Evaluation of Multi-Layer Perceptron Regression

Abstract—**This report evaluates the performance of various regression models in predicting target values from a multivariate dataset. A baseline linear regression model, a baseline multi-layer perceptron regression model and two optimized multi-layer perceptron regression models comparing different hyperparameter tuning techniques were implemented. The trade-offs between accuracy, training time and tuning techniques are discussed in detail.**

I. Introduction

Regression analysis is an important part of predictive modelling, allowing models to be created with the aim of predicting numerical values. There are various algorithms which are used for this, ranging from making linear assumptions, to complex nonlinear models.

This report aims to compare the performance of a baseline linear regression model, a baseline multi-layer perceptron model and two optimised multi-layer perceptron models, The investigation evaluates the models on predictive accuracy and computational efficiency.

Section II. provides descriptions of the backgrounds of the dataset, neural networks, multilayer perceptron regression, linear regression and the performance metrics used. Section III. Describes the methodology used to implement the regression models, the hyperparameter tuning techniques and the feature normalisation technique used. Section IV. Displays results for each model, including performance metrics and computational efficiency data. Finally, Section V. provides a detailed conclusion of the analysis, including recommendations for hyperparameters to use for this type of investigation and recommendations for further work.

II. Background

A. *Concrete Compressive Strength Dataset*

The Concrete Compressive Strength Dataset (CCSD) is sourced from the UCI Machine Learning Repository, which is a widely used dataset repository used for machine learning tasks and benchmarking models. The dataset contains over 1,000 instances, each with 8 input features which describe the properties and composition of the concrete. The target variable is the compressive strength of the concrete, measured in megapascals (MPa).

The dataset was originally studied by Yeh in 1998[1], providing a benchmark for evaluating regression models. Predicting compressive strength is a crucial task in the civil engineering field, where accurate estimations are needed for ensuring structural safety. The interactions between the feature variables for prediction of compressive strength is complex and highly nonlinear, meaning that an advanced regression model is required for accurate predictions.

The CCSD is complete and does not contain missing values, meaning it can be used directly without the need for methods of handling missing data.

B. *Neural Networks*

Neural networks are an advanced type of machine learning algorithm inspired by the human brain. They consist of small units called neurons, which work together in layers:

- Input layer takes in the data
- Hidden layers identify patterns in the data
- Output layer gives the final prediction.

Neural networks excel at capturing nonlinear relationships, making them particularly suitable for applications like predicting compressive strength of concrete.

C. *Multi-Layer Perceptron Regression (MLP)*

MLP regression is a type of neural network supervised learning technique used for predicting numerical values. The algorithm uses a neural network composed of layers of neurons to model nonlinear relationships between features and the target variable. Each neuron processes the input data and sends the results to the next layer. Preprocessing steps, such as feature normalization, are critical for ensuring consistent scaling of inputs, ensuring the model is as accurate as possible. The methodology MLP regression is further detailed in Section III.A. Overall, it is an ideal algorithm for capturing complex relationships between input features, which is the case with the dataset used in this investigation.

D. *Linear Regression*

Linear regression is one of the simplest, but also most widely used models for predicting a numerical target variable. It assumes a linear relationship between the feature variables and the target. The algorithm is both computationally efficient and easily interpretable, making it a good starting point for implementing regression.

It is a solid choice for comparing more complex models, because it makes minimal assumptions about the data, therefore providing an excellent baseline. Due to the nature of the linear assumptions of the regressor, it shows clearly the advantages provided by more flexible nonlinear models such as MLP.

E. *Performance Metrics*

To evaluate the performance of the regression models, two key performance metrics are used: Mean Squared Error and the Coefficient of Determination (R^2).

MSE measures the average squared difference between the predicted and actual values, and the equation used to calculate this is given in Equation 1. Smaller MSE values indicate better performance as they indicate that predicted values are close to the actual values.

$$MSE = \frac{1}{N}\sum_{j=1}^{N}\left(y_j - \hat{y}_j\right)^2 \tag{1}$$

The R^2 value evaluates the proportion of variance in the target variable. The closer the value is to 1, the better the performance of the regressor. The R^2 value is calculated using the formula in Equation 2.

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{2}$$

where SS_{res} is the residual sum of squares:

$$SS_{res} = \sum_{i=1}^{n}(y_i - \breve{y}_i)^2 \qquad (3)$$

And SS_{tot} is the total sum of squares:

$$SS_{res} = \sum_{i=1}^{n}(y_i - \bar{y})^2 \qquad (4)$$

where y_i is the actual value, $\breve{y}_i$ is the predicted value and $\bar{y}$ is the mean of the actual values.

III. Methodology

This section details how each of the regression models was implemented, including how hyperparameters were tuned and the method of feature normalisation used.

A. MLP Regressor

The MLP regressor is implemented using a network with multiple layers of neurons. The input layer receives the features from the dataset, which are normalised, as described in Section III.C. Hidden layers process the data using weighted connections and activation functions, which allow the network to recognise complex patterns in the features. The output layer then provides a prediction of the compressive strength based on the processing completed in the hidden layers. The model is trained by comparing its predictions to actual values using the mean squared error function and adjusting weights and biases in the hidden layers through backpropagation. The process is repeated over many epochs to minimise errors.

B. Linear Regression

The linear regression algorithm is implemented using Scikit-learn's LinearRegression class, which which fits a linear model with coefficients $(\omega_1, \ldots, \omega_p)$ for equation 5:

$$\hat{y}(\omega, x) = \omega_0 + \omega_1 x_1 + \cdots + \omega_p x_p \qquad (5)$$

where y is the predicted value.

The algorithm minimises the residual sum of squares between the targets, solving equation 6 [2]:

$$\min \omega \, ||X\omega - y||_2^2 \qquad (6)$$

where X is the feature matrix and y is the target vector.

C. Hyperparameter Tuning

To optimise the performance of the MLP model, hyperparameter tuning was used to identify the best combination of parameters to minimise prediction error. For the purpose of this investigation, two automated hyperparameter tuning functions were compared: Keras Tuner and GridSearchCV.

Keras Tuner uses an intelligent search strategy to efficiently explore the hyperparameters. For the MLP regressor, the Hyperband algorithm is used, which dynamically alllocates resources to promising configurations and disregards less effective ones.[3]

GridSearchCV uses a more exhaustive approach, testing all possible combinations of hyperparameters to determine the optimal combination. Each combination is evaluated using 3-fold cross-validation, ensuring the model performs for variations in data.

D. Feature Normalisation

Feature normalisation is critical in the preprocessing step of both the linear regression and the MLP algorithm. The StandardScaler function is used to apply z-score normalisation, standardising the features of the dataset by removing the mean and scaling to unit variance.[4]

IV. Results

This section provides results for each of the models implemented. It displays the list of hyperparameters which are tuned by both tuning models and performance metrics for each model.

A. MLP Hyperparameter Tuning

Table I displays each of the hyperparameters which are tuned for the optimised MLP model. The table provides descriptions of each of the hyperparameters, along with the ranges they were tuned from. Due to the computational cost of GridSearchCV, the number of hyperparameters investigated is limited. Default hyperparameters can be found on the TensorFlow website [5].

TABLE I. List of Hyperparameters Tuned

	Hyperparameter	**Description**	**Range**
Keras Tuner	units_input	Number of neurons in the input layer	[32, 64, …, 256]
	activation_input	Activation function of the input layer	[‘relu’, ‘tanh’, ‘sigmoid’]
	num_layers	Number of hidden layers	[1, 2, 3, 4, 5]
	units_layer	Number of neurons in the i-th hidden layer	[32, 64, …, 256]
	activation_layer	Activation function for the i-th layer	[‘relu’, ‘tanh’, ‘sigmoid’]
	use_dropout_layer	Whether to apply dropout after the i-th hidden layer	[True, False]
	dropout_rate	Dropout rate for the i-th hidden layer	[0.0, 0.1,…, 0.5]
	optimizer	Optimiser used for training	[‘adam’, ‘rmsprop’]
	learning_rate	Learning rate for the selected optimizer	[1e-4, 1e-3, 1e-2]
	batch_size	Batch size used during training	[16, 32, 64, 128]
GridSearchCV	optimizer	Optimizer used for the training	[‘adam’, ‘rmsprop’]
	learning_rate	Learning rate for the selected optimizer	[0.001, 0.01, 0.1]
	batch_size	Batch size used during training	[16, 32]
	epochs	Number of epochs for training	[50, 100]

B. Model Performance Metrics

Table II summarises the performance of the three regression models – linear regression, baseline MLP and optimised MLP. The metrics reported include the MSE and R^2 score for both the training and testing datasets. This allows for a direct comparison of the models predictive accuracy and potential overfitting or underfitting.

TABLE II. MSE AND R^2 SCORES FOR EACH MODEL

Model	Dataset	MSE	R^2
Linear Regression	***Training***	110.6449	0.6105
	Test	95.9709	0.6276
Baseline MLP	***Training***	101.6354	0.6422
	Test	97.2075	0.6228
Keras Tuner Optimised MLP	***Training***	20.9020	0.9264
	Test	42.9010	0.8335
GridSearchCV Optimised MLP	***Training***	47.7416	0.8319
	Test	32.7806	0.8728

C. Training Process for MLP Models

This section presents the loss curves for the training and testing datasets for both the baseline MLP and optimised MLP models. This graph is useful as both training and testing datasets provide insight into how well the model performs. Training loss indicates how well the model fits the training data and the testing loss indicates how well the model reacts to unseen data.

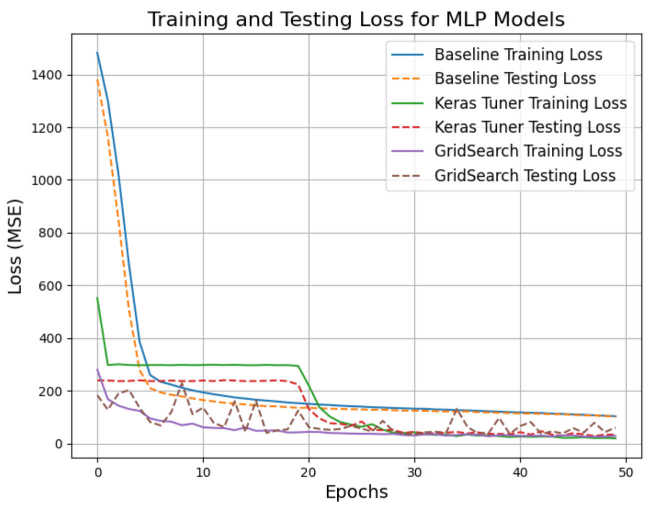

Fig. 1. Training and Testing Loss for MLP Models

D. Preciction Performance: Actual vs. Predicted Values

Fig. 2 displays a scatter plot comparing the actual target values with the predicted values for each model on the testing dataset, the closer the points on this plot lie to the diagonal line, the better the models performance.

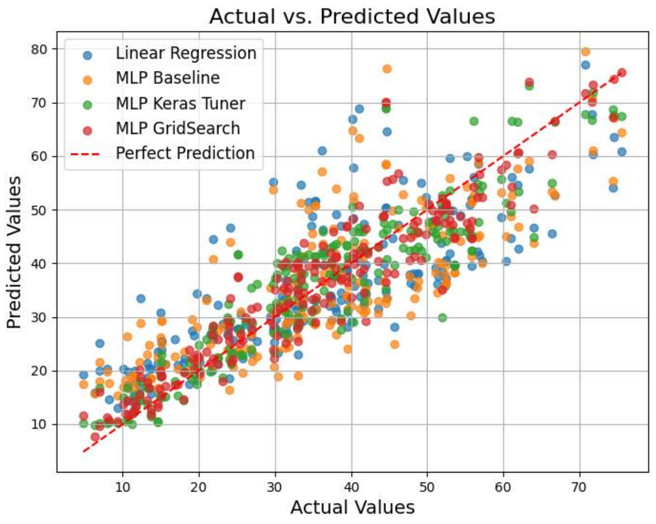

Fig. 2. Actual vs. Predicted Values for Each Model

E. Residual Analysis of Models

Fig. 3 shows residual plots, which plot the difference between actual and predicted values, with the goal of having the residual as close to zero as possible, indicating a perfect model with zero error. This clearly shows how each model performs in terms of minimising prediction errors.

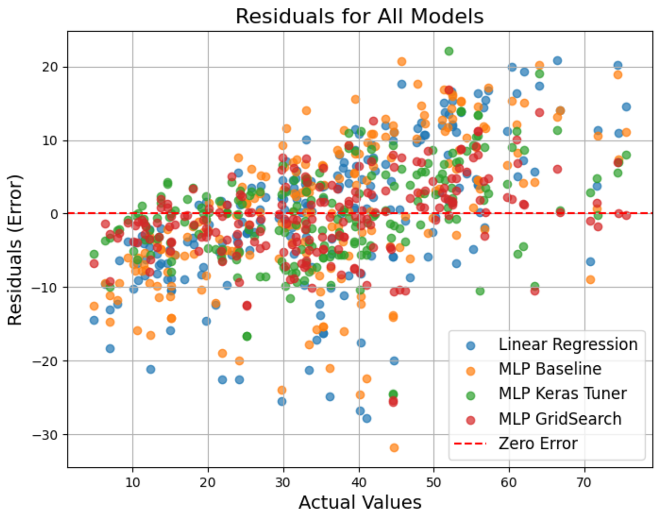

Fig. 3. Residual Plots for Each Model

Fig. 4 displays the density plot for the residuals of each of the regression models, allowing for a clear comparison of the accuracy of each model. The goal is to have max density at the zero residual value.

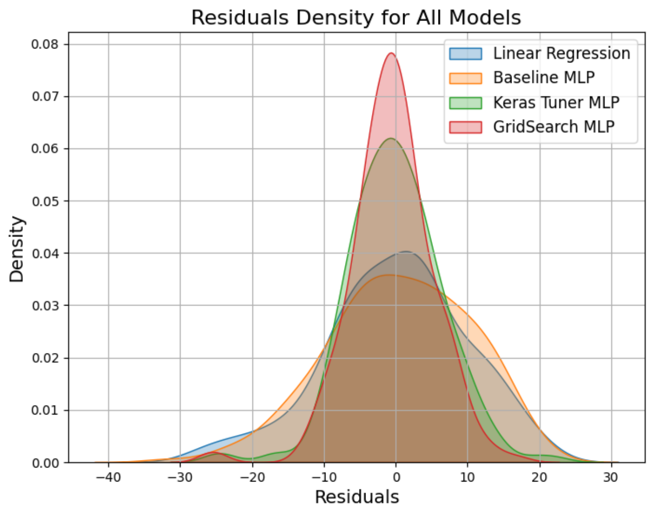

Fig. 4. Residuals Density Graph for Each Model

F. *Comparison of Computational Efficiency*

Table III displays the number of parameters investigated in each of the models, along with the total training times of each.

TABLE III. COMPARISON OF COMPUTATIONAL EFFICIENCY OF EACH MODEL

Model	**Training Time (s)**	**Parameters**
Linear Regression	0.01	9
Baseline MLP	5.04	4801
Keras Tuner Optimised MLP	7.51	55937
GridSearchCV Optimised MLP	319.08	9473

V. CONCLUSIONS

It can be concluded from table II, that both models with hyperparameter tuning perform significantly better in comparison to the baseline MLP and linear regression models. The Keras Tuner performs exceptionally well on the training dataset, with an MSE of 20.9020 and R^2 of 0.9264. However, there is a large drop in performance when using the unseen test dataset, suggesting that the model may be overfit to the training set.

The GridSearchCV performs consistently well across both the training and testing dataset, with performance even improving for the test set. This comes at a large computational cost, as can be seen in Table III. The GridSearchCV model takes 42 times longer to train than the Keras Tuner model, as it iterates through all possible combinations to determine the best one. The baseline MLP model only performs slightly better than the linear regression model for both training and test data when comparing MSE values, and actually falls behind linear regression in terms of R^2 value for the test set, highlighting the importance of hyperparameter tuning for MLP regression.

Fig. 1 provides insight into the loss of each MLP model over epochs, and each model performs as expected, with loss decreasing over epochs. Interestingly, the Keras Tuner model has the highest loss value over lower epochs, before improving in line with the GridSearch model after ~20 epochs. The baseline MLP improves significantly over the epochs, but doesn't reach the level of performance of either of the optimised models over the 50 epochs that all models are tested, however the baseline model is still improving at the 50 epoch mark, suggesting it may improve to the level of the optimised models with more epochs, however this would also increase computational cost.

Fig. 2 and Fig. 3 provide similar insights into the accuracy of each model, with Fig. 2 plotting the predicted values to the actual values for each model and Fig. 3 plotting the residuals of each model. The goal of Fig. 2 is to have as small a difference between the actual and predicted values as possible, which would result in a straight diagonal line. From the plot, both optimised models perform significantly better than the baseline models, with much fewer outliers and the majority of values being on the perfect diagonal line. Fig. 3 plots the residuals of the actual vs predicted values, with the goal being to have the residual value to be zero, indicating a perfect prediction. Once again, the optimised models outperform the baseline models, demonstrating their ability to capture nonlinear relationships in the data, which is visualised clearly in Fig. 4, with the baseline models having a much more spread out residual density plot, whereas the optimised models both have the majority of their residuals around zero, with the GridSearchCV model performing best.

Table IV. and Table V. show the hyperparameters which were found to give optimal performance when using GridSearchCV and Keras Tuning hyperparameter tuning respectively.

TABLE IV. GRIDSEARCHCV OPTIMISED HYPERPARAMETERS

batch_Size	16
learning_rate	0.01
optimizer	rmsprop
units_layer1	128
units_layer2	32

TABLE V. KERAS TUNER OPTIMISED HYPERPARAMETERS

units_input	96
activation_input	Relu
num_layers	4
units_layer 0, 1, 2 ,3, 4	192, 128, 32, 224, 64
activation_layer 0, 1, 2, 3, 4	tanh, sigmoid, tanh, relu
use_dropout_layer 0,1, 2, 3, 4	False, False, True, True
dropout_rate 0, 1, 2, 3, 4	0.2, 0.2, 0.0, 0.3
optimizer	Adam
learning_rate	0.003
batch_size	32

The Keras Tuner allows for many hyperparameters to be tuned quickly and computationally efficiently, however it is not as thorough in the hyperparameter tuning process as the GridSearchCV function. Therefore, for optimal results, it is recommended to use GridSearchCV for hyperparameter tuning, and for this dataset, it is recommended to use the hyperparameters presented in Table IV. for MLP regression for optimal results. In future work, it may also be worthwhile using Keras Tuner to narrow down the ranges used for GridSearchCV tuning, a combination of the two algorithms could provide accurate results while remaining computationally efficient.

REFERENCES

[1] I.-C. Yeh, “Modeling of Strength of High-Performance Concrete Using Artificial Neural Networks”, Cement and Concrete Research, Vol. 28, No. 12, pp 1797-1808

[2] Scikit-learn User Guide,, “1.1 Linear Models,” https://scikit-learn.org/1.5/modules/linear_model.html

[3] TensorFlow Tutorials, “Introduction to the Keras Tuner” https://www.tensorflow.org/tutorials/keras/keras_tuner

[4] Scikit-learn User Guide, “StandardScaler” Scikit-learn User Guide. https://scikitlearn.org/dev/modules/generated/sklearn.preprocessing.StandardScale r.htm

[5] TensorFlow Tutorials, “Multilayer Perceptrons” https://www.tensorflow.org/guide/core/mlp_core#train_the_model