Using a GAN Network to Maximize Concrete Compressive Strength to Mix Weight Ratio

Charles Middleton, Jonathan Hwang, Stephan Jean, Cecilia Doyle

535.742-Fall 2023 Final

December 12, 2023

Abstract

A machine learning model was developed to identify a concrete mixture having maximum strength while minimizing mixture weight using a dataset of various concrete mixes. This can be formally defined as maximization of $\frac{28\text{-}day\ compressive\ strength}{mix\ weight}$. To do so, several machine learning techniques were employed, including three deep neural networks, a GAN model, and combinatorial pattern recognition. Ultimately, the developed neural network models were used to estimate the 28-day compressive strength of concrete mixtures, and optimization techniques were applied to find an optimal mix that maximizes the defined objective function. For the data set analyzed, the optimization model predicted a minimum ratio of mix weight to 28-day compressive strength of 31.611 kg/MPa.

1.0 Introduction

A plurality of machine learning architectures were devised, trained, and combined to analyze a set of concrete data and estimate an optimum mixture to maximize the 28-day compressive strength while minimizing a mix weight. The architectures included three deep neural networks (DNN) and a generative adversarial network (GAN). Data used for this study was downloaded from the University of California, Irvine (UCI) machine learning repository. The dataset provided the compressive strength of several concrete mixtures, each composed of seven components (in kg/m^3 of mixture), which were as follows: cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate, and fine aggregate. Although much more data was provided in the UCI data set, the present study evaluated only mixtures with an age of 28 days. Neural networks in the model were trained to predict the 28-day compressive strength of a mixture based on the mixture composition or the amount of each of the seven components. Once the strength was predicted from a mixture composition, this was incorporated into an optimization model to maximize the ratio of $\frac{28\text{-}day\ compressive\ strength}{mix\ weight}$, ensuring a concrete mixture with high strength while maintaining a low weight.

2.0 Model Methods and Results

2.1 Data Segregation

This analysis was only concerned with 28-day concrete mixes. Thus, the initial step in the optimization process was to extract the 425 data points corresponding to 28-day compressive strength from the full UCI dataset. This subset of the data was saved and prepared for use to train the First DNN, which is discussed in Section 2.3 below.

2.2 Data Pre-Processing

The extracted data set was scaled for importation into the machine learning model. The input and output data were scaled using the MinMaxScaler function to transform the data in the range of [0, 1], ensuring the features have similar scales while preserving any underlying data distributions.

2.3 First Deep Neural Network

The first step was to process the extracted data through a Deep Neural Network, this deep neural network is composed of four hidden layers of different sizes. The first layer is comprised of 64 neurons, the second with 32 neurons, the third with 16 neurons, and final hidden layer with 8 neurons. All the hidden layers have a twenty percent drop out rate, and all used a ReLU Activation function. The output layer used a linear activation function. The validation split was set to 10 percent. The deep neural network used a mean square error loss function and was coupled with an Adaptive Moment Estimation optimizer. The batch size was set to 512 and 1000 iterations were used in the model. This resulted in 3265 trainable parameters and no non-trainable parameters. The First DNN is summarized in Table 1 below.

The output of the First DNN was plotted. The plot in Figure 1 is the number of iterations versus the model loss. In Figure 1, we can see that the First DNN gets consistent with the set dropout rate through the loss values. When the model crosses its 600th iteration, it converges to a steady value of approximately .012 for the loss and about .0052 for the validation loss.

Layer	Type	Neurons | Rate	Activation Function
One	Dense	64	ReLU
	Dropout	0.2	
Two	Dense	32	ReLU
	Dropout	0.2	
Three	Dense	16	ReLU
	Dropout	0.2	
Four	Dense	8	ReLU
	Dropout	0.2	
Output	Dense	1	Linear

Table 1: Architecture of First DNN

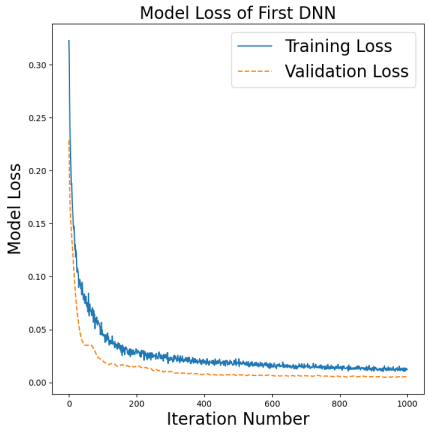

Figure 1. Training and Validation Loss of the First DNN

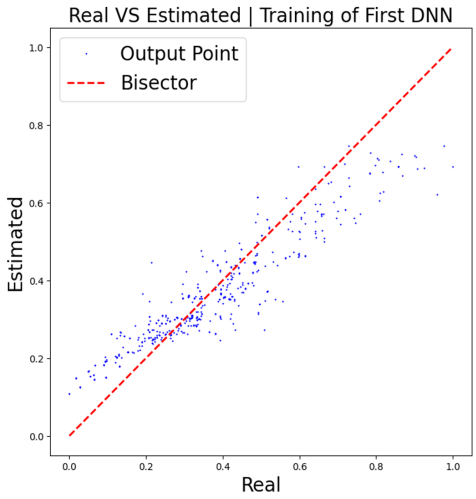

Figure 2. Real vs. Estimated of the First DNN

Next, the real versus established value for training was plotted in Figure 2. All the output points were within acceptable proximity of the bisector line; there is also a good concentration of points around the value of 0.2 to 0.6 for both real and estimated data. Furthermore, the scaled range showed outlier values away from the set bisector line. However, the amount of output points that lie away from the bisector was considered negligible.

2.4 GAN Model

A GAN is a deep learning model commonly used for generating realistic data, such as images, text, or other structured data. The GAN model will include two neural networks: a generator and a discriminator. The core idea behind a GAN model is to create a system where the generator and discriminator are trained together and in a competitive manner. A properly designed and trained GAN network can generate similar data to a given dataset. The architecture of the generator and discriminator networks developed are summarized below in Table 2, and the GAN model was trained using the segregated subset (e.g., the 425 data points) of data. For the discriminator network, the requirements (e.g., type, neurons, and activation function) were specified in the problem statement. More design latitude was permitted with the generator network, with the only firm requirement being that the network be a deep neural network with three hidden layers. ReLU activation functions were used in the deep networks for

both the generator and discriminator networks. The ReLU activation function is simple and computationally efficient, involving an essential thresholding operation. The linear and sigmoid activation functions were selected for the output layers for the generator and discriminator networks, respectively. Suppose the input data is linear in nature. In that case, the linear activation function maintains the linearity, which is beneficial in solving problems having linear relationships between input features and output targets, such as our data. The sigmoid activation function has the advantage of mapping its input to a fixed range of [0, 1], representing our data. During training, the discriminator's weights were not updated since, in the GAN framework, the discriminator is not trained alongside the generator. Additionally, noise was introduced into the model while training. Lastly, we used the Adaptive Moment Estimation optimizer, "Adam," for the model.

GAN			
Generator			
Layer	Type	Neurons	Activation Function
One	Dense	32	ReLU
Two	Dense	64	ReLU
Three	Dense	16	ReLU
Output	Dense	1	Linear
Discriminator			
Layer	Type	Neurons	Activation Function
One	Dense	32	ReLU
	Dropout	0.2	
Two	Dense	16	ReLU
	Dropout	0.2	
Three	Dense	8	ReLU
	Dropout	0.2	
Output	Dense	1	Sigmoid

Table 2: GAN Architecture

2.5 New Data Generation

The trained GAN model was then used to generate 575 new data points. One advantage of using the GAN model to generate new training data is that the data was more likely to have the same statistical properties and pedigree as the original training data than if points were randomly generated. These new 575 data points were combined with the original 425 data points used to train the GAN model, increasing our training data to 1000 data sets.

2.6 Second Deep Neural Network

This data set was the input to a DNN with four hidden layers.. The architecture of the DNN is displayed in Table 3. The addition of a dropout layer is to prevent overfitting. Additionally, the parameters for the DNN are number of epochs of 1000, batch size of 512, and validation of 0.1.

Layer	Type	Neurons | Rate	Activation Function
One	Dense	56	ReLU
	Dropout	0.1	
Two	Dense	32	ReLU
	Dropout	0.1	
Three	Dense	16	ReLU
	Dropout	0.1	
Four	Dense	8	ReLU
	Dropout	0.1	
Output	Dense	1	Linear

Table 3: Architecture of Second DNN

Figure 3 below displays the model loss vs. the model's number of epochs. Additionally, Figure 4 indicates the training vs the estimated data within the calibrated domain of [0,1]. As shown in the actual vs. calculated plot, there is a concentrated set of data points at around the 0.5 bisector section. This can be explained by the data generated by the GAN network, which only produced data that had an output in the middle of the calibrated domain. Additionally, this could in part be due to the GAN model approaching mode collapse during training. This was identified as a known issue; however, due to time and computing complexity, we accepted this issue, which will be noted as a future improvement.

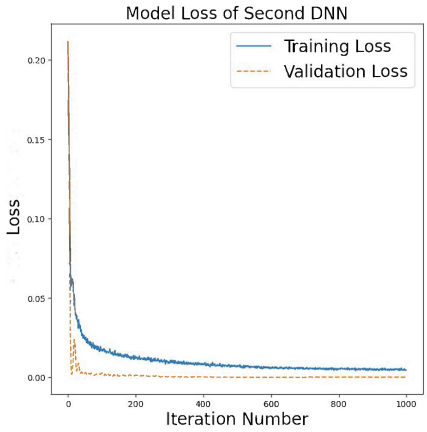

Figure 3. Training and Validation Loss of the Second DNN

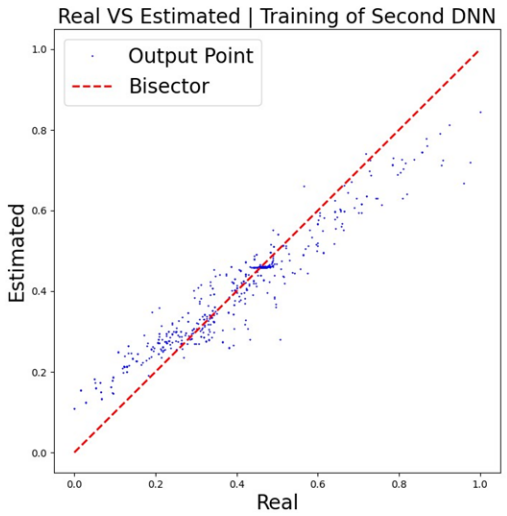

Figure 4. Real vs. Estimated of the Second DNN

2.7 Combination Optimization

A combinatory pattern recognition model was created using an RTT of 0.1 and RRS of 3. Due to the computing complexity, only a DNN model was used for combination analysis, with its parameters shown in Table 4. This model aims to define which of the 7 input features most impact the overall output.

Layer	Type	Neurons | Rate	Activation Function
One	Dense	28	Relu
Two	Dense	10	Relu
Output	Dense	1	Linear

Table 4: Architecture of DNN for Combination Optimization

The number of possible combinations of features is $2^n - 1$, where n is the number of features. In this case the total number of combinations is 127. Each combination was run within each unique RTT and RRS to determine the best outcome. The results are shown in Figure 5. At a glance, it can be determined

that the three variables that have the most impact are Cement, Blast Furnace Slag, and Superplasticizer. At the same time, water had some effect on the output.

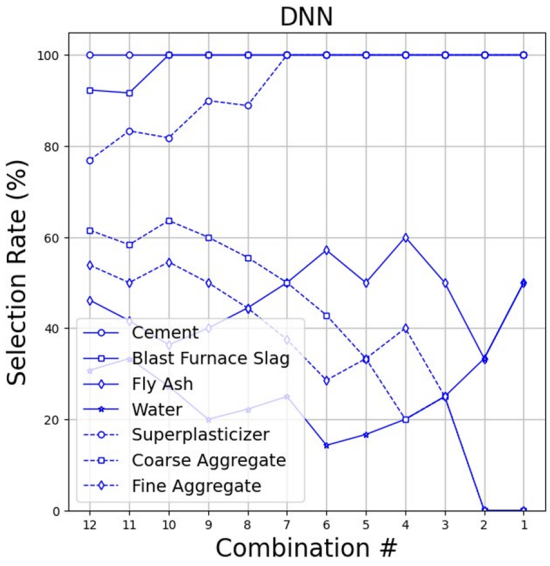

Figure 5. Selection Rate

2.8 Use Best Combination

Determined by the combination optimization, the input variables of cement, blast furnace slag, superplasticizer, and water had the most impact. Thus, the other variables were removed from the total dataset. The modified dataset was run through a DNN, which was structured similarly to the second DNN, to determine the difference, if any, in estimated strength. Likewise, Figure 6 displays the model loss vs. the model's number of epochs, and Figure 7 indicates the training vs the estimated data within the calibrated domain of [0,1]. Comparing the two figures with the output from the second DNN, the results have some differences. First, the training loss value for this DNN (0.0053) is higher compared to the loss value of the Second DNN (0.0045). This can be validated by the additional number of output points further away from the bisector line, thus, the new combination of variables resulted in a worse outcome. Since the loss figure converges closer to zero and most of the output points are close to the bisector line. We determined that the variables the combination optimization selected were correct.

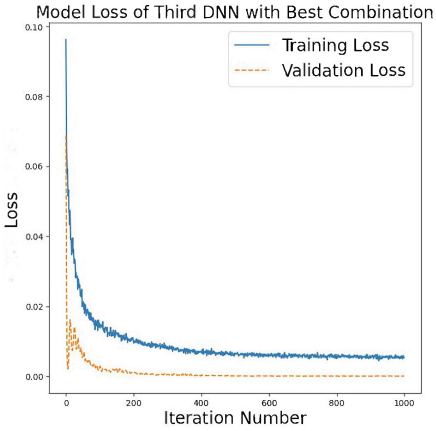

Figure 6. Training and Validation Loss with Best Combination

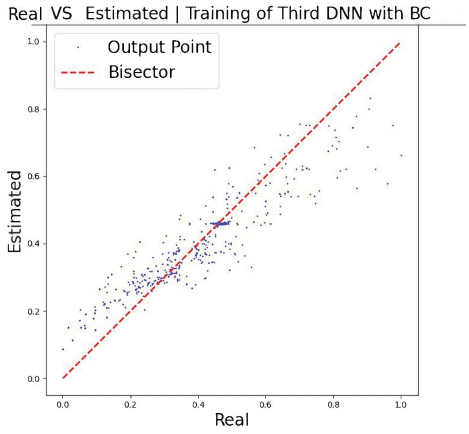

Figure 7. Real vs Estimated with Best Combination

2.9 Optimization

Finally, the optimization problem can be formulated and solved to achieve the overall goal of the paper: determining a concrete mixture that will maximize strength while minimizing weight. The objective function and associated constraints can be defined as:

$$\max \quad \frac{28-day\ compressive\ strength}{mix\ weight}$$

$$s.t.$$

$$2300 \leq weight \leq 2500 \ \frac{kg}{m^3}$$

$$0.48 \leq \frac{water}{cement} \leq 0.59$$

However, the optimization was solved using the Scipy minimize package, which requires the problem to be formulated as a minimization problem. Additionally, constraints must be rewritten in a format that the package recognizes, thus, the problem can be reformulated as:

$$\min \quad \frac{mix\ weight}{28-day\ compressive\ strength}$$

$$s.t.$$

$$weight - 2300 \geq 0$$

$$2500 - weight \geq 0$$

$$\frac{water}{cement} - 0.48 \geq 0$$

$$0.59 - \frac{water}{cement} \geq 0$$

The input into the optimization model is a vector $\boldsymbol{X}$, with 7 entries. Each entry describes how much of a particular component is added to the cement mixture. The components are cement, blast furnace slag, fly ash, water, superplasticizer, coarse aggregate and fine aggregate, in order. For example, the 4th element of $\boldsymbol{X}$, denoted as $\boldsymbol{X}_4$, would be the amount of water in the mixture.

It is important to note that the input vector $\boldsymbol{X}$ is a calibrated data point. Before training the neural networks, the original data was scaled as described in section 2.2 such that all input data points are some value between 0-1. This calibrated data was used to train all models described above. Thus, the 28-day compressive strength of the mixture, denoted as S, can be found from the calibrated input $\boldsymbol{X}$ using the trained model defined in section 2.8. However, values for weight in the original (non-calibrated) domain are meant to satisfy the constraints. Thus, the non-calibrated input x should be calculated from the calibrated input $\mathbf{X}$ using the built-in inverse transform function from sklearn. This non-calibrated data is used for the weights in the constraints and to calculate the total weight of the mixture in the objective

function. Using this representation, the problem can be reformulated once more with respect to the calibrated input $\boldsymbol{X}$ and the calculated non-calibrated datapoint x:

$$\min \quad \frac{\sum_{i=1}^{7} x_i}{S}$$

$$s.t.$$

$$\sum_{i=1}^{7} x_i - 2300 \geq 0$$

$$2500 - \sum_{i=1}^{7} x_i \geq 0$$

$$\frac{x_4}{x_1} - 0.48 \geq 0$$

$$0.59 - \frac{x_4}{x_1} \geq 0$$

$$0 \leq \boldsymbol{X}_i \leq 1 \quad \forall\, i = 1, 2, \ldots, 7$$

The initial calibrated input datapoint was randomly generated (7 random values between 0-1), and the optimization method used was 'SLSQP.' After running the optimization, it was determined that the objective function value, or the minimum ratio of mix weight to 28-day compressive strength, is 31.61 kg/MPa. Additionally, the optimal mix weight is 2300 kg, which provides a concrete with strength 72.77 MPa. The composition of the optimal mixture is shown in Table 5.

Component	**Calibrated Mix (*X* output)**	**Non-Calibrated Mix (kg/m^3 of mix)**
Cement	0.692	405.204
Blast Furnace Slag	0.767	275.701
Fly Ash	0.006	1.231
Water	0.581	194.948
Superplasticizer	0.246	7.926
Coarse Aggregate	0.052	818.831
Fine Aggregate	0.007	596.610

Table 5: Optimization results: optimal concrete mix

3.0 Discussion

3.1 Maximization Strategy

The overall goal is to maximize $\frac{28-day\ compressive\ strength}{mix\ weight}$, but in the optimization model this was reformulated as a minimization of $\frac{mix\ weight}{28-day\ compressive\ strength}$. This strategy is acceptable because by minimizing the inverse of the ratio, the original ratio is still maximized. In other words, the maximization of the ratio requires the strength to be large and the weight to be small, and the minimization of the inverse also requires the strength to be large and the weight to be small. Alternatively, when reformulating a maximization problem as a minimization problem, a widely accepted strategy is negating (e.g., to make negative) the objective function. Therefore, another objective function could be the minimization of $-\frac{28-day\ compressive\ strength}{mix\ weight}$. However, the interpretation of this solution would be less straightforward because a negative value of strength or weight is not possible, and thus a solution of a negative ratio is not intuitive. Therefore, it is more logical to use the strategy discussed in section 2.9, where the objective function is the minimization of $\frac{mix\ weight}{28-day\ compressive\ strength}$.

3.2 New Data Generation Strategy

Large quantities of training data are almost always required when training neural networks. Generally, the more training data the better with a caveat being that the training data is valid. For our analysis, 575 new data sets were generated using our trained GAN network and a neural network was trained using an augmented training set consisting of the original training data plus the newly generated training data. The GAN network was trained using the original set of 425 data sets of 28-day concrete property. Using our trained GAN model to generate new training data was one way to increase the likelihood that the statistical distribution and underlying data distributions would be carried over to the generated data.

Using a random generation technique to produce new data would be a valid method to generate the new data but it would be very unlikely that the underlying data distribution or semantics of the data

would be captured. Using a GAN network is far more likely to generate new data with the same statistical properties as the underlying training data since GANs are designed to learn and generate data samples that closely match the distribution of real data. There are no doubt types of problems where randomly generated training data would be acceptable, if, for example, the range of training data was very narrow such as suggested in the question statement (e.g., between 0 and 1). Given the detail of our data set, using randomly generated training data may work for this particular situation where the data was scaled to be between 0 and 1, but it very unlikely any of the underlying distribution trends would be replicated with randomly generated data.

If our training data were images, GAN's would be preferred over random techniques to augment a training set. GANs have demonstrated exceptional capabilities in generating high-quality, visually appealing images that can be difficult to distinguish from real images making GAN's one of the most proficient ways to generate additional image data. Furthermore, GAN's can produce images with rich textures, details, and structures. Use of random techniques to produce quality images will be difficult because of the complexity and high dimensionality of image data. It would probably include large amounts of noise in the images.

3.3 Use of New Data

The reliability of the results of our model, had the model been trained using only the initial training data sets versus using both the initial training data and generated training data through data augmentation, would probably be improved, but this can depend on several factors. For example, if the generated training data is high quality and accurately represents the underlying data distribution, adding it to the initial training data can improve the model's performance. High-quality data augmentation can provide additional diversity and help the model generalize better to unseen examples. It should also be considered that in many cases, more data tends to lead to better model performance, at least up to a certain point. Additionally, adding generated data can help reduce overfitting if the generated data is diverse and not too similar to the initial training data. It can help the model generalize better by exposing it to a broader range of examples. Suppose the generated data is low-quality or too similar to the initial data. In

that case, it will increase the risk of overfitting since the model could learn to generate similar examples rather than capturing the underlying patterns. Data augmentation using a GAN model, for this type of data set is beneficial.

The results presented in Figure 1 for the model loss of the First DNN when compared to the model loss of the Second DNN in Figure 3 tend to support this suggestion. The model loss for the First DNN, trained with the original set of 425 data points, was 0.0111 (the 1000th Epoch). The model loss for the Second DNN, trained with the combined set of original data and generated data was 0.0045 (the 1000th Epoch), which is approximately a 60% reduction in error. This reduction in error would tend to indicate that for this particular set of data, the model was improved with a larger set of training data, including augmented data.

3.4 Computation on MARCC for Complex Model

When dealing with the MARCC environment, leveraging parallel computing to distribute workload efficiently across multiple nodes and cores can be beneficial. Utilizing multiple RRTs and RRS incentivizes running the model in parallel since several combinations are based on each pair of RRT and RRS. There can be multiple ways to structure a parallel model for the DNN used in the combination optimization portion. First, a node can be responsible for a specific RRT, so multiple DNNs will run with different initializations. Once all are complete, the results can be compiled. Another method can be model parallelism, where if the architecture allows it, the other parts of the DNN can be distributed across different nodes. In this case, the later suggestions may not be better since some nodes may not be engaged while others are.

Nevertheless, best practices should be followed to enable efficient use of computing resources. According to a JHU website guide (“Best Practices”, 2023), several steps exist. First, it is essential to understand the model can run multiple processes. In our case, the DNN was created using the Keras package through tensorflow so that we can use TensorFlow Distribute. Second, run some benchmarks, ideally for a short period (1-2 hours) and a few processes and cores. Lastly, the user can monitor the status

while the job runs, such as running “gpustat” for GPU jobs. Ideally, the efficiency percentage should be close to 80%.

3.5 Validation

The optimization solution’s estimated objective value could be validated by comparing the achieved result to the original data. The optimization model calculated a minimum ratio of mix weight to 28-day compressive strength of 31.611 kg/MPa. Figure 8 below shows a histogram of the weight to strength ratio of the concrete mixes in the original dataset. It can be seen that the minimum ratio achieved by the optimization is higher than most of the ratios of the raw data. The constraints in the optimization model may restrict these lower ratios from being achievable, however, another conclusion is that the result achieved by the model may be too high.

There could be several reasons for this potential inaccuracy. One reason is the model could have

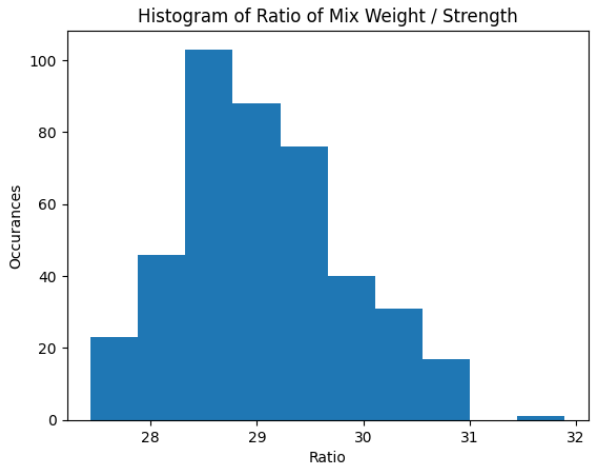

Figure 8. Ratio of Mix Weight to Strength in Data Set

overfit to the training data and given a poor estimate of the strength based on the concrete mix composition when calculating the objective function value. The GAN model developed could also be another possible source of error. We noticed that when combined with the original training data and used to train the Second DNN, the generated training data from the GAN model showed a large concentration of data located in the central portion of the Real vs. Estimated Training curve for the Second DNN (Figure 4). This led us to believe the GAN model may not be performing correctly.

4.0 Conclusion

Overall, the result achieved by the model gave a minimized weight to strength ratio of 31.61 kg/MPa, which is a maximized strength to weight ratio of 0.032 MPa/kg. This maximized ratio could be more optimal. Furthermore, the model loss for the Third DNN should have been the lowest, yet was not. The model loss going from the First DNN to the Second DNN was reduced significantly. However, as previously mentioned, the model loss from the Second DNN to the Third DNN increased by about 18% when the loss should have been reduced. This can partly be explained by the fact that only the model losses for the very last Epoch's were compared instead of taking the average of maybe the last 50 Epochs.

The following section discusses the positive and negative components of the model that achieved this result and provides some further directions.

4.1 Pros

- Collecting real-world data for concrete strength can be time-consuming and costly, as it may require conducting physical experiments on various concrete mixtures. Using a GAN to augment the dataset reduced the need for extensive data collection, potentially saving time and resources.
- The neural network trained with the augmented dataset improved generalization capabilities since it better captures the underlying patterns and factors affecting concrete strength are better captured, leading to more accurate predictions for new and unseen concrete compositions.
- The DNNs achieved better performance in optimizing concrete strength with the more diverse and expanded dataset.

4.2 Cons

- Optimizing the hyperparameters for both the DNNs and the GAN was time-consuming and training and evaluating deep neural networks required substantial computational resources, including GPUs or TPUs.

- Incorporating an additional machine learning technique, such as Random Forest, may have improved the results of the generated data coming from GAN model.

4.3 Future Directions

- One potential area of improvement is the initial guess for the optimization model. In the present model, the initial guess is randomly generated, but other methods could be used to start from a better initial guess, such as using a datapoint in the original data with a low weight to strength ratio. Another option is to run the model several times for various randomly generated initial data points and analyze the variation in the results. The optimization model could also be improved by using another optimization algorithm aside from the one used here, which was 'SLSQP' or Sequential Least Squares Quadratic Programming.
- A training set of 425 data points may not have been sufficiently large to train the GAN model. Using a more extensive set of initial training should be considered.
- Given that the newly generated data points from the GAN model were clustered together, another method of data generation could be considered to improve the training data for the final neural network, which was used for optimization. For example, Kernel Density Estimation (KDE) is a method used to estimate the underlying probability density function (PDF) of a dataset. New data points can then be obtained by sampling from the estimated density. This may allow for a newly generated dataset that more accurately follows the distribution of the original data.

5.0 References

Yeh, I-Cheng. (2007). *Concrete Compressive Strength*. UCI Machine Learning Repository. https://doi.org/10.24432/C5PK67.

Best practices. ARCH Advanced Research Computing. (2023). http://www.arch.jhu.edu/best-practices/

6.0 Group Member Contributions

Charles Middleton: Code: GAN model development and obtaining the 575 additional training points; report sections: Abstract, Introduction, GAN model, Pros, Cons, Future Directions, editing.

Jonathan Hwang: Code: Second DNN, Combination pattern recognition model, DNN with best combination; Report: Second DNN and combination step discussion, question 10e.

Stephan Jean:Code: First DNN, Initial Data processing, Data scaling, Group debugging; Report: First DNN result plots and discussion

Cecilia Doyle: Code: optimization model, group debugging; Report: abstract, intro, optimization related portions (2.9, 3.1, 3.5), future directions

Appendix

First Deep Neural Network

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                512

 dropout (Dropout)           (None, 64)                0

 dense_1 (Dense)             (None, 32)                2080

 dropout_1 (Dropout)         (None, 32)                0

 dense_2 (Dense)             (None, 16)                528

 dropout_2 (Dropout)         (None, 16)                0

 dense_3 (Dense)             (None, 8)                 136

 dropout_3 (Dropout)         (None, 8)                 0

 dense_4 (Dense)             (None, 1)                 9

=================================================================
Total params: 3265 (12.75 KB)
Trainable params: 3265 (12.75 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

First Deep Neural Network Training

Epoch 1/1000
1/1 [==============================] - 7s 7s/step - loss: 0.2344 - val_loss: 0.1724
Epoch 2/1000
1/1 [==============================] - 0s 27ms/step - loss: 0.2137 - val_loss: 0.1551
Epoch 3/1000
1/1 [==============================] - 0s 27ms/step - loss: 0.1936 - val_loss: 0.1389
Epoch 4/1000
1/1 [==============================] - 0s 27ms/step - loss: 0.1813 - val_loss: 0.1241
Epoch 5/1000
1/1 [==============================] - 0s 27ms/step - loss: 0.1635 - val_loss: 0.1104
Epoch 996/1000
1/1 [==============================] - 0s 29ms/step - loss: 0.0109 - val_loss: 0.0059
Epoch 997/1000
1/1 [==============================] - 0s 30ms/step - loss: 0.0132 - val_loss: 0.0061
Epoch 998/1000

1/1 [==============================] - 0s 34ms/step - loss: 0.0122 - val_loss: 0.0063
Epoch 999/1000
1/1 [==============================] - 0s 28ms/step - loss: 0.0103 - val_loss: 0.0064
Epoch 1000/1000
1/1 [==============================] - 0s 26ms/step - loss: 0.0111 - val_loss: 0.0065

GAN Model

```
Model: "Generator_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_109 (Dense)           (None, 32)                256

 dense_110 (Dense)           (None, 64)                2112

 dense_111 (Dense)           (None, 16)                1040

 dense_112 (Dense)           (None, 7)                 119

=================================================================
Total params: 3527 (13.78 KB)
Trainable params: 3527 (13.78 KB)
Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________
Model: "Discriminator_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_113 (Dense)           (None, 32)                   256

 dropout_43 (Dropout)        (None, 32)                0

 dense_114 (Dense)           (None, 16)                528

 dropout_44 (Dropout)        (None, 16)                0

 dense_115 (Dense)           (None, 8)                 136

 dropout_45 (Dropout)        (None, 8)                 0

 dense_116 (Dense)            (None, 1)                    9

=================================================================
Total params: 929 (3.63 KB)
Trainable params: 929 (3.63 KB)
Non-trainable params: 0 (0.00 Byte)

_________________________________________________________________
Model: "GAN"
_________________________________________________________________
```

```
 Layer (type)                Output Shape              Param #
=================================================================
 Generator_model (Sequentia  (None, 7)             3527
 l)

 Discriminator_model (Seque  (None, 1)             929
 ntial)

=================================================================
Total params: 4456 (17.41 KB)
Trainable params: 3527 (13.78 KB)
Non-trainable params: 929 (3.63 KB)
```

GAN Model Training

Epoch 01 | Batch 01 | D__loss=0.711 | G_loss=0.702
Epoch 01 | Batch 02 | D__loss=0.703 | G_loss=0.701
Epoch 01 | Batch 03 | D__loss=0.703 | G_loss=0.698
Epoch 01 | Batch 04 | D__loss=0.698 | G_loss=0.695
Epoch 01 | Batch 05 | D__loss=0.697 | G_loss=0.693
Epoch 500 | Batch 01 | D__loss=0.699 | G_loss=0.654
Epoch 500 | Batch 02 | D__loss=0.698 | G_loss=0.655
Epoch 500 | Batch 03 | D__loss=0.698 | G_loss=0.654
Epoch 500 | Batch 04 | D__loss=0.698 | G_loss=0.657
Epoch 500 | Batch 05 | D__loss=0.696 | G_loss=0.653

Second Deep Neural Network

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_13 (Dense)            (None, 56)                448

 dropout_7 (Dropout)         (None, 56)                0

 dense_14 (Dense)            (None, 32)                1824

 dropout_8 (Dropout)         (None, 32)                0

 dense_15 (Dense)            (None, 16)                528

 dropout_9 (Dropout)         (None, 16)                0

 dense_16 (Dense)            (None, 8)                 136

 dropout_10 (Dropout)        (None, 8)                 0

 dense_17 (Dense)            (None, 1)                 9

=================================================================
Total params: 2945 (11.50 KB)
```

```
Trainable params: 2945 (11.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Second Deep Neural Network Training

```
Epoch 1/1000
2/2 [==============================] - 2s 131ms/step - loss: 0.2109 - val_loss: 0.2122
Epoch 2/1000
2/2 [==============================] - 0s 24ms/step - loss: 0.1934 - val_loss: 0.2012
Epoch 3/1000
2/2 [==============================] - 0s 24ms/step - loss: 0.1761 - val_loss: 0.1819
Epoch 4/1000
2/2 [==============================] - 0s 24ms/step - loss: 0.1587 - val_loss: 0.1528
Epoch 5/1000
2/2 [==============================] - 0s 24ms/step - loss: 0.1384 - val_loss: 0.1234
Epoch 996/1000
2/2 [==============================] - 0s 27ms/step - loss: 0.0050 - val_loss: 1.1117e-04
Epoch 997/1000
2/2 [==============================] - 0s 26ms/step - loss: 0.0046 - val_loss: 9.7315e-05
Epoch 998/1000
2/2 [==============================] - 0s 26ms/step - loss: 0.0045 - val_loss: 9.1510e-05
Epoch 999/1000
2/2 [==============================] - 0s 24ms/step - loss: 0.0044 - val_loss: 8.4029e-05
Epoch 1000/1000
2/2 [==============================] - 0s 26ms/step - loss: 0.0045 - val_loss: 6.5590e-05
```

Combination Pattern Recognition Model

Best Combinations and Their Corresponding MSEs

```
-----------------------------------------------
```

	Cement	Blast Furnace Slag	Fly Ash	Water	\
DNN Training	1	1	0	1	
DNN Testing	1	1	0	1	

	Superplasticizer	Coarse Aggregate	Fine Aggregate	\
DNN Training	1	0	0	
DNN Testing	1	0	0	

	MSE Training	MSE Testing
DNN Training	0.006476	0.008265
DNN Testing	0.008265	0.006476

Run Time

546.9651169776917

Third Deep Neural Network Training

Epoch 1/1000
2/2 [==============================] - 1s 133ms/step - loss: 0.0962 - val_loss: 0.0688
Epoch 2/1000
2/2 [==============================] - 0s 25ms/step - loss: 0.0804 - val_loss: 0.0439
Epoch 3/1000
2/2 [==============================] - 0s 25ms/step - loss: 0.0655 - val_loss: 0.0232
Epoch 4/1000
2/2 [==============================] - 0s 25ms/step - loss: 0.0611 - val_loss: 0.0103
Epoch 5/1000
2/2 [==============================] - 0s 25ms/step - loss: 0.0576 - val_loss: 0.0043
Epoch 996/1000
2/2 [==============================] - 0s 33ms/step - loss: 0.0051 - val_loss: 6.2472e-05
Epoch 997/1000
2/2 [==============================] - 0s 28ms/step - loss: 0.0059 - val_loss: 6.1507e-05
Epoch 998/1000
2/2 [==============================] - 0s 28ms/step - loss: 0.0058 - val_loss: 7.6889e-05
Epoch 999/1000
2/2 [==============================] - 0s 29ms/step - loss: 0.0054 - val_loss: 8.5933e-05
Epoch 1000/1000
2/2 [==============================] - 0s 27ms/step - loss: 0.0053 - val_loss: 8.5053e-05

Optimization

X Solution (calibrated): [0.6922462 0.76711371 0.0061524 0.5806537 0.24614504 0.05183518
 0.00654638]

Concrete Mix (unscaled): [[405.20383438 275.70066904   1.23109426 194.49784339   7.92587016
  818.83130341 596.60938536]]

Objective Function Value (mix weight / strength): 31.611350975838324