Evaluation and comparison of different loss regression algorithms in Machine Learning

Research Question: To what extent do different loss regression algorithms in machine learning affect the accuracy and efficiency of a machine learning model?

Computer Science

3986 words

Introduction

Machine learning has a variety of applications and there are many different machine learning models. This paper will focus on the supervised regression models. Regression in machine learning is creating a prediction for continuous values. Classification is the other category of machine learning models, it is used to create a prediction for discrete labels. This paper is an investigation into the comparison of different loss regression algorithms and to what extent the choice of algorithm will affect the accuracy and efficiency of a machine learning model.

Background information about machine learning

Supervised machine learning is a process in which the machine learning algorithm is provided with a data set containing various features and labels. Features are characteristics of an object which may correlate to the label. Labels are the value that will be predicted which is based on the features provided. In a supervised machine learning process, labels are provided for every object.

E.g. If a machine learning algorithm was to be created for the purposes of predicting a person's max heart rate then some features might include: age, amount of exercise, gender, or diet. The label in this scenario would simply be the person's max heart rate. Some features may have a higher amount of correlation with the label, while others may have low correlation or none at all. The amount of correlation between a specific feature and the label is called the weight. The goal of the machine learning model is to assign correct weights to features so that the model will fit the data and provide an accurate prediction of a label based on given features.

For the algorithm to adjust the weights to better fit the data, it needs to know how accurate the prediction was. This is where loss regression algorithms come in. To determine the degree of inaccuracy, algorithms called loss regression algorithms are implemented.

Loss algorithms effectively measure the distance between the predicted label and the actual label. There are many variations of loss algorithms and different loss algorithms calculate a loss value in different ways. Because of all these variations, choosing a different loss regression algorithm should greatly affect the outcomes of the machine learning model.

The further a loss value is from zero, the less accurate the machine learning algorithm is. The machine learning algorithm then uses this loss value provided by the loss algorithm to adjust the weights in the right direction in a process called optimization.

Optimization is the way the machine learning model reduces loss, it adjusts the values assigned to the weights according to the loss . Further detail on optimization is outside the scope of this paper. For the experimentation portion of this paper, RMSProp will be utilized as the optimization algorithm and will remain as a constant (Brownlee, 2021).

The process of iteratively finding a loss value and then optimizing the machine learning model is then repeated for every datapoint in the dataset until the loss values reach an acceptable value or begin to plateau. Different loss regression models provide different loss values, respond differently to outliers, and have different degrees of accuracy.

This paper will be comparing Mean Squared Error, Mean Absolute Error, Huber loss, Log-Cosh loss, Mean Absolute Percentage Error, and Mean Squared Logarithmic Error, in the context of machine learning models. And evaluating the extent to which these different algorithms will affect the efficiency and accuracy of a machine learning model. The

implementation of the loss algorithms will be aided with the help of the Tensorflow Keras API (Tensorflow, 2021).

Introduction of the Loss Algorithms to be tested

Mean Squared Error (MSE / L2 Loss)

Mean Squared Error, also known as MSE or L2 Loss, is the most commonly used loss algorithm (Grover, 2018). It is defined as:

$$L = \frac{\sum_{i=1}^{n} (y_i - y_i^p)^2}{n}$$

Where n is the number of predicted values, y_i is the i^{th} true value from the data set, y_i^p is the i^{th} predicted output value, and L is the loss value. This algorithm takes the squared difference between the predicted value and actual value. It then averages this across the entire data set. L2 regression is quite responsive to outliers in a dataset. Because the difference is squared, the loss value will increase quadratically. When the difference is very high, as in the case of an outlier, the loss will be exponentially higher. As such, outliers are penalized much more than regular data points and this algorithm will make the machine learning algorithm adjust to fit outliers much more accurately at the expense of fitting the regular data.

"Although MSE is commonly-used in machine learning, it is neither the only practical loss function nor the best loss function for all circumstances" (Google Developers, 2020). Because MSE is the most commonly used algorithm for loss evaluation, for the purposes of this

paper, MSE will be used as the control, in order to provide a comparison to evaluate the extent to which choosing different loss regression algorithms will affect a machine learning model. In addition, MSE should be the best in terms of accuracy and efficiency.

Mean Absolute Error (MAE / L1 Loss)

Mean absolute error, also known as L1 loss or MAE, is another loss regression algorithm given as:

$$L = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n}$$

Where n is the number of predicted values, y_i is the i^{th} true value from the data set, y_i^p is the i^{th} predicted output value, and L is the loss value. This algorithm takes the absolute value of the difference between the actual and predicted values and then averages it across the entire dataset. "MAE is not sensitive towards outliers and given several examples with the same input feature values, and the optimal prediction will be their median target value" (Peltarion, n.d). The insensitivity towards outliers results from taking the absolute value of the error rather than squaring it as MSE does. This means that MAE will work well with datasets that have a normal distribution.

Huber Loss

Huber loss is given as:

$$L_{\delta}(y, y_{pred}) = \frac{1}{2}(y - y_{pred})^2 \quad for\ |y - y_{pred}| \leq \delta$$

$$Otherwise\ L_{\delta}(y, y_{pred}) = \delta|y - y_{pred}| - \frac{1}{2}\delta^2$$

Where δ is a hyperparameter, y is the true value and, y_{pred} is the predicted value. In machine learning, a hyperparameter is a variable that is changed manually by the programmer, in this case, the hyperparameter can be adjusted to determine the error value at which Huber will use the top equation or the bottom. This allows for the developer to modify the algorithm to better accommodate certain datasets. When the error is less than δ, then it will output a quadratic. When the error is greater than δ, then it will output an absolute error (Grover, 2018). Large errors usually result from an outlier. As a result, Huber loss will be less sensitive to outliers as it will be using the absolute error when evaluating outliers rather than the quadratic. In addition to that, as the machine learning algorithm becomes more accurate, the loss values will decrease. The Huber loss algorithm will then adapt to become more sensitive to slight changes making it more precise.

Log-Cosh Loss

Log-cosh is the logarithm of the hyperbolic cosine of the prediction error. Given as:

$$L = \sum_{i=1}^{n} log(cosh(y_i^p - y_i))$$

Where n is the number of predicted values, y_i is the i^{th} true value from the data set, y_i^p is the i^{th} predicted output value, and L is the loss value. For small error values, Log Cosh will output a value similar to $x^2/2$, where x is the error between the predicted value and actual value. In contrast, when the error is large, it will produce an output similar to $|x| - log(2)$ (Grover, 2018).

This is similar to the Huber Loss algorithm, however, in this algorithm, there is no hyperparameter for the programmer to adjust. So while it possesses the same concept of smoothing out large errors as Huber loss and becoming more precise as the machine learning model becomes more accurate. The value at which the algorithm will adjust for large errors is fixed and cannot be changed by the developer. This means that it cannot be adjusted to accommodate all datasets.

Mean Absolute Percentage Error (MAPE)

Mean Absolute Percentage Error is an extension of the Mean Absolute Error loss algorithm. It is given as:

$$\mathrm{L} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - y_i^p}{y_i} \right|$$

Where L is the loss, n is the number of predicted values, y_i is the i^{th} true value in the dataset and, y_i^p is the i^{th} predicted value. MAPE takes the absolute value of the error and divides it by the true value. It then takes that value as a percentage. Because it is a percentage the values are relative, this means that MAPE works well with datasets that have high orders of magnitude. In those datasets, loss functions which use exact values will output extremely high losses, because the magnitude of error will be increased. MAPE however, will produce the same error.

According to Pelatorion, an AI development company, "The MAPE also tends to be biased in favor of small predictions, because a prediction smaller than the true value can never have an error over 100%, whereas too large predictions can have arbitrarily large error" (Peltarion, n.d.). This means that a prediction that is greater than the true value will be more heavily penalized than a prediction that is less than the true value.

Mean Squared Logarithmic Error (MSLE)

Mean Squared Logarithmic Error is an extension of the Mean Squared Error loss algorithm. It is given as:

$$\mathrm{L} = \sum_{i=0}^{n} \frac{(log(y_i+1) - log(y_i^p+1))^2}{n}$$

Where L is the loss, n is the number of predicted values, y_i is the i^{th} true value in the dataset and, y_i^p is the i^{th} predicted value. Mean Squared Logarithmic Error can provide us with an effective measure of the ratio between the true and predicted value (Peltarion, n.d.).

The use of the logarithm allows the algorithm to focus on the relative differences between the predicted and true value rather than the exact difference. This means that when working with large values and data, the differences between the predicted and true values can be greater. A broad difference in a data set with large numbers will effectively be treated the same as a small difference in a data set with smaller values. In addition to this, Mean Squared Logarithmic Error penalizes underestimates more than overestimates. When the prediction is less than the true value, the loss will be higher. This will affect the weights and biases that the machine learning model will produce, and in turn affect the accuracy of the model.

Experimentation Outline

This experiment will test the different loss regression algorithms in a machine learning model and analyze the results.

Testing Materials

For the testing and implementation of these algorithms, the programming language Python will be used. It is a very powerful language often used for machine learning and is complete with many machine learning libraries. There are many prewritten machine learning Python libraries available making the testing of the different Loss algorithms simpler.

The datasets that will be used will be the “California Housing Data Set" provided by Google (Google, 2019), the “Heart Failure Prediction Dataset” (fedesoriano, 2021), and “Uber and Lyft Dataset Boston” (Maharjan, 2019). The latter two datasets are provided by Kaggle. Kaggle allows users to publish open datasets. Since these are open datasets uploaded by people, the validity and accuracy may come into question, however, for the purposes of this experiment,

the data does not have to be necessarily accurate. The machine learning model will still function as intended regardless of the data as long as it is formatted correctly.

The “Heart Failure Prediction Dataset” contains around 1,000 entries and will be abbreviated as “Heart dataset’. The “California Housing Dataset” contains around 17,000 entries and will be abbreviated as “Housing dataset”. The “Uber and Lyft Dataset Boston”contains around 600,000 entries and will be abbreviated as “Temperature Dataset”. These datasets have been selected because they provide a range of dataset sizes. Each dataset is a different size to see how different loss algorithms may differ based on dataset size. They all contain well documented data to provide insight on how the different loss algorithms would function in a real world application.

The implementation of the Machine learning loss regression algorithm using the Tensorflow APIs in Python is as shown below (Full code with comments included in Appendix A):

```
data = pd.read_csv(r"C:\Users\visitor\Downloads\heart.csv")
features = np.array((data['Age']))
label = np.array(data['MaxHR'])
model = keras.models.Sequential()
t1 = process_time()
model.add(keras.layers.Dense(units=1, input_shape=(1,)))
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=.
01),
              loss=LOSS_Algorithm,
              metrics=[keras.metrics.RootMeanSquaredError()])

history = model.fit(
    features,
    label,
    batch_size=10,
    epochs=200
)
t2 = process_time()
print(t2-t1)
weight = model.get_weights()[0]
bias = model.get_weights()[1]
print(weight)
print(bias)
```

Where `LOSS_Algorithm` is replaced with the current loss algorithm being evaluated.

Testing Method

With reference to Appendix A (Full Code). The dataset that will be used for evaluation will be imported into the variable `data` using Pandas. The string containing the directory for the dataset will be changed to test the current dataset.

The features and labels are chosen for each dataset. The Heart dataset will be using Age vs Max Heart Rate. The Housing dataset will be using Median Income vs Median House Price. The Temperature dataset will be using Temperature vs Apparent Temperature.

- The optimizer is a constant and will be Root Mean Squared Propagation.
- The metric that will be used to evaluate accuracy is Root Mean Squared Error. Further detail on the metric is mentioned below in the Evaluating Accuracy section.
- To test different loss algorithms, the parameter for loss will be changed.

Epochs and batch size will be constants when testing different loss algorithms within the same dataset, but will change for each dataset. In machine learning, each iteration through the dataset is called an epoch. The number of epochs represents the amount of times the machine learning algorithm has passed through an entire dataset (Bell & Gaillard, 2020). Batch size is the amount of samples that the model will go through before updating the parameters. Increasing the batch size will speed up the machine learning process.

For the Heart dataset, 200 epochs and a batch size of 10 will be used. For the Housing dataset, 200 epochs and a batch size of 64 will be used. For the Temperature dataset, 40 epochs and a batch size of 200 will be used. This is due to time limitations.

Method of Evaluating Accuracy

To evaluate the accuracy of the model, the specific loss value produced by each loss algorithm cannot be used as a comparison of accuracy. These loss values are not comparable to each other as they are of varying magnitudes and have different interpretations. To solve this issue of comparison, a metric will be used. Metrics in machine learning are used to measure and evaluate the performance of a machine learning model. The metric that will be used for this experiment will be Root Mean Squared Error. “The root mean square error (RMSE) has been used as a standard statistical metric to measure model performance in me-teorology, air quality, and climate research studies” (Chai & Draxler, 2014, 1247). Since RMSE is used as a standard, it can provide an accurate metric for the evaluation of the loss algorithms. In addition to that, RMSE will allow for easy interpretation as the value it produces will be in the same scale as the dataset (Bajaj, 2021). A RMSE value will be generated for each loss algorithm tested. This allows for a common value that can be utilized for the sake of comparison.

Method of Evaluating Efficiency

Machine learning in practice requires immense amounts of time and computing power. A machine learning algorithm requires many epochs to become accurate. As a result, small differences in the efficiency of the algorithms used in the model, will then therefore greatly affect the amount of time a machine learning model will need to train. Therefore choosing a different loss regression algorithm should greatly affect the efficiency of a machine learning model.

Number of Operations

One such method of evaluating the efficiency of an algorithm would be counting the number of operations performed. In loss regression algorithms, there are two factors that will affect the number of operations performed. The first is the predetermined number of operations that must be performed to calculate a loss value. The second is the number of epochs. As the number of epochs increases, so does the number of times the loss regression algorithm must be executed. However, counting operations does not take into account the complexity of the operations. An algorithm with two very long and complex operations will take longer than an algorithm with three short operations, even though the first algorithm has less operations. Along with this, the loss algorithm implementation in Keras consists of classes making evaluation via operation counting unviable.

Time for Execution

Instead, to provide a measurement for the efficiency of the algorithms, the time in seconds that the algorithm takes to run, will be recorded. Time measurement will also provide a real world context to measuring the efficiency of the algorithms. To do this, the `time` module in Python will be used. A variable (`t1`) will be set to `process_time()` at the beginning of training the machine learning model. After the algorithm code, another variable (`t2`) will be set to `process_time()`. The method `process_time()` gives the current process time in seconds (Python, 2021). Therefore time the algorithm took to execute will be calculated as `t1 - t2`. The time taken will be recorded for each algorithm.

Analysis

Efficiency

Dataset	MSE (Control)	MAE	Huber	Log Cosh	MAPE	MSLE
Heart	80.0625	79.640625	83.009375	79.01875	82.68125	83.353125
Housing	214.846875	215.84375	226.0375	218.09375	226.390625	220.759375
Temperature	531.946875	538.409375	573.934375	551.8875	559.246875	567.3125

Table 1: Avg. Time in seconds (Higher Value = Lower Efficiency)

Dataset	MSE (Control)	MAE	Huber	Log Cosh	MAPE	MSLE
Heart	0%	-0.53%	3.68%	-1.30%	3.27%	4.11%
Housing	0%	0.46%	5.21%	1.51%	5.37%	2.75%
Temperature	0%	1.21%	7.89%	3.75%	5.13%	6.65%
Avg % Diff.	0%	0.38%	5.59%	1.32%	4.59%	4.50%

Table 2: Percentage difference in Time from MSE (Control) based on Table 1

(Higher Percentage = Greater discrepancy)

Table 1 shows the average time that each algorithm took to complete the machine learning model for the respective dataset. Table 2 shows the percentage difference of each algorithm from the control (MSE) for each dataset, and the average percent difference across all datasets. It can be seen from Table 2 that all the algorithms were less efficient than the control. In ascending order, most to least efficient were, MSE, MAE, LogCosh, MSLE, MAPE, and Huber.

Huber loss appeared to be the least efficient algorithm. It had consistently higher average times in each experiment. In reference to Table 1, Huber took 573 seconds for the Temperature dataset, 226 seconds for the Housing dataset, and 83 seconds for the Heart dataset.

This can be attributed to the fact that Huber loss uses If statements to check whether or not the difference between the predicted value and actual value is greater than the hyperparameter δ. This constant checking may result in longer processing times.

When compared to the control, this resulted in an average of 5.59% less efficiency. While 5% isn't a large difference, according to Nanonets, a company specialized in training machine learning models for consumers, "[Our] training usually takes between 2-8 hours depending on the number of files and queued models for training" (Nanonets, n.d.). 5% of 8 hours is around 24 minutes, this shows that there can be a moderate decrease in efficiency when choosing loss algorithms. 5% was the highest discrepancy between an algorithm and the control. This means this was the furthest extent to which a loss algorithm affected the efficiency of the model.

MAE had the lowest discrepancies and was actually faster than MSE in the Heart dataset. For MAE this is most likely due to the similarity between MAE and MSE. The only operation that changed between the two algorithms was squaring the error rather than taking the absolute value. The similarity in complexity and algorithm structure may have resulted in the extremely small difference of 0.38%.

Overall however, MSE had the highest efficiency in comparison to the other algorithms.

Accuracy

Dataset	MSE (Control)	MAE	Huber	Log Cosh	MAPE	MSLE
Heart	27.89726	28.57866	28.50894	28.77952	28.27932	28.42298
Housing	83.8396	85.56638	85.55954	85.53376	93.55132	85.28362
Temperature	2.5743	2.60834	2.59868	2.59244	2.70888	2.58454

Table 3: Avg. Accuracy in RMSE Values (Higher Value = Lower Accuracy)

Dataset	MSE (Control)	MAE	Huber	Log Cosh	MAPE	MSLE
Heart	0.000	0.681	0.612	0.882	0.382	0.526
Housing	0.000	1.727	1.720	1.694	9.712	1.444
Temperature	0.000	0.034	0.024	0.018	0.135	0.010

Table 4: Difference from Control in RMSE Values (Higher Value = Lower Accuracy)

Table 3 shows the average accuracy of each loss algorithm for each dataset as RMSE values. Table 4 shows the difference in accuracies from the control. As mentioned in the Evaluating Accuracy section above, RMSE values are in the same scale as the dataset. This means that it can be interpreted as the average difference between a predicted label and true label.

These tables show that MAPE has one of the lowest accuracies. This is most likely because it is based on percentages and relative values. All the datasets had values that were relatively small. The Heart dataset's magnitude remained in the hundreds. The Housing dataset's values were scaled down, and thus it's magnitude remained only in the ones. The Temperature dataset's magnitude remained in the tens. This is evident as the Housing dataset had the smallest magnitude, and it was in the Housing dataset that MAPE had a much lower accuracy in comparison with the other algorithms. In addition to that, in the Heart dataset which had the highest magnitude, MAPE had an accuracy of 28.27932; this was the 2nd highest accuracy, behind MSE. Without any large values MAPE may fall behind in predicting accuracy, as MAPE's strength lies in predicting relative values. Further experimentation could be done on datasets with higher magnitude scales.

MSLE is a loss algorithm that is similar to MAPE in the sense that it deals with relative values. However, MSLE was one of the most accurate algorithms, second to MSE. The reason why MSLE is much more accurate than MAPE even though they are both relative errors, is because MSLE penalizes under predictions more, while MAPE penalizes over predictions more (Peltarion, n.d.).

Overall, MSE also had the lowest RMSE values, resulting in the highest accuracy.

Possible Error

Execution time depends on the hardware the machine learning model is being trained on. This experiment was conducted on a computer equipped with a Nvidia RTX 2060. With a higher or lower end machine, the results for time efficiency may be different. Additional testing on different hardware would resolve this.

In addition to that, this does not take into account the background processes that may have been running at the time of testing. Background processes may also affect the timing as extra resources would be consumed. This may account for spikes or drops in the amount of time taken for some trials. To minimize this error, steps such as doing a fresh restart of the system before running any programs, closing all other background processes and programs that may be running, monitoring if other programs are using a significant amount of resources via the task manager.

Another factor that may have affected the outcomes would be the learning rate. Learning rate is a hyperparameter and controls the amount to which the weights and biases are updated after each batch (Brownlee, 2021). Changing the learning rate can greatly affect the accuracy of a machine learning model. To minimize this error, the experiment was conducted with learning rate as a constant. Perhaps choosing different learning rates would have yielded different outcomes. To resolve this, additional testing could have been with different learning rates and comparing the results between them.

Conclusion

Machine learning is a quickly evolving field and is responsible for many of the new technologies in this day. Autonomous vehicles, pattern recognition, and predictive analysis, all

rely on machine learning models. These machine learning models must be accurate. Even a small discrepancy can lead to a wrong label and that may be the difference between a pedestrian and open road. The training of these models are also very important and can take a long time. Thus the efficiency of training a model is of great importance.

The extent to which different loss regression algorithms in a machine learning model will affect the efficiency is moderate. When training machine learning models on small datasets, the time discrepancy between different algorithms is only a few seconds, and has little to no statistical significance. However, when scaled up to datasets containing hundreds of thousands, to millions of data points, the time discrepancy will increase. Though some algorithms such as MAE and LogCosh, remained consistent with the control MSE.

The extent to which different loss regression algorithms in a machine learning model will affect the accuracy is usually slight but can be high in certain situations. Which algorithm is best will greatly depend on the dataset, however, the difference in accuracy between each algorithm did not vary by much, except for the MAPE algorithm, which was greatly affected by the scale of the dataset and which in turn greatly affected the machine learning model's accuracy.

MSE, the most widely used loss regression proved to be the most efficient and most accurate algorithm.

Appendix

Appendix A (Full code):

This is the full code used for testing along with comments for documentation

```
import pandas as pd
from tensorflow import keras
import numpy as np
from time import process_time

# import data from csv file, pathway string is changed to test each
dataset
# C:\Users\visitor\Downloads\heart.csv is the Heart dataset
# C:\Users\visitor\Downloads\california_housing_train.csv is the
Housing dataset
# C:\Users\visitor\Downloads\rideshare_kaggle.csv is the Temperature
dataset
data = pd.read_csv(r"C:\Users\visitor\Downloads\heart.csv")

# Input data into numpy array
# the specific features and labels change for each dataset
features = np.array((data['Age']))
label = np.array(data['MaxHR'])

# Create a Keras Sequential Model and compile it with specified
# optimizer, loss, and metric
# loss will be changed to fit the current experiment:
# keras.losses.MeanSquaredError()
# keras.losses.MeanAbsoluteError()
# keras.losses.huber()
# keras.losses.log_cosh()
# keras.losses.MeanAbsolutePercentageError()
# keras.losses.MeanAbsoluteLogarithmicError()
model = keras.models.Sequential()
```

```
# Start the time flag
t1 = process_time()
model.add(keras.layers.Dense(units=1, input_shape=(1,)))
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=.01),
              loss=keras.losses.MeanSquaredError(),
              metrics=[keras.metrics.RootMeanSquaredError()])

# model.fit trains the model using the specified parameters
history = model.fit(
    features,
    label,
    batch_size=10,
    epochs=200
)
# End the time flag
t2 = process_time()
# Prints the time taken
print(t2-t1)

# Get and print weights and bias
weight = model.get_weights()[0]
bias = model.get_weights()[1]
print(weight)
print(bias)
```

Appendix B (MSE Data)

The following tables contain data from the three analyzed datasets.

Dataset: Heart	Epochs: 200	Batch Size: 10	Learning Rate: 0.5
Trial #	MSE - Time	MSE - Loss Value	MSE - Accuracy
1	79.28125	745.3384	27.3009
2	80.109375	802.7323	28.3325
3	78.125	776.1408	27.8593
4	80.953125	799.184	28.2698
5	81.84375	768.6115	27.7238
Averages	80.0625	778.4014	27.89726

Dataset: Housing	Epochs: 200	Batch Size: 64	Learning Rate: 0.5
Trial #	MSE - Time	MSE - Loss	MSE - Accuracy
1	215.71875	7025.999	83.8212
2	211.59375	7029.0898	83.8397
3	211.6875	7027.9331	83.8328
4	214.96875	7033.1709	83.864
5	220.265625	7029.189	83.8403
Averages	214.846875	7029.07636	83.8396

Dataset: Temperature	Epochs: 40	Batch Size: 200	Learning Rate: 0.01
Trial #	MSE - Time	MSE - Loss	MSE - Accuracy
1	534.421875	6.6267	2.5742
2	528.109375	6.627	2.5743
3	527.390625	6.6287	2.5746
4	529.609375	6.6255	2.574
5	540.203125	6.6275	2.5744
Averages	531.946875	6.62708	2.5743

Appendix C (MAE Data)

The following tables contain data from the three analyzed datasets.

Dataset: Heart	Epochs: 200	Batch Size: 10	Learning Rate: 0.5
Trial #	MAE - Time	MAE- Loss Value	MAE - Accuracy
1	78.609375	23.224	28.9019
2	80.734375	23.3114	29.0838
3	80.40625	22.4244	27.6788
4	78.609375	23.2471	28.7784
5	79.84375	22.6173	28.4504
Averages	79.640625	22.96484	28.57866

Dataset: Housing	Epochs: 200	Batch Size: 64	Learning Rate: 0.5
Trial #	MAE - Time	MAE- Loss	MAE - Accuracy
1	213.15625	60.9833	85.5525
2	213.453125	60.9703	85.5862
3	213.796875	60.9703	85.5561
4	222.015625	60.9981	85.6102
5	216.796875	60.9829	85.5269
Averages	215.84375	60.98098	85.56638

Dataset: Temperature	Epochs: 40	Batch Size: 200	Learning Rate: 0.01
Trial #	MAE - Time	MAE- Loss	MAE - Accuracy
1	546.390625	2.1218	2.6082
2	539.5625	2.1221	2.6082
3	540.65625	2.1224	2.6081
4	528.859375	2.1229	2.6085
5	536.578125	2.1222	2.6087
Averages	538.409375	2.12228	2.60834

Appendix D (Huber Data)

The following tables contain data from the three analyzed datasets.

Dataset: Heart	Epochs: 200	Batch Size: 10	Learning Rate: 0.5
Trial #	Huber - Time	Huber - Loss Value	Huber - Accuracy
1	81.03125	22.1997	28.0667
2	87.59375	22.8585	28.5868
3	80.15625	22.4961	28.873
4	84.546875	22.7087	28.4987
5	81.71875	22.5772	28.5195
Averages	83.009375	22.56804	28.50894

Dataset: Housing	Epochs: 200	Batch Size: 64	Learning Rate: 0.5
Trial #	Huber - Time	Huber - Loss	Huber - Accuracy
1	227.75	60.4674	85.5761
2	229.90625	60.4867	85.571
3	225.453125	60.4515	85.5333
4	224.90625	60.5031	85.6012
5	222.171875	60.4684	85.5161
Averages	226.0375	60.47542	85.55954

Dataset: Temperature	Epochs: 40	Batch Size: 200	Learning Rate: 0.01
Trial #	Huber - Time	Huber - Loss	Huber - Accuracy
1	580.5625	1.6564	2.598
2	571.828125	1.6572	2.5989
3	569.21875	1.6571	2.5989
4	578.1875	1.6571	2.5989
5	569.875	1.657	2.5987
Averages	573.934375	1.65696	2.59868

Appendix E (LOG_COSH Data)

The following tables contain data from the three analyzed datasets.

Dataset: Heart	Epochs: 200	Batch Size: 10	Learning Rate: 0.5
Trial #	LOG_COSH- Time	LOG _COSH - Loss	LOG_COSH - Accuracy
1	77.703125	21.9634	27.9778
2	77.890625	22.0179	28.2639
3	79.109375	22.6065	28.4532
4	80.375	22.13	28.2063
5	80.015625	22.6136	29.2137
Averages	79.01875	22.26628	28.42298

Dataset: Housing	Epochs: 200	Batch Size: 64	Learning Rate: 0.5
Trial #	LOG_COSH- Time	LOG _COSH - Loss	LOG_COSH - Accuracy
1	215.015625	60.2694	85.4987
2	218.28125	60.283	85.544
3	213.8125	60.2678	85.4883
4	222	60.294	85.6273
5	221.359375	60.293	85.5105
Averages	218.09375	60.28144	85.53376

Dataset: Temperature	Epochs: 40	Batch Size: 200	Learning Rate: 0.01
Trial #	LOG_COSH- Time	LOG _COSH - Loss	LOG_COSH - Accuracy
1	544.828125	1.5249	2.5923
2	551.8125	1.5252	2.5924
3	548.96875	1.5253	2.5926
4	557.734375	1.5252	2.5922
5	556.09375	1.5256	2.5927
Averages	551.8875	1.52524	2.59244

Appendix F (MAPE Data)

The following tables contain data from the three analyzed datasets.

Dataset: Heart	Epochs: 200	Batch Size: 10	Learning Rate: 0.5
Trial #	MAPE - Time	MAPE - Loss	MAPE - Accuracy
1	81.15625	17.9909	28.8798
2	82.25	17.9597	28.6416
3	81.1875	18.3942	29.6905
4	83.796875	17.8524	28.7449
5	85.015625	17.565	27.9408
Averages	82.68125	17.95244	28.77952

Dataset: Housing	Epochs: 200	Batch Size: 64	Learning Rate: 0.5
Trial #	MAPE - Time	MAPE - Loss	MAPE - Accuracy
1	224.4375	30.8172	93.5199
2	230.140625	30.8303	93.6111
3	228.15625	30.8248	93.5474
4	229.765625	30.8066	93.7032
5	219.453125	30.8058	93.375
Averages	226.390625	30.81694	93.55132

Dataset: Temperature	Epochs: 40	Batch Size: 200	Learning Rate: 0.01
Trial #	MAPE - Time	MAPE - Loss	MAPE - Accuracy
1	560.734375	6.6677	2.7088
2	561.734375	6.6689	2.709
3	557.609375	6.668	2.7093
4	560.59375	6.667	2.7082
5	555.5625	6.6677	2.7091
Averages	559.246875	6.66786	2.70888

Appendix G (MSLE Data)

The following tables contain data from the three analyzed datasets.

Dataset: Heart	Epochs: 200	Batch Size: 10	Learning Rate: 0.5
Trial #	MSLE - Time	MSLE - Loss	MSLE - Accuracy
1	82.78125	0.0475	28.4621
2	82.484375	0.046	27.9387
3	85.4375	0.0468	28.2892
4	83.5625	0.049	28.6676
5	82.5	0.0459	28.039
Averages	83.353125	0.04704	28.27932

Dataset: Housing	Epochs: 200	Batch Size: 64	Learning Rate: 0.5
Trial #	MSLE - Time	MSLE - Loss	MSLE - Accuracy
1	229.21875	0.1713	85.3395
2	212.46875	0.171	85.2827
3	222.0625	0.171	85.2924
4	222.59375	0.171	85.2298
5	217.453125	0.171	85.2737
Averages	220.759375	0.17106	85.28362

Dataset: Temperature	Epochs: 40	Batch Size: 200	Learning Rate: 0.01
Trial #	MSLE - Time	MSLE - Loss	MSLE - Accuracy
1	560.421875	0.0083	2.5844
2	574.46875	0.0083	2.5845
3	570.046875	0.0083	2.5845
4	564.359375	0.0083	2.5846
5	567.265625	0.0083	2.5847
Averages	567.3125	0.0083	2.58454

Appendix H (Temperature Weights and Biases)

The following tables show the weights and biases for the Temperature dataset produced by the machine learning model.

Trial #	MSE Weight	MSE Bias	MAE Weight	MAE Bias
1	1.108846	-8.220525	1.121087	-8.571456
2	1.1292224	-8.200765	1.1053643	-8.629949
3	1.1013932	-8.239942	1.1235367	-8.55694
4	1.0979809	-8.214067	1.1112779	-8.607643
5	1.1078904	-8.206572	1.1091402	-8.5943775
Averages	1.10906658	-8.2163742	1.11408122	-8.5920731

Trial #	Huber Weight	Huber Bias	LogCosh Weight	LogCosh Bias
1	1.1052512	-8.741795	1.1260625	-8.810906
2	1.1163074	-8.740128	1.132462	-8.852224
3	1.1125615	-8.747874	1.1124601	-8.861714
4	1.1153513	-8.718907	1.1337826	-8.787208
5	1.0948904	-8.759996	1.1272645	-8.835968
Averages	1.10887236	-8.74174	1.12640634	-8.829604

Trial #	MAPE Weight	MAPE Bias	MSLE Weight	MSLE Bias
1	1.0524025	-6.3364296	1.0859221	-7.49919
2	1.0484768	-6.4157805	1.10104	-7.565475
3	1.0420319	-6.3579936	1.0717499	-7.6128883
4	1.0474616	-6.374208	1.0909668	-7.514061
5	1.062215	-6.3523254	1.1064085	-7.6117845
Averages	1.05051756	-6.36734742	1.09121746	-7.56067976

Appendix I (Housing Weights and Biases)

The following tables show the weights and biases for the Housing dataset produced by the machine learning model.

Trial #	MSE Weight	MSE Bias	MAE Weight	MAE Bias
1	42.2116	45.00687	43.811195	21.064585
2	41.277092	42.715454	44.63333	20.161041
3	41.994198	43.19452	44.672108	20.822624
4	41.988735	43.96257	44.83229	21.380533
5	41.888454	43.081284	42.386726	18.435741
Averages	41.8720158	43.5921396	44.0671298	20.3729048

Trial #	Huber Weight	Huber Bias	LogCosh Weight	LogCosh Bias
1	43.84214	19.473877	43.317924	19.121943
2	43.52707	19.519585	42.525726	17.392075
3	44.61944	19.317661	44.083755	17.899347
4	43.89161	20.512718	44.48384	20.90856
5	42.831028	18.809006	45.2915	19.720121
Averages	43.7422576	19.5265694	43.940549	19.0084092

Trial #	MAPE Weight	MAPE Bias	MSLE Weight	MSLE Bias
1	42.71601	2.9315662	41.388153	28.344296
2	41.73403	0.8021039	41.43916	30.144321
3	41.461193	1.7265112	41.442665	30.261599
4	42.659016	2.4462001	41.44225	30.176409
5	40.45945	1.2975962	41.47616	30.123442
Averages	41.8059398	1.84079552	41.4376776	29.8100134

Appendix J (Heart Weights and Biases)

The following tables show the weights and biases for the Heart dataset produced by the machine learning model.

Trial #	MSE Weight	MSE Bias	MAE Weight	MAE Bias
1	-0.95088553	192.30116	-1.3818727	192.3214
2	-0.84067196	191.80081	-0.6946636	195.70607
3	-0.67174673	191.97098	-1.4883727	195.14899
4	-1.4016434	191.32152	-0.8588391	194.22064
5	-0.3952073	192.73055	-1.0108745	194.25867
Averages	-0.852030984	192.025004	-1.08692452	194.331154

Trial #	Huber Weight	Huber Bias	LogCosh Weight	LogCosh Bias
1	-1.7873967	194.20598	-1.2654377	194.55777
2	-1.4991885	194.40128	-0.5462213	194.20438
3	-0.6781417	195.45302	-1.3078338	194.99435
4	-1.6341299	195.22679	-0.7296423	195.13484
5	-0.5848535	194.50253	-1.1964031	193.84624
Averages	-1.23674206	194.75792	-1.00910764	194.547516

Trial #	MAPE Weight	MAPE Bias	MSLE Weight	MSLE Bias
1	-1.4761598	189.17418	-0.75187886	185.67654
2	-1.0120485	190.06659	-1.0127996	186.24951
3	-0.8076544	189.72086	-1.3855598	185.17455
4	-1.281069	188.42126	-1.0041956	185.26941
5	189.06065	-1.3369223	-1.2061927	185.6139
Averages	36.89674366	151.2091935	-1.072125312	185.596782

References

Algorithmia. (2018, April 30). *Introduction to Loss Functions*. Algorithmia. https://algorithmia.com/blog/introduction-to-loss-functions

Bajaj, A. (2021, July 19). *Performance Metrics in Machine Learning [Complete Guide] - neptune.ai*. Neptune.ai. https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide

Bell, D. J., & Gaillard, F. (2020, May 27). *Epoch (machine learning)* | *Radiology Reference Article*. Radiopaedia. https://radiopaedia.org/articles/epoch-machine-learning?lang=us

Brownlee, J. (2019, January 25). *Understand the Impact of Learning Rate on Neural Network Performance*. Machine Learning Mastery. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Brownlee, J. (2019, January 30). *How to Choose Loss Functions When Training Deep Learning Neural Networks*. Machine Learning Mastery. https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

Brownlee, J. (2021, May 24). *Gradient Descent With RMSProp from Scratch*. Machine Learning Mastery. https://machinelearningmastery.com/gradient-descent-with-rmsprop-from-scratch/

Chai, T., & Draxler, R. R. (2014, June 30). Root mean square error (RMSE) or mean absolute error (MAE)? –Arguments against avoiding RMSE in the literature. *Geoscientific Model Development*, *7*(3), 1247. https://doi.org/10.5194/gmd-7-1247-2014

Demand Planning. (2019). *Mean Absolute Percentage Error (MAPE) &WAMPE*. Demand Planning LLC. https://demandplanning.net/MAPE.htm

fedesoriano. (2021, September 10). *Heart Failure Prediction Dataset*. Kaggle. https://www.kaggle.com/fedesoriano/heart-failure-prediction

Garmsiri, S. (2018, September 6). *Art of Choosing Metrics in Supervised Models Part 1*. Towards Data Science. https://towardsdatascience.com/art-of-choosing-metrics-in-supervised-models-part-1-f960ae46902e

Google. (2019, March 26). *California Housing Data Set Description | Machine Learning Crash Course*. Google Developers. https://developers.google.com/machine-learning/crash-course/california-housing-data-description

Google Developers. (2020, February 10). *Descending into ML: Training and Loss | Machine Learning Crash Course*. Google Developers. Retrieved 2021, from https://developers.google.com/machine-learning/crash-course/descending-into-ml/training-and-loss

Grover, P. (2018, 6 5). *5 Regression Loss Functions All Machine Learners Should Know*. Heart Coment ML. https://heartbeat.comet.ml/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0

Keras. (n.d.). *RMSprop class - Optimizers*. Keras. Retrieved 2021, from https://keras.io/api/optimizers/rmsprop/

Maharjan, B. (2019, October 12). *Uber and Lyft Dataset Boston, MA*. Kaggle. Retrieved 2021, from https://www.kaggle.com/brllrb/uber-and-lyft-dataset-boston-ma

Nanonets. (n.d.). *How long does it take to train a model*. Nanonets. Retrieved 2021, from https://nanonets.com/help/ocr/how-long-does-it-take-to-train-a-model

Peltarion. (n.d.). *MAPE / Mean absolute percentage error on the Peltarion Platform*. Peltarion. Retrieved 2021, from

https://peltarion.com/knowledge-center/documentation/evaluation-view/regression-loss-metrics/mape-/-mean-absolute-percentage-error

Peltarion. (n.d.). *Mean absolute error loss function*. Peltarion. Retrieved 2021, from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-absolute-error

Peltarion. (n.d.). *Mean squared error loss function*. Peltarion. Retrieved 2021, from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error

Peltarion. (n.d.). *Mean squared logarithmic error (MSLE)*. Peltarion. Retrieved 2021, from https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-logarithmic-error-(msle)

Python. (2021, 12 14). *time — Time access and conversions — Python 3.10.1 documentation*. Python Docs. Retrieved 2021, from https://docs.python.org/3/library/time.html#time.process_time

Tensorflow. (2021, November 12). *Training and evaluation with the built-in methods*. TensorFlow. Retrieved 2021, from https://www.tensorflow.org/guide/keras/train_and_evaluate

Tensorflow. (2021, 12 13). *TensorFlow API Python*. TensorFlow. Retrieved 2021, from https://www.tensorflow.org/api_docs/python/tf

Tensorflow. (2021, 12 13). *TensorFlow Losses Overview*. TensorFlow. Retrieved 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/losses