used as predictors for the compressive strenght. Machine learning (ML) models can be used for these predictions. It must be noted that when predicting mixes with new ingredients, a number of laboratory test must first be performed with specimen containing the new ingredient to ensure a reliable training set. Yeh (1998) states: "... the method is not applicable to extrapolation beyond the domain of the data accumulated in the past."

It must be noted, that (ML) models can also further our understanding of the building material and the mixing proportions, as Yeh (1998) states: "The more we know about the concrete composition versus strength relationship, the better we can understand the nature of concrete and how to optimize the concrete mixture".

According to Asteris and Mokos (2020) an advantage of the Artificial Neural Network (ANN) models is that they can be retrained anytime, and thus adapt to new data. No dimensionality reduction is used in this study, so an ANN is a good choice as Wan et al. (2021) writes "Particularly, for original features, the model with highest R-squared is ANN model ..." when comparing an ANN to three other machine learning models (Linear Regression, Support Vector Regression and Extreme Gradient Boosting) using original features and the dataset collected by Yeh (1998).

There have been a number of studies devoted to creating ML models (such as multiple linear regression, ANNs, support vector regressions or extreme gradient boosting) to predict concrete compressive strength, and to compare the predictions. However, there have been no previous studies done to investigate which hyperparemeter tuning method for ANNs is the most advantageous in the context of concrete compressive strength.

The findings could be beneficial to those wanting to know which method to use to optimize the hyperparameters of an ANN to predict concrete compressive strength, however, do not want to try out the various options for the optimization methods under certain conditions. Three optimization methods, which are a for loop, a grid search algorithm and a bayesian optimization algorithm have been tested in this thesis. It is important to understand that this is not an exhaustive research. More optimization methods can be compared, some of which are mentioned in Chapter 5 of this thesis.

1.1.1. Neural networks

Similar to functions, ANNs take input and provide output. In ANNs however, there are hidden layers of ”neurons” between the input and output layers. A generalized visualization of a ANN can be seen in Figure 1.1. When the ANN ”learns”, the weights between the neurons are updated in such a way to reduce the cost function. When this learning process is successful (no under or over fitting), the network will predict an output with high accuracy when given input that it has never seen before.

Some aspects of deep learning will be discussed in the coming chapters, however much of the details on deep learning fall outside of the scope of this Thesis and can be widely found on the internet. For further insight I recommend the machine learning course by Andrew Ng on Coursera (Ng, 2011).

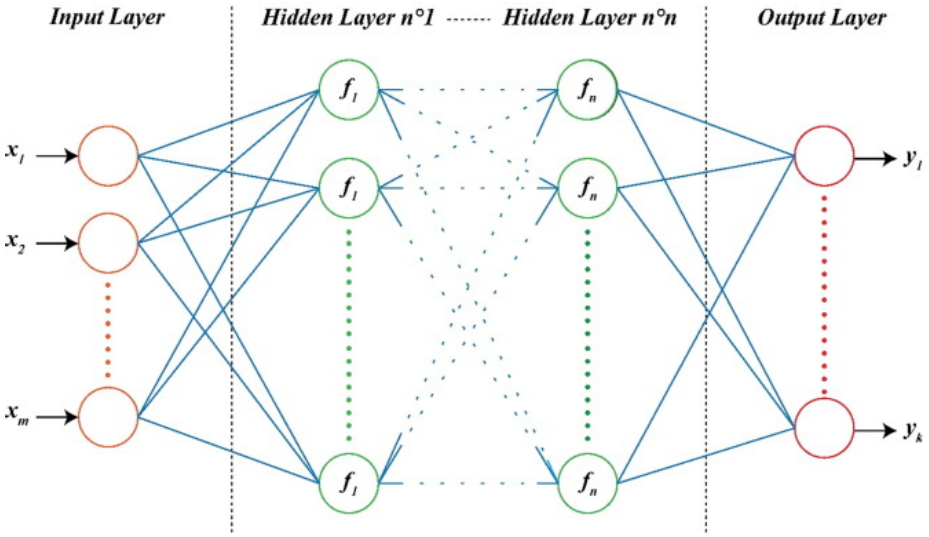


Figure 1.1: General structure of ANN (Ben Chaabene et al., 2020)

Hyperparameter	Approximate sensitivity
Learning rate	High
Optimizer choice	Low
Other optimizer params (e.g., Adam beta1)	Low
Batch size	Low
Weight initialization	Medium
Loss function	High
Model depth	Medium
Layer size	High
Layer params (e.g., kernel size)	Medium
Weight of regularization	Medium
Nonlinearity	Low

Figure 1.2: Approximate sensitivities of some hyperparameters for artificial neural networks (Tobin, 2019)

1.1.2. Hyperparameters

When designing a neural network, the engineer has to choose parameters that decide the architecture of the ANN. These parameters are called hyperparameters. According to Chollet (2017) optimizing hyperparameters has a dramatic effect on the accuracy of the model and is an essential step to get to a state-of-the-art model. It must be said, that it should be done systematically by a machine instead of manually by an engineer to get better results. A list of hyperparameters and the measures of their approximate sensitivity can be seen in Figure 1.2.

1.2. Problem statement

ANN models use hyperparameters that govern the way the network works. These hyperparameters can be chosen by hand, but there exist methods that choose these parameters with the help of hyperparameter optimization algorithms. The next step is to see which hyperparameter optimization method is preferable when specifically predicting concrete compressive strength.

1.3. Research question

The research question is: **"Which hyperparameter optimization method for artificial neural networks predicting concrete compressive strength is preferable?"**. Three subquestions were formulated that help answer the research question. These subquestions are answered in the Chapter 4. They are the following:

1. What is the error for the ANN algorithm with the three respective methods?
2. What are the running times for the optimization methods?
3. What is the overall preferable optimization method?
4. What effect does the epoch number have during optimization?

1.4. Scope

The 3 hyperparameter tuning methods used in this research are:

- For loop
- Grid search
- Bayesian optimization (with Gaussian process)

1.5. Report structure

The following chapters contain:

- Chapter 2 describes the experiment setup, the ANN, the hyperparameters that were chosen beforehand, the hyperparameters that were optimized, and the three optimization methods
- Chapter 3 shows the results and what they imply
- Chapter 4 shows the conclusions that were drawn from the study by answering the subquestions
- Chapter 5 describes what follow up research is recommended

2
Methodology

2.1. Setup

The concrete dataset used in the research was collected by Yeh (1998). This dataset contains 1030 examples of a concrete mix with eight features (independent variables) and the resulting compressive strength (dependent variable). Table 2.1 shows the statistical properties of this dataset. The dataset was devided into a 70-30 split of training and test data.

The scientific programming was done using the python programming language in Jupyter Notebook and PyCharm integrated development environments (IDEs). The ANNs were buildt with the Keras API of Google's open source TensorFlow library.

Because of the prallel architecture of graphical processing units the GPU-enabled TensorFlow using NVIDIA CUDA was used in this research because as Chollet (2017) states "Deep neural networks, consisting mostly of many small matrix multiplications, are also highly parallelizable...".

The for loop was implemented with the built in for loop of Python. The grid search algorithm was performed using scikit-learn's GridSearchCV function. The Bayesian optimization algorithm (with Gaussian process) was performed with the Bayesian optimization tool for python by Nogueira (2014).

The ANNs input layer has eight neurons and the output layer has one neuron. The eight features of the concrete composition are passed to the eight neurons in the input layer. We have one target, the compressive strength which corresponds to the singe output neuron. It is a regression ANN as we are interested in an outcome (concrete compressive strength) on a continuous scale. A visualization of this artificial neural network is shown in Figure 2.1. The ANN is set up with the Mean Absolute Error (MAE) loss function, a validation split of 10%, the Adam optimization algorithm (with default values, except for $epsilon = 10^{-8}$ and `decay=False`), the ReLU activation function and 1000 epochs.

The most preferable hyperparameter optimization method results in the ANN with the best prediction for the concrete compressive strenght for examples of the test set. The R-squared score (also called the coefficient of determination) was chosen as the metric for 'goodness-of-fit'. Equation 2.1 shows a formula for the R-squared score (Ben Chaabene et al., 2020). The run times of the optimization methods also needs to be taken into account when deciding which one is preferable. The Root Mean Square Error (RMSE) with unit [MPa] is also an interesting metric to for this thesis, as if the RMSE is small, no large errors were made in the prediction of the concrete compressive strength, as large outlier errors would inflate the RMSE.

$$R^2 = 1 - \frac{\sum(y_i - y_i')^2}{\sum(y_i - \bar{y})^2} \tag{2.1}$$

y_i = actual value of i-th sample
y_i' = predicted value of i-th sample
$\bar{y}$ = mean value (in this case of compressive strength)

Table 2.1: Statistical properties of the Yeh's dataset

	Cement (kg/m3)	Blast Furnace Slag (kg/m3)	Fly Ash (kg/m3)	Water (kg/m3)	Superplasticizer (kg/m3)	Coarse Aggregate (kg/m3)	Fine Aggregate (kg/m3)	Age (days)	Compressive Strength (MPa)
Mean	281.168	73.896	54.188	181.567	6.205	972.919	773.580	45.662	35.818
Std	104.456	86.237	63.966	21.344	5.971	77.716	80.137	63.139	16.698
Min	102	0	0	121.8	0	801	594	1	2.33
Max	540	359.4	200.1	247	32.2	1145	992.6	365	82.6
Range	358	359.4	200.1	125.2	32.2	344	398.6	364	80.26

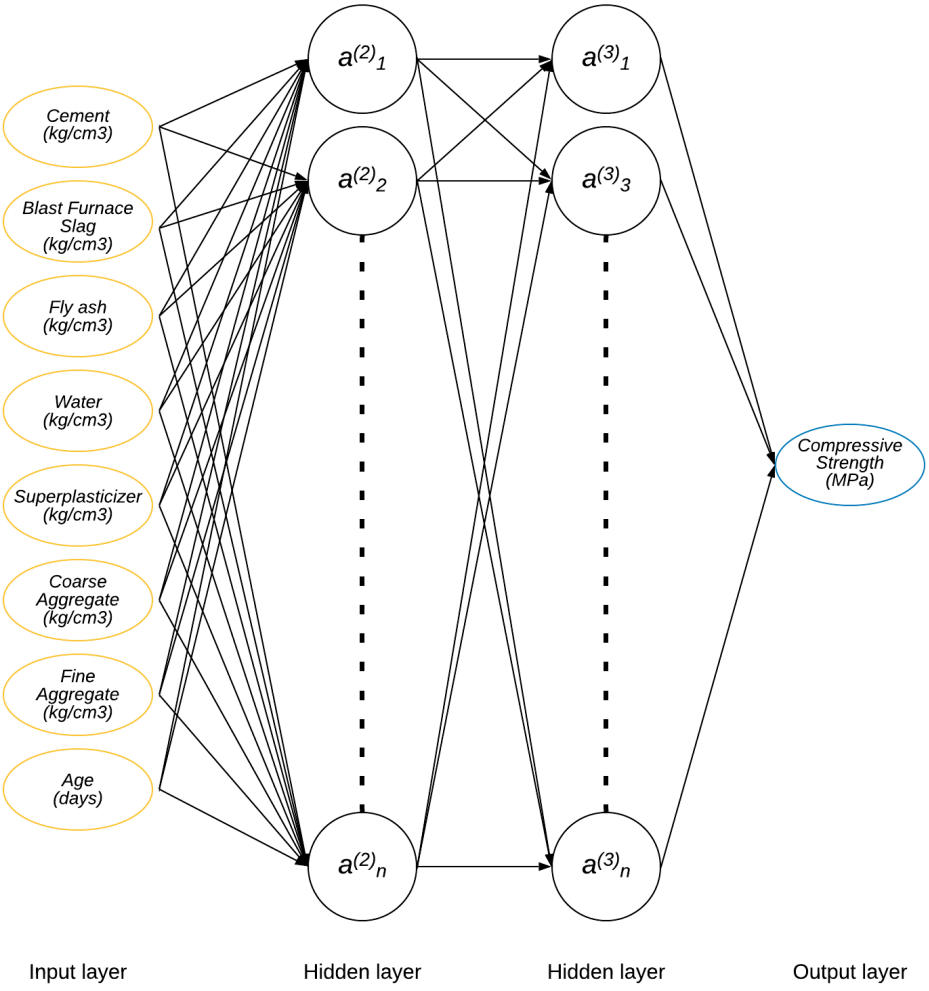


Figure 2.1: Visualization of the ANN used in this research

2.2. The fixed hyperparameters

This section describes which hyperparameters were not tuned, but were instead chosen at the start of the experiment.

2.2.1. Number of layers

In this research the ANN was chosen to have two dense hidden layers similarly to the ANN used by Wan et al. (2021) as the size of the dataset is small. A dense or fully connected layer is a layer in which each neuron takes input from all neurons of the previous layer. Dense layers are the most basic type of hidden layers in deep learning. The more hidden layers, the better fitting capacity the model will have.

However, with each added layer the number of computations grows exponentially. Adding extra layers also increases the likelihood of overfitting when a small training set is used and no extra penalty is carried out, which is the case is this thesis. Besides, a network with only two hidden layers can already take on any nonlinear shape. It is therefore often not preferable to use more than two layers, as the cost of the dramatically increased running time might not pay off.

2.2.2. Activation function

The 'ReLU' (Rectified Linear Unit) activation function was chosen in both hidden layers as it is currently regarded as the activation function that yields the best solutions. Goodfellow et al. (2016) states "In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU". An advantage of ReLU is that it can converge in a short time compared to the Sigmoid or Tanh functions. The equation for ReLU can be seen in equation 2.2 (Liu, 2017).

$$y = max(0, x) \tag{2.2}$$

2.2.3. Loss function

According to Brownlee (2019) "The cost or loss function has an important job in that it must faithfully distill all aspects of the model down into a single number in such a way that improvements in that number are a sign of a better model." The Mean Absolute Error (MAE) is a loss function that is used for regression models and was chosen in this research. The loss function is used in the backward propagation step. The equation for the MAE can be seen in equation 2.3 (Glen, 2020).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - x_i| \tag{2.3}$$

y_i = predicted value of i-th sample
x_i = true value of i-th sample

2.2.4. Optimization algorithms

Optimization algorithms find the values of parameters such that a loss function is at its lowest. The Adam optimizer which stands for adaptive moment estimation "... combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems." (Brownlee, 2021). The Adam algorithm is a popular choice among data scientists. This optimization algorithm is used in the backward propagation step.

2.2.5. Number of epochs

An epoch is one pass through the whole training dataset. In general it takes many epochs for a model to learn enough so that it can have a good enough accuracy for the engineer. Too few epochs may lead to underfitting, which results in poor predictions. It is also important to note that running through too many epochs can cause the model to overfit, which means that it can perfectly (or almost perfectly) predict outcomes from data with which it was trained (which however is not useful), but fails to accurately predict outcomes from data it has not yet seen. A good number of epochs results in a model that does not overfit nor underfit. The number of epochs changes from run-to-run for each optimization method (tuning by hand). This was done to get a better understanding of how the number of epochs influences the results.

2.3. The Hyperparameters that were optimized

The following hyperparameters were tuned with hyperparameter optimization methods:

- Number of neurons in hidden layer one
- Number of neurons in hidden layer two
- Learning rate
- Batch size

The functionality of these hyperparameters is briefly described in the following subsections. The exact parameters tried for each optimization method can be seen in Tables 2.2, 2.3 and 2.4.

2.3.1. Number of neurons

In Figure 2.1 neurons $a_1^{(2)}$ up to $a_n^{(2)}$ make up a hidden layer and are fully interconnected with all neurons in both adjacent layers. The number of neurons in the two hidden layers were tuned with a for loop, the grid search algorithm and the Bayesian optimization algorithm. When tuning with the for loop and with grid search, in order to help speed up the running times, the number of neurons were chosen to be powers of two, to comply with the Boolean logic of the computer.

2.3.2. Learning rate

As the name suggests, the model will learn quicker or slower depending on the learning rate. If the learning rate is chosen too large, the model might converge too quickly and miss an optimum. If it is chosen too small, the model might take a very long time to converge or might even get stuck. For a good model, a learning rate that fits the problem is essential. As can be seen in Figure 1.2 changes in the learning rate have a high impact on the quality of the model. According to Goodfellow et al. (2016) ”The learning rate is perhaps the most important hyperparameter. If you have time to tune only one hyperparameter, tune the learning rate”.

2.3.3. Batch size

Instead of updating the model after each example/sample, the model gets updated after a whole batch of examples is processed. The batch size can be anywhere from one to the number of examples in the training set. The engineer will try different batch sizes to decide which gives the best model.

2.4. Optimization with for loop

A for loop (FL) is a basic programming algorithm that can be called with a built in python function. As the name suggests it loops or iterates through a list that is passed to it.

Lists were created for each of the four hyperparameters. An overview of the lists that were passed to the for loops can be seen in Table 2.2.

The neuron numbers in the two separate hidden layers were aggregated in a nestled for loop. The goodness-of-fit was calculated within the inner loop. Also the learning rate and the batch size were aggregated. The reason not all 4 hyperparameters were tuned together in a four layered nestled for loop, is that the execution time would have been too long. Even with the separated nestled loops, this hypertuning method has the longest running time. The running times per optimization method can be found in subsection 3.2.2. A validation split of 25% was used.

As a clean ANN is created within each inner loop, 1000 epochs are running for every combination. This means 94 x 1000 epochs were run in total.

Cross validation could have been implemented with a piece of python code within the nestled loops; however, it was chosen not to do so because the running time would increase even more.

Four runs were performed. Run 2 proved to have the best combination of hyperparameters.

Table 2.2: Hyperparameter optimization runs performed with for loop

	Number of neurons	Learning rate	Batch size	Epochs	R^2	Time (min)
Run 1	16, 32, 64, 128, 256, 512, 1024	0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1	8, 16, 32, 64, 128	500	0.918	31
Run 2	16, 32, 64, 128, 256, 512, 1024	0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1	8, 16, 32, 64, 128	1000	0.926	65
Run 3	16, 32, 64, 128, 256, 512, 1024	0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1	8, 16, 32, 64, 128	3000	0.924	229

2.5. Optimization with grid search

Grid search (GS) is a traditional way of hyperparameter tuning. A parameter grid is manually defined according to what the engineer thinks are values that could give good results. The GridSearchCV performs an exhaustive search over the specified parameter values. An overview of lists that define the four dimensional grid can be seen in Table 2.3. The scikit learn GridSearchCV class by Pedregosa

et al. (2011) has an input paramerer for the number of cross validations, which allows this optimization method to perform k-fold cross validation.

Four-fold cross validation was used, which corresponds to a changing validation split of 25%.

Four runs were performed. Run 2 proved to have the best combination of hyperparameters.

Table 2.3: Hyperparameter optimization runs performed with grid search

	Number of neurons	Learning rate	Batch size	Epochs	R^2	Time (min)
Run 1	16, 32, 64, 128, 256, 512, 1024	0.0003, 0.001, 0.003, 0.01	8, 16, 32, 64	500	0.912	27
Run 2	16, 32, 64, 128, 256, 512, 1024	0.0001, 0.0003, 0.001, 0.003, 0.01, 0.03	8, 16, 32, 64, 128	500	0.924	59
Run 3	16, 32, 64, 128, 256, 512, 1024	0.0003, 0.001, 0.003, 0.01	8, 16, 32, 64	1000	0.884	30

2.6. Optimization with Bayesian optimization

Bayesian optimizations (BO) is known to find better hyperparameters significantly faster than grid search (for large datasets; however, Yeh's dataset would generally be classified as small). The YouTube video of a University of Toronto lecture by Hinton (2017) explains that this is because for a GS you have many points along an axis where only one parameter is changing and the others are fixed. If the fixed parameters inevitably result in a bad outcome, you have wasted a lot of time looking at the whole axis. The reason why BO is more efficient is the smart nature of the algorithm, instead of looking at every grid point, it uses the previous results to predict regions of the hyperparameter space that might give better results. Usually Bayesian optimization does this using the Gaussian process model that predicts how a set of hyperparameters will do on the validation set and also assigns a variance to this combination of hyperparameters based on whether it is close to a previous prediction or not (high variance for different points, low variance for similar points). This allows it to make educated guesses to where in the parameter space it wants to be looking next.

The parameter space can be found in Table 2.4. Notice that for BO we have a parameter space instead of a parameter grid. This means that hyperparameters can take on any value within the continuous parameter space. Four-fold cross validation was implemented manually, which corresponds to a changing validation split of 25%. The input parameter `n_iter` for the maximize method in the Bayesian Optimization package, was set to 20, wich means that 20 steps of BO were performed.

Four runs were performed. Run 3 proved to have the best combination of hyperparameters.

Table 2.4: Hyperparameter optimization runs performed with Bayesian optimization

	Number of neurons	Learning rate	Batch size	Epochs	R^2	Time (min)
Run 1	8 - 1024	0.0001 - 1	8 - 128	500	0.912	70
Run 2	8 - 1024	0.0001 - 1	8 - 128	1000	0.926	75
Run 3	8 - 2048	0.0001 - 3	8 - 256	1000	0.933	73
Run 4	8 - 4096	0.0001 - 10	8 - 512	1000	0.924	115
Run 5	8 - 1024	0.0001 - 1	8 - 128	3000	0.919	176

3
Results and Discussion

3.1. Results

This chapter shows the results of the research and what they imply.

3.1.1. Results for loop

Figure 3.1 shows the learning curves of the FL run that yielded the highest R-squared score (run 2). With hyperparameters:

- Number of neurons layer 1: 512
- Number of neurons layer 2: 1024
- Learning rate: 0.1
- Batch size: 16
- Epochs: 1000

On the y-axis the loss (MSE), and the x-axis the epochs are drawn. It is not surprising that the curve from the training data converges faster than the ANN is trained on that data. Therefore there is a difference between the loss in training and validation data. This is called the generalization gap. Both training and validation curves have a downwards tendency, this means that the model is not (yet) overfitting at 1000 epochs. Would the training curve continue to go further down and the validation curve would level or even tend upwards, overfitting would be observed.

The 'elbow' of the curve is at around 12 epochs. This implies that the optimum space is found after 15 epochs.

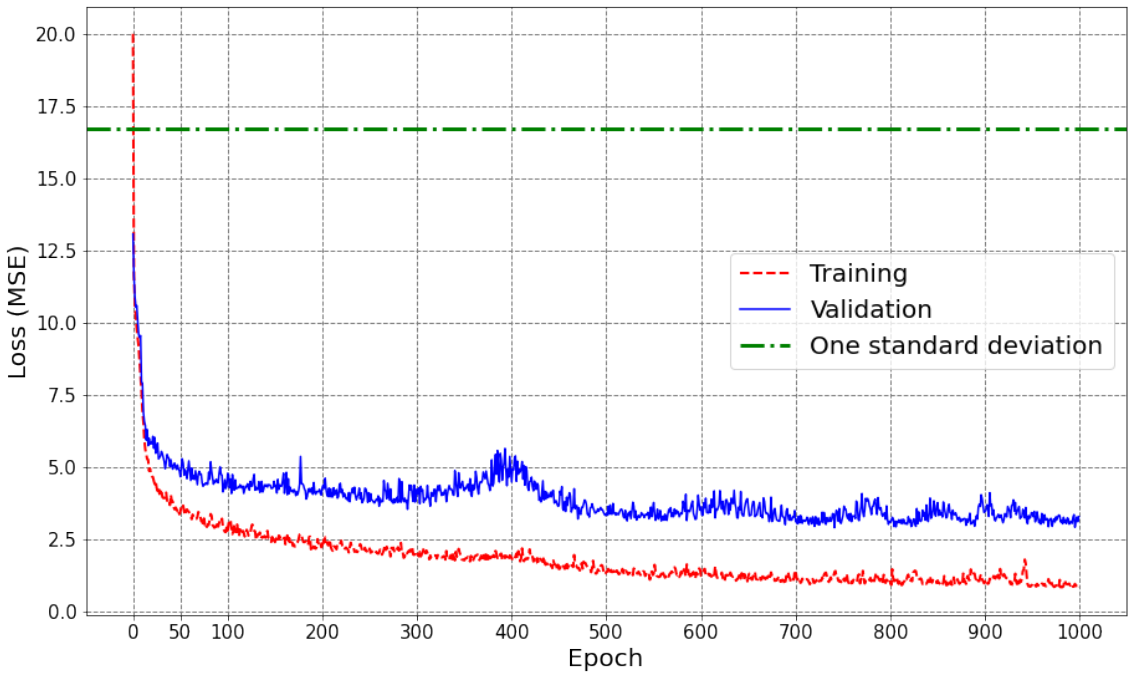


Figure 3.1: Learning curve of model loss (for loop)

Figure 3.2a shows the actual strength and the predicted strength of each example in the **test set**. The closer the data points are to the black diagonal line (where the predicted strength equals the actual strength) the better the prediction. Figure 3.2b shows the actual strength and the predicted strength of each example in the **training set**. The green points are closer to the black line than in Figure 3.2a. This is logical because the ANN was trained on these points (the training set). What we can read from the training plot is that there is still no overfitting, this is because there is still some dispersion of the points and there are some outliers.

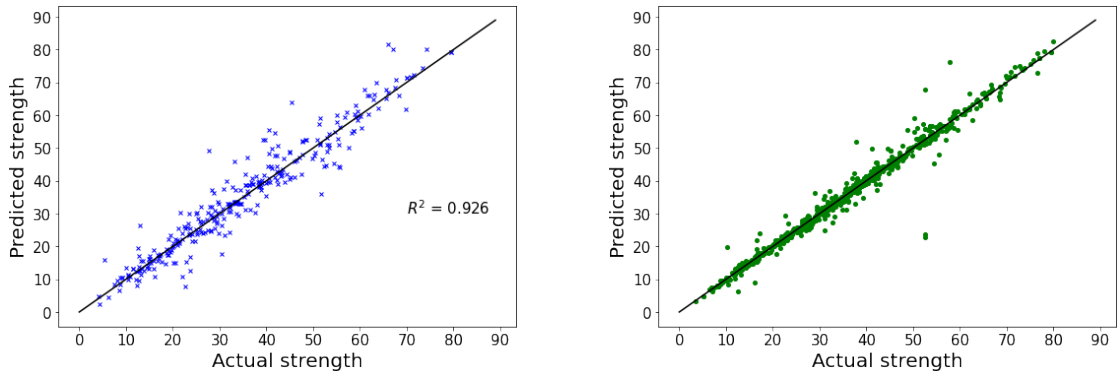


Figure 3.2: Comparison of actual and predicted strength (for loop) for (a) test set (b) training set

3.1.2. Results grid search algorithm

Figure 3.3 shows the learning curves for the GS run which yielded the highest R-squared score (run 2). With hyperparameters:

- Number of neurons layer 1: 512
- Number of neurons layer 2: 64
- Learning rate: 0.01
- Batch size: 8

- Epochs: 500

On the y-axis the loss (MSE), and the x-axis the epochs are drawn. It is not surprising that the curve from the training data converges faster than the ANN is trained on that data.

The 'elbow' of the curve is at around 20 epochs. This implies that the optimum space was found after 20 epochs.

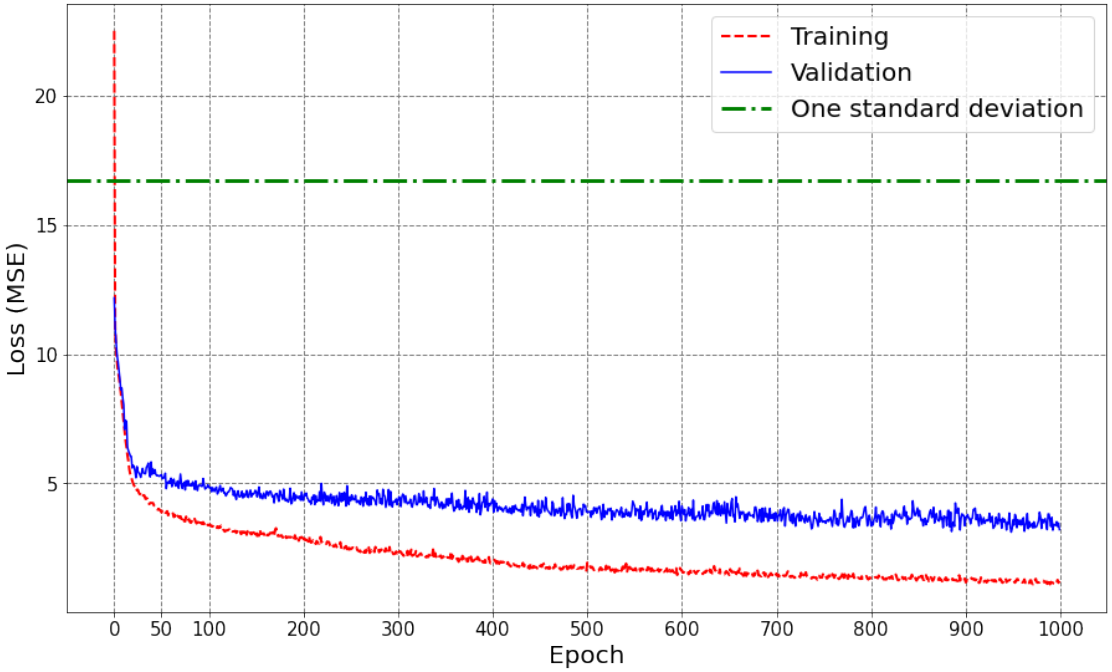
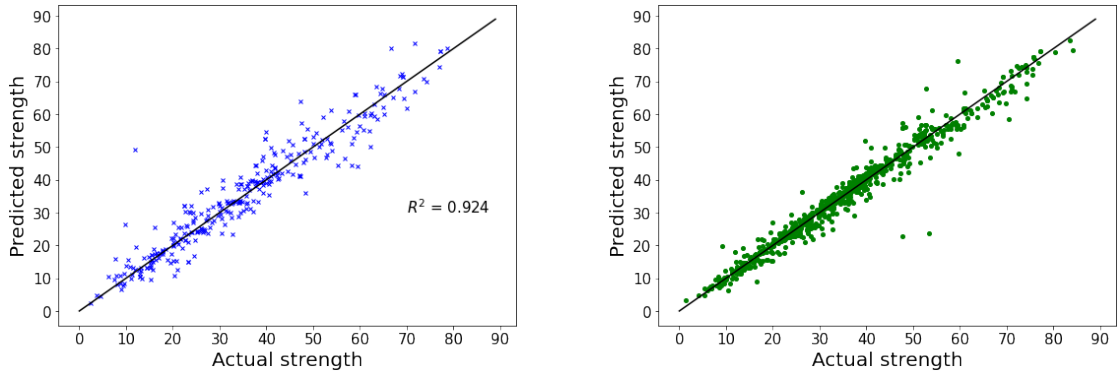


Figure 3.3: Learning curve of model loss (grid search)

Figure 3.4a shows the actual strength and the predicted strength of each example in the **test set**. The closer the data points are to the black diagonal line (where the predicted strength equals the actual strength) the better the prediction. Figure 3.4b shows the actual strength and the predicted strength of each example in the **training set**. The green points are closer to the black line than in Figure 3.4a. This is logical because the ANN was trained on these points (the training set). What we can read from the training plot is that there is still no overfitting, this is because there is still some dispersion of the points and there are some outliers.



(a) (b)

Figure 3.4: Comparison of actual and predicted strength (grid search) for (a) test set (b) training set

3.1.3. Results Bayesian optimization algorithm

Figure 3.5 shows the learning curves for the BO run which yielded the highest R-squared score (run 3). With hyperparameters:

- Number of neurons layer 1: 765
- Number of neurons layer 2: 936
- Learning rate: 0.000186
- Batch size: 64
- Epochs: 1000

The number of neurons are not powers of two, this is because the Bayesian algorithm works with a parameter space instead of a parameter grid. It is possible to code the algorithm in such a way that only powers of two are tried, however this would be contrary to the objective of the parameter space as the algorithm would then run through a parameter grid.

On the y-axis the loss (MSE), and the x-axis the epochs are drawn. It is not surprising that the curve from the training data converges faster than the ANN is trained on that data.

The 'elbow' of the curve is at around 20 epochs. This implies that the optimum space was found at around 20 epochs.

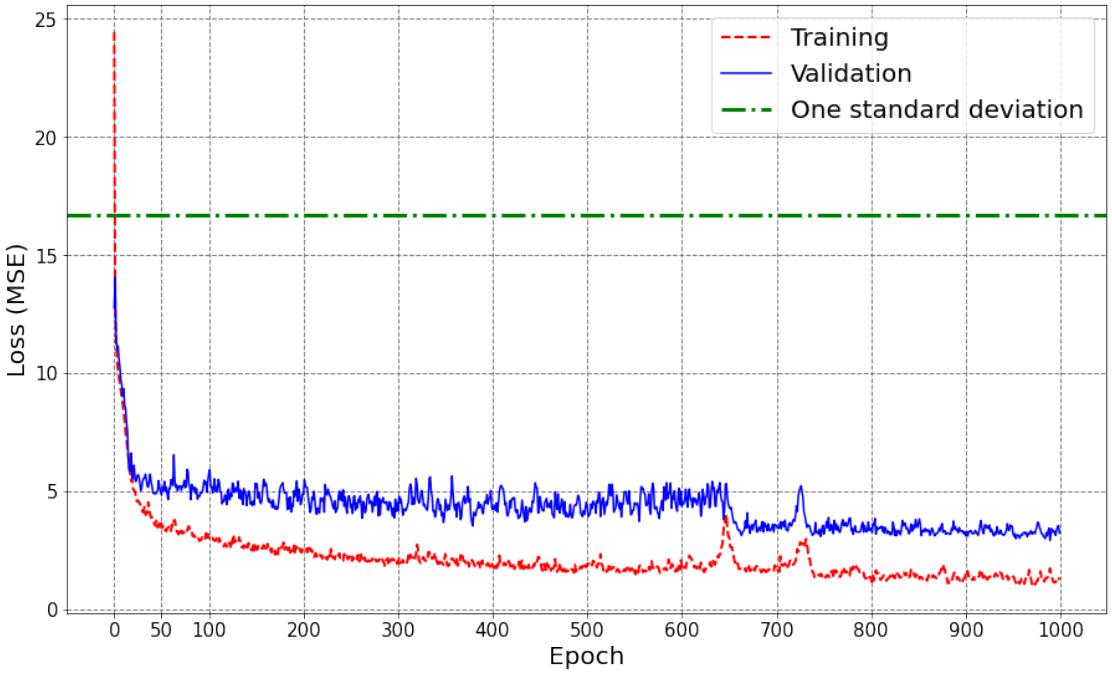


Figure 3.5: Learning curve of model loss (Bayesian optimization)

Figure 3.6a shows the actual strength and the predicted strength of each example in the **test set**. The closer the data points are to the black diagonal line (where the predicted strength equals the actual strength) the better the prediction. Figure 3.6b shows the actual strength and the predicted strength of each example in the **training set**. The green points are closer to the black line than in Figure 3.6a. This is logical because the ANN was trained on these points (the training set). What we can read from the training plot is that there is still no overfitting, this is because there is still some dispersion of the points and there are some outliers.

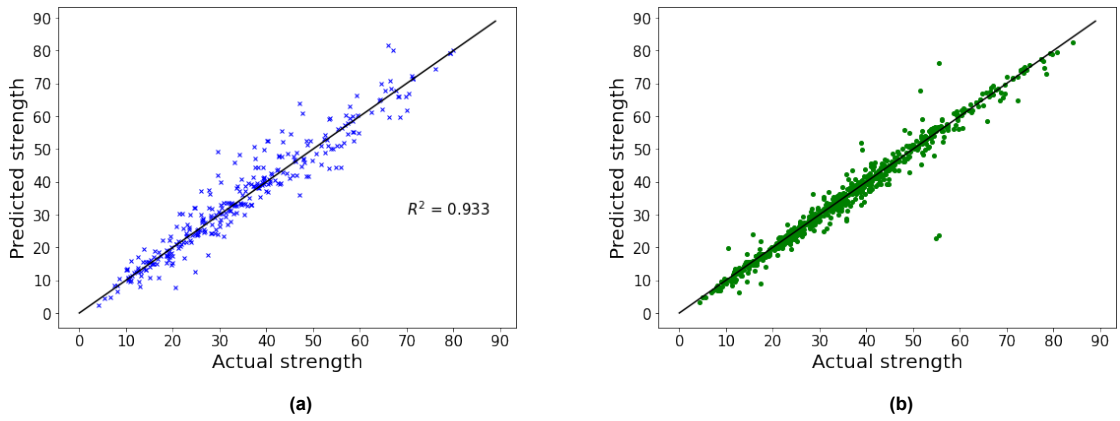


Figure 3.6: Comparison of actual and predicted strength (Bayesian optimization) for (a) test set (b) training set

3.2. Discussions

3.2.1. Comparing goodness-of-fit of the ANNs

The R-squared score, the RMSE with the test set and the RMSE with train set for only the best run (highest R-squared score) of each optimization method are shown in Figures 3.7, 3.8a and 3.8b.

The R-squared scores of each method are very close together (negligible difference). This is also considering that every time the data set is fitted, the scores fluctuate several hundredths up to a few tenths.

It is interesting to see that the RMSEs of the test data are around 4.5-5, because this indicates that the predictions of the concrete compressive strength are approximately going to be mistaken by 4.5-5 MPa, which is a good prediction in the context of structural engineering with concrete.

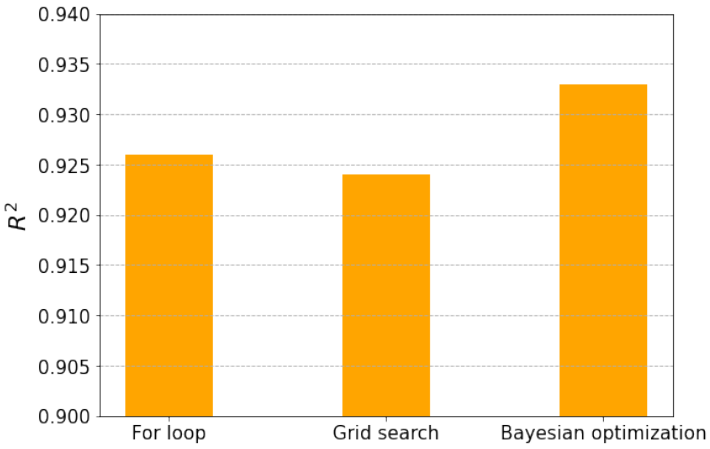


Figure 3.7: Comparison of R-squared scores

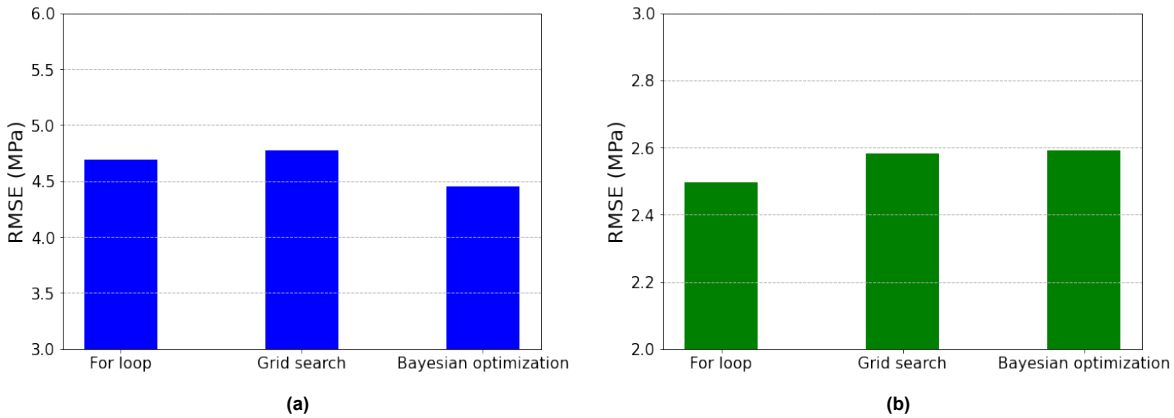


Figure 3.8: Comparison of RMSE (a) testset and (b) trainset

3.2.2. Comparing run times

Next to goodness-of-fit, an other important metric that can differentiate hyperparameter optimization methods is run time. Figure 3.9 shows the observed run times of the only the best run (highes R-squared score) of each algorithm.

When an optimization method produces an ANN with a slightly higher goodness-of-fit, however has a much longer execution time compared to another method, it is important to consider which metric is decisive.

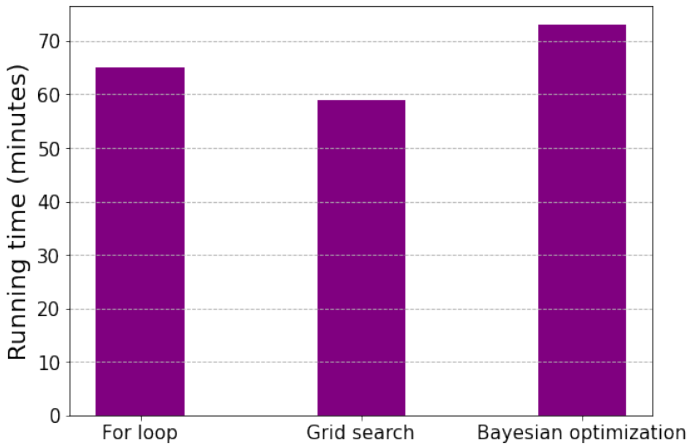


Figure 3.9: Comparison of the run times for the three different optimization methods

3.2.3. Comparing with multiple linear regression (LR)

It can be said that a linear regression is a baseline for machine learning prediction as it is used by most researchers and companies for regression problems. To illustrate the advantages of a neural network over linear regression, a multiple linear regression was built. A multiple linear regression is a linear regression with multiple independent variables (here X_i). A general equation describing a multiple linear regression is shown in Equation 3.1

$$y_p = Intercept + m_1 X_1 + m_2 X_2 + ... + m_n X_n \tag{3.1}$$

where:

$m_{1...n}$ = weights of ingredients
$X_{1...n}$ = values of ingredients (independent variables)
y_p = prediction (dependent variable)
$Intercept$ = y-intercept of regression line

The y-intercept and the weights are calculated by fitting the linear regression model to the data collected by Yeh (1998). Then 3.1 becomes equation 3.2. It is interesting to note that the fourth weight is a negative number and that it is multiplied with the water content in the mix. This is in line with the the fact that concrete decreases in strength as the amount of water in the mix increases.

$$y_p = -8.587 + 0.116X_1 + 0.0967X_2 + 0.0892X_3 + (-0.163)X_4 + 0.242X_5 + 0.0113X_6 + 0.0150X_7 + 0.112X_8 \tag{3.2}$$

The fitting is done using scikit learn's 'LinearRegression'. This methods 'fit' function uses the ordinary least squares method which calculates the best fit by minimizing the sum of the squares of the difference between the predicted values and the actual values.

The obvious benefit of a linear regression is that the fitting process runs in virtually no time (unlike with ANNs). However the R-squared score of the LR prediction was found to be 0.616. This is significantly lower than the R-squared value found with the ANN. The RMSE is 10.70 [MPa]. A visualisation of the actual strength versus the predicted strength can be seen in Figure 3.10. It is clear that the blue crosses each representing a data point are more dispersed and further away from the black diagonal line than in the three figures corresponding to the ANN. This larger distance implies that the fit is less good, which is reflected in the R-squared score. The results obtained in this thesis are in line with the findings of Khademi and Behfarnia (2016) on the comparison of ANNs and multiple linear regression.

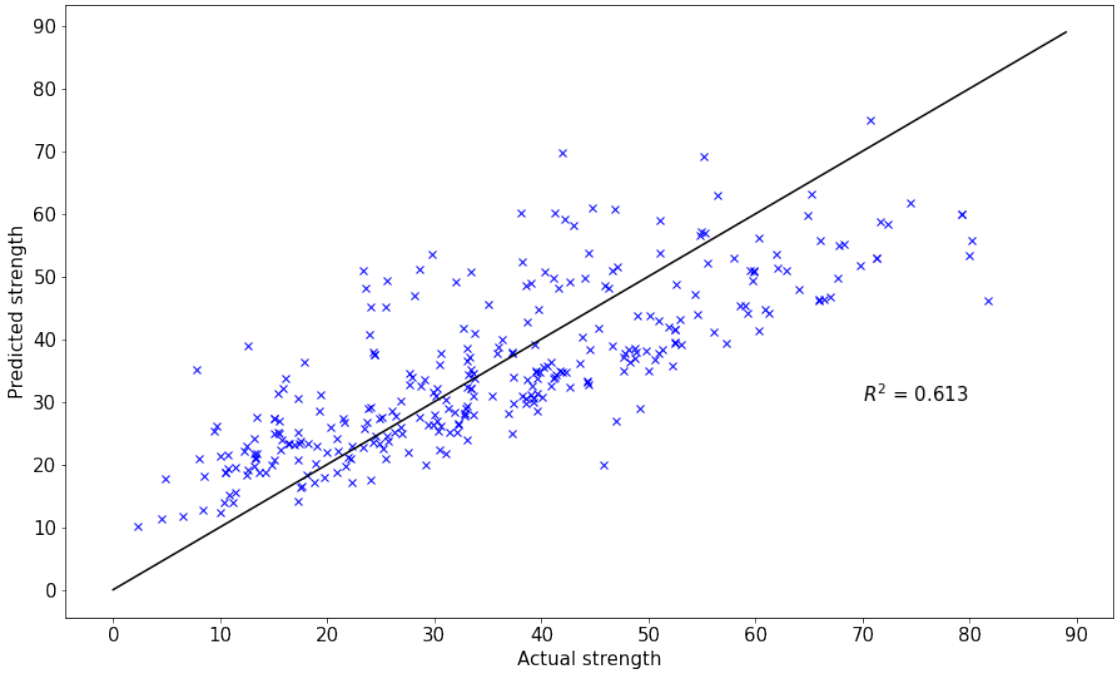


Figure 3.10: Comparison of actual and predicted strengths (linear regression)

4 Conclusions

This thesis is meant to give insight into what hyperparameter optimization method to choose, when using data collected by Yeh (1998) and the ANN described in Chapter 2. A completely fair comparison is not possible because of the difference in nature of the optimizations methods such as whether cross validation is used, how many times a new neural network is made in the algorithm or how large the chosen parameter space is. This means that the properties and the setups of the runs of the optimization methods described in Chapter 2 have to be taken into account when reading the conclusions.

1. The method with the largest R-squared score is BO with 0.933 and a run time of 73 minutes.

2. The method with the shortest run time is GS with 27 minutes and an R-squared score of 0.912.

3. The two overall preferable optimization methods in the scope of this thesis are BO and GS. BO is preferable because it results in the highest goodness-of-fit and the run time is not very long. GS is preferable because it provides a good goodness-of-fit and has the shortest run time. The decision should be made by the engineer if goodness-of-fit or short run time is more important. Goodness-of-fit should in most cases be decisive as the hyperparameter tuning does not have to be run often. It would be rerun only when new concrete mix examples would be added to the data set, or when a new parameter grid is tried.

4. The goodness-of-fit metrics of all three methods with all runs are very close together (the difference is negligible), which means that not one is clearly better than the others. However, it is possible that clearly better results are obtainable when trying more runs.

5. It is clear that the goodness-of-fit of BO and GS increases more when the parameter grid/parameter space is broadened, than if the number of epochs is increased.

5
Recommendations

Follow up research could include the following points:

- Using computer with more memory for GS. The reason 1000 was chosen for the number of epochs and a limited parameter grid was passed to the GS, is that it gave an OOM (Out Of Memory) error when more epochs or a larger parameter grid were added. By using more computational power (in this case more memory), the number of epochs or the size of the parameter grid could be increased, likely resulting in better results.

- If a better result is required, the next trial run could be using BO with 1000 epochs and an even larger parameter space (such as number of neurons: 8 - 8192, learning rate: 0.00001 - 30 and batch size: 8 - 1024). Run 4 with BO (increased parameter space) didn't have a very high R-squared score, but that could be because it was an 'unfortunate' run. Trying it more times and averaging over the results would give more insight.

- Tuning the number of hidden layers (model depth). Increasing the number of layers increases the computation time exponentially, however it could be interesting to see how much it improves the model. Then it can be decided if the better fit is worth the extra execution time for this specific 'concrete problem' (not likely).

- Tuning with Keras Tuner. Keras Tuner is a simple hyperparameter optimization library with which you can implement algorithms like Bayesian optimization and Random search.

- Tuning with the Genetic Algorithm (an evolutionary algorithm). Evolutionary algorithms are powerful and elegant algorithms that use mechanisms like reproduction, mutation and selection.

- Tuning with scikit learn's RandomizedSearchCV where instead of trying all parameter combinations (like grid search), a fixed number (predefined 'budget' by the engineer) of randomly selected combinations are tried.

- Visualization of hyperparameter tuning with for instance a parallel coordinates plot. There are a number of applications available that can create interesting visualizations of hyperparameter tuning.

- The fluctuations of the R-squared scores of the methods could be narrowed down if one would write a simple piece of code with a loop, in which the ANN is trained and then calculates the R-squared score (using the same hyperparameters in each loop). A more reliable R-squared score could then be found when averaging over the scores obtained in the loops. This could yield a better comparison.

Bibliography

Asteris, P. and Mokos, V. (2020). Concrete compressive strength using artificial neural networks. *Neural Computing and Applications*.

Ben Chaabene, W., Flah, M., and Nehdi, M. (2020). Machine learning prediction of mechanical properties of concrete: Critical review. *Construction and Building Materials*, 260:1–18.

Brownlee, J. (2019). Loss and loss functions for training deep learning neural networks.

Brownlee, J. (2021). Gentle introduction to the adam optimization algorithm for deep learning.

Chollet, F. (2017). *Deep Learning with Python*. Manning Publications, City, State or Country, version 6 edition.

GCP (2018). Using the right concrete mix.

Glen, S. (2020). Absolute error & mean absolute error (mae).

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. Adaptive computation and machine learning. MIT Press.

Hinton, A. I. A. i. O. (2017). Lecture 16.3 — Bayesian optimization of hyper parameters — [Deep Learning | Hinton | UofT].

Khademi, F. and Behfarnia, K. (2016). Evaluation of concrete compressive strength using artificial neural network and multiple linear regression models. *International Journal of Optimization in Civil Engineering*, 6:423–432.

Liu, D. (2017). A practical guide to relu.

Ng, A. (2011). Machine learning.

Nogueira, F. (2014). Bayesian Optimization: Open source constrained global optimization tool for Python.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Tobin, J. (2019). debugging_jan19.

Wan, Z., Xu, Y., and Šavija, B. (2021). On the use of machine learning models for prediction of compressive strength of concrete: Influence of dimensionality reduction on the model performance. *Materials*, 14:713.

Yeh, I.-C. (1998). Modeling of strength of high-performance concrete using artificial neural networks." cement and concrete research, 28(12), 1797-1808. *Cement and Concrete Research*, 28:1797–1808.