Math 76 Final: Machine Learning for Weather Prediction

Catherine Chu, Divik Verma, Henry Moore, Paul Chirkov

August 21, 2024

Abstract

Our project uses 2 types of neural networks: Artificial Neural Networks (ANN) and Recurrent Neural Networks (RNN) to capture the relationships in Weather data. We use a dataset found from Kaggle, listed in more detail in the sources section that details time series data for Weather. Since the data is time series dependent, we used RNNs to accurately reflect the temporal dependencies and patterns in weather over time. Additionally, we also wanted to contrast the use of ANNs, which can capture relationships in tabular data by learning from the input features to predict a target variable (we chose max_temperature).

1 In the News Today

The cornerstone of modern weather prediction is Numerical Weather Prediction (NWP), which involves the use of mathematical models, mostly partial differential equations, to simulate the atmosphere's behavior. These models are based on physics, including the laws of motion, thermodynamics, and the conservation of mass and energy. NWP models typically divide the atmosphere into a three-dimensional grid, with each cell in the grid representing a specific area of the atmosphere at a given altitude. The models then calculate the changes in atmospheric variables—such as temperature, pressure, humidity, and wind speed—within each grid cell over time. To run these models, vast amounts of data are required. This data is collected from various sources, including weather stations, satellites, weather balloons, radar systems, and aircraft. The data provides initial conditions for the models, which then use partial differential equations to project how the atmosphere will evolve over time. The accuracy of NWP depends on the quality of the initial data, the resolution of the grid, and the computational power available to run the models.

As weather forecasting continues to evolve, artificial intelligence has been tested as a powerful tool that complements and enhances traditional meteorological techniques. While traditional methods of weather prediction, such as NWP, rely heavily on mathematical models grounded in the physics of the atmosphere, AI analyzes vast datasets and identifies patterns that might not be immediately evident through these conventional methods. This integration of AI into weather forecasting is not aimed at replacing traditional techniques but rather at augmenting them, making forecasts more accurate, timely, and relevant, especially in the context of extreme weather events. When using AI, no physical equations are required to create a forecasting system. Training the neural nets just requires vast amounts of weather data, from which patterns can be learned. In the past five years, the application of AI for weather forecasting has morphed from being simply an academic pursuit. Companies and weather agencies are beginning to use neural networks as a tool to aid in their forecasts.In May, Microsoft released a forecast tool called Aurora that predicts global air pollution 5000 times faster than models run by the National Oceanic and Atmospheric Administration. In

November of 2023, Google's DeepMind released GraphCast which uses graph neural networks to predict weather conditions up to 10 days in advance.

2 Current Research on NN Architectures for Weather Prediction

We were interested in testing the abilities of neural networks to predict weather for ourselves. We took inspiration from the article "Using Recurrent Neural Networks for Localized Weather Prediction with Combined Use of Public Airport Data and On-site Measurements" by Jung Min Han et al. which explores the application of Recurrent Neural Networks (RNNs) to improve weather prediction accuracy. The study addresses the limitations of traditional weather files. The authors propose a methodology that integrates public weather data with localized measurements to generate more accurate synthetic weather data. They specifically compare the performance of different neural network architectures, including Feed-Forward Neural Networks (FFNN), Long Short-Term Memory (LSTM) networks, and Gated Recurrent Units (GRU). Their results indicate that RNNs, particularly the GRU model, outperform traditional FFNNs in predicting localized weather conditions. We wanted to test different neural networks architectures and see how they would perform on different weather data for ourselves.

3 PART I: Recurrent Neural Network for Temperature Prediction

3.1 RNNs, GRUs, LSTMs, and the Vanishing Gradient Problem

The primary distinguishing feature of RNNs is that neurons can have connections that loop back to themselves, allowing the network to keep track of parts of the network state from previous inputs to mimic autoregression. In an RNN, much like a FFNN, we use a combination of weights and biases to get from one layer to the next, tuning these parameters as we train. However, in an RNN, the equation from one layer to the next includes an additional key variable: the hidden layer (and its accompanying weight matrix). At each time step, the hidden layer receives the current input, but it crucially also receives the information from the previous state of the hidden layer. The hidden layer combines these two inputs and passes them through an activation function to get the final output, which is used to calculate the next visible layer. This process allows the network to recognize relationships and context between inputs, which is essential for effectively processing sequential data with time dependencies and stochastic properties in ways that cannot be achieved by feedforward neural networks.

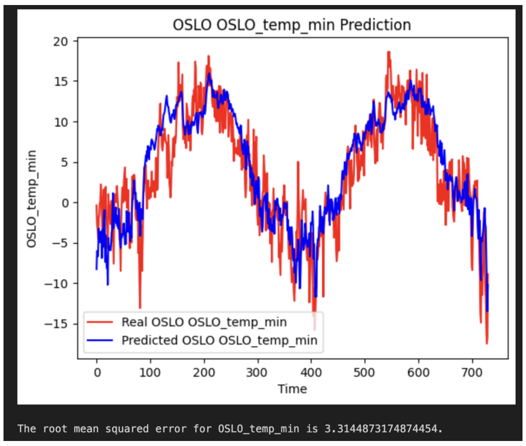
Long short-term memory (LSTM) is an architectural element that can be applied to RNNs. The LSTM unit is made up of four feedforward neural networks: forget, input, output, and memory. It receives three input vectors: the states of the memory and hidden layer from the previous instant, plus one from the RNN. First the forget gate uses the input and the previous hidden state to update the previous memory state by deleting information that it no longer deems relevant. The input gate uses the same vectors to decide which new information from the input should be added to the memory state. Finally, the output gate receives a "candidate vector" from the memory layer and uses it to create a new state of the hidden layer, which ends up as the "output" of the LSTM that gets passed back into the RNN. LSTM allows for better handling of long-term dependencies and mitigates the vanishing/exploding gradient problem latent in standard RNN architecture.

Gated recurrent unit (GRU) networks fulfill a similar role to LSTM, using gating mechanisms to selectively update the hidden state of the network at each time step. However, they combine the input and forget gates of LSTM into a single “update” gate, reducing parameters and significantly improving training efficiency and computational expense. The “reset” gate evaluates how much of the hidden state from the previous timestep should be forgotten, while the “update” gate determines how much of the new input should be used to update the hidden state. A “candidate” vector is again calculated from the hidden state, then combined with the input vector to the degree specified by the update gate.

Based on these models and techniques, we opted to train three models. The first of these was a vanilla RNN with no GRU or LSTM as something of a baseline. This model performed remarkably well to begin with, and with more epochs and some slight feature engineering to prevent overfitting, it is likely that this model could become more accurate further still. The second model was adapted with GRUs to enhance the accuracy. This had the intended effect, causing the RMSE to decrease for virtually every category of prediction. The last model we trained was a far more complicated model consisting of an RNN equipped with LSTM and GRU layers. The accuracy of these models is discussed below.

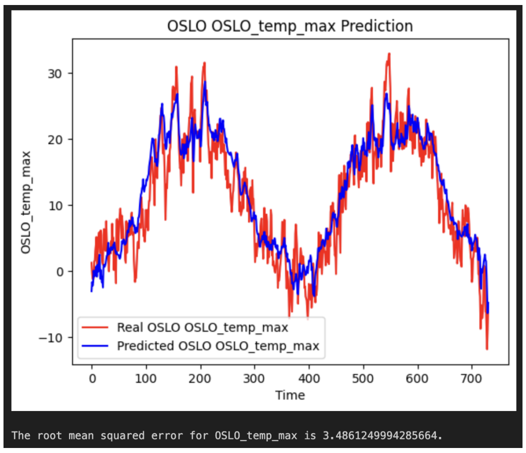
Our findings were that the accuracy of the models, in order, were: 1. RNN with GRUs - 2. Vanilla RNN - 3. RNN + LSTM + GRU. These findings are consistent with the work of Han et al (2021) who find that Gated Recurrent Units (GRU) performed best in the study with low mean squared error. Given more time, a more sophisticated analysis of the error would have been the best course of action, but the long training times necessary for RNNs required more time with hyperparameter tuning, implementation, and training than anticipated. Models were trained on NVIDIA T4 GPUs courtesy of Google Colaboratory. Across all implementations, a few factors were held constant. An 80/20 train/test split was used, and given that a dataset contained 10 years of data, the model had 8 years of training data. In addition, the various RNNs implemented all restricted themselves to training data belonging to the same city as the test data. A baseline was established using a vanilla RNN with 50 epochs and a batch size of 32 using the Adam optimizer. Then, a model with GRUs was implemented, using the Han paper as a model. Then, in order to see if further model complexity could capture even more trends in the data, a model with GRUs and LSTM was implemented.

An explanation of the results can be found in the structure of the data. Due to the fact that the GRUs have fewer parameters, it is possible that it prevents overfitting, which is likely one factor that contributed to the worsened accuracy in the case of the GRU+LSTM model. Furthermore, neural network architectures are meant to model dependencies that are already present in the data. For this reason, a bidirectional RNN would not be particularly useful, for example, as there are very few dependencies between future weather data that affects current weather data. Additional changes that could be make to further improve the GRU model’s accuracy are to increase the variance of the data further, or to investigate trends in the data that cause the model to be slightly more accurate with some cities as compared to others. It would be interesting to investigate whether all the models struggle with certain cities, as this would provide insights into the nature of the challenge facing these models. Examples of the GRU model performing particularly well are showcased below, along with the loss and validation loss as a function of epoch. Additionally, the attached code shows analogous plots for each city for each of the three RNN models trained.



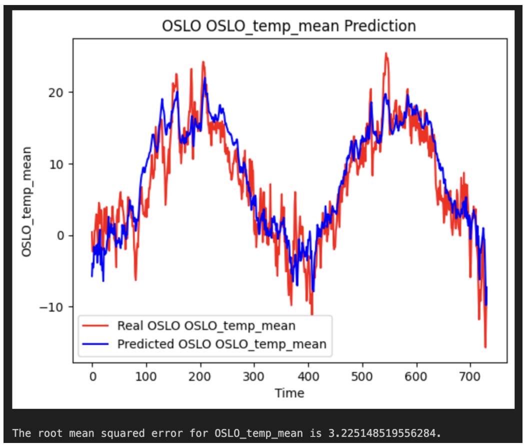
```
The root mean squared error for OSLO_temp_min is 3.3144873174874454.
```

(a) Oslo Minimum Temperature Predictions vs Actual



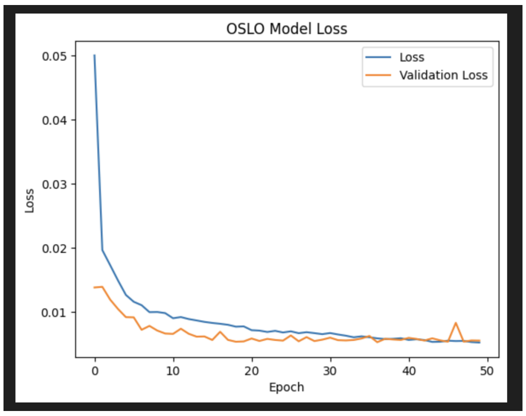
```
The root mean squared error for OSLO_temp_max is 3.4861249994285664.
```

(b) Oslo Maximum Temperature Predictions vs Actual



```
The root mean squared error for OSLO_temp_mean is 3.225148519556284.
```

(a) Oslo Mean Temperature Predictions vs Actual



(b) Loss and Validation Loss as a Function of Epoch

4 PART II: Artificial Neural Network and Convolutional Neural Network for Maximum Temperature Prediction in Heathrow London

4.1 Abstract

The purpose of this section of the project is to predict the maximum temperature at Heathrow by analyzing various weather-related variables including humidity, pressure, cloud cover, precipitation, sunshine, temperature mean, and temperature minimum. By leveraging an artificial neural network, the code that was written will model the complex relationships between these weather variables to make accurate predictions about maximum temperature. The structure of this paper will start from the first neural network that we tried, followed up by the utilization of hyperparameter tuning methods before a gradient boosting algorithm. The results of our data indicate that gradient boosting produces the most accurate test data predictions for the temperature maximum in Heathrow when compared to an artificial neural network before tuning and even an artificial neural network that's been tuned based on the gradient descent learning rate.

4.2 Motivation

The motivation behind this section of the project is twofold. Firstly, weather is an important factor in all of this group's hobbies. For example, before big hikes, golfing events, or bike races, the weather is a key indicator of how the day will go. Thus, our group was interested in seeing if we could train a neural network to predict the maximum temperature. The second motivation was that we believe neural networks are an important skill that will likely be highly applicable to a great variety of fields that our group will go on to pursue beyond college.

4.3 Main Findings

This section covers the theory behind our choices of neural networks, followed by a comparison of those theories to our results. Theoretically, the code listed in weather.ipynb works by feeding multiple weather-related features into a neural network, which will then learn the patterns and relationships between these features and then output its prediction for the maximum temperature. We first chose to use an ANN (given that we already thought that the RNN would have the greatest accuracy since RNNs are more typically used for temporal dependent data) because ANNs can capture nonlinear relationships and interactions between input variables, which matches the data set that we are given. By training on historical weather data, the network adjusts its internal parameters to minimize the difference between its predictions and the actual recorded maximum temperatures. Once the ANN model has been trained, the model can then be used to predict future maximum temperatures based on new weather data.

4.4 Explanation of Results

Our results indicate that a gradient-boosting algorithm is much better at predicting the maximum temperature at Heathrow than an artificial neural network. These conclusions are derived from the plots that indicate the actual value versus the predicted values in addition to the residual plots. Gradient boosting is probably more effective in this case because it was able to capture a complex, nonlinear relationship through weaker decision trees, which can build on correcting the errors of the previous trees. Ultimately, the ANN wasn't as effective likely because of the more complex parameter tuning, including learning rates and the number of layers and dropout layers, all of which we had tried but didn't produce better results.

4.5 Methods

1) The first step of this project involved selecting the feature we wanted to predict and the city that we were interested in observing. Preliminarily, we selected 5 cities: Heathrow, Oslo, Stockholm, Sonnblick, and Kassel. These cities were chosen because in the dataset they had all the columns filled out and the least amount of data missing. Note that all the temperatures that will be plotted below are given in units of Celsius. We started by plotting the maximum temperature for data from 2000 to 2010, which is the timeframe that we have data for. The results are displayed below.

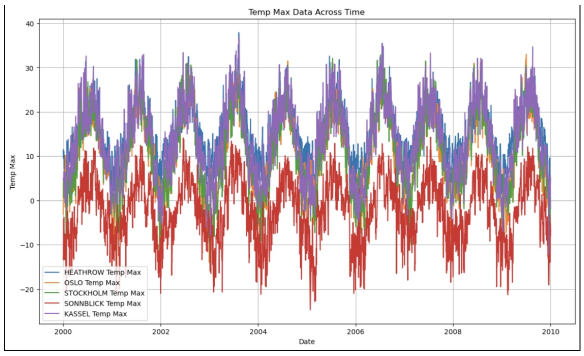


Figure 3: Maximum temperatures for various cities over time

This data indicates that besides Sonnblick, the temperature max fluctuations between Heathrow, Oslo, Stockholm, and Kassel are all relatively similar. Since Catherine had just visited Heathrow over winter break, we chose to train the neural network on the Heathrow data and predict the temperature maximum of Heathrow throughout time. Ideally, however, the results of Heathrow can be extrapolated to other locations because of how similar their fluctuations are to one another.

The second step in this process was reproducing some of the data that was found from the original dataset that's been linked in the dataset section. The original data corresponded to the times that people would barbeque in each of these locations at different times of the year. Their paper also discussed their findings with a correlation heatmap for Dresden. We decided to reproduce their data for Heathrow.

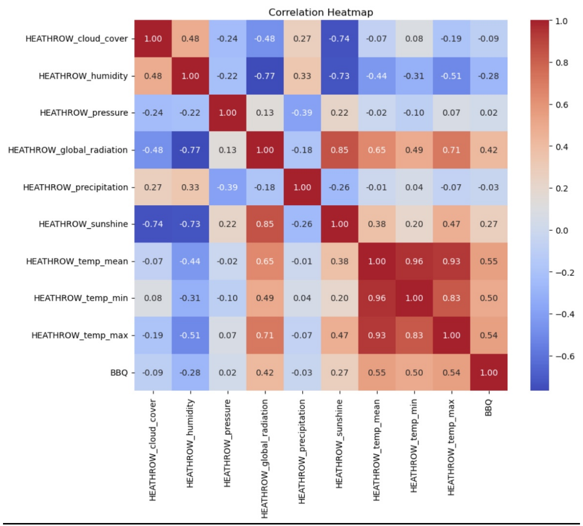


Figure 4: Feature correlations - note the temperature correlations

This image strongly corresponds with the one for Dresden that was provided by the previous research paper in which we derived the dataset to us, ensured that we had cleaned the data effectively, and that we had all the data we needed to start training our neural networks.

Here, we started writing the majority of the code for this project to train an ANN for Heathrow to predict the temperature maximum on any given day. We sorted through the dataset to clean and isolate the data needed. Then we loaded the data into Pytorch. We had originally tried using TensorFlow, another package that had been used in much of the literature that we had read. However, TensorFlow conflicts with many of the packages that come with macOS, so we decided to use Pytorch instead since it was more compatible with our kernel environments.

We defined our ANN, called WeatherANN, then trained 100 epochs in groups of 10 and determined the associated loss values. The results were as follows: Epoch [10/100], Loss: 0.7956; Epoch [20/100], Loss: 0.7302; Epoch [30/100], Loss: 0.7045; Epoch [40/100], Loss: 0.6908; Epoch [50/100], Loss: 0.6729; Epoch [60/100], Loss: 0.6593; Epoch [70/100], Loss: 0.6559; Epoch [80/100], Loss: 0.6472; Epoch [90/100], Loss: 0.6413 Epoch [100/100], Loss: 0.6220. From these results, we can see that the training loss decreases gradually over the 100 epochs. It starts at 0.7956 for the first epoch of 10, but drops to 0.6220 by epoch 100. This decrease reflects the training process of the artificial neural network, since the model learns from each of the epochs and gradually improves its ability to predict the maximum temperature from the training data. As the model trains, it will keep adjusting to better predict the maximum temperature. The test loss is 0.742, which is higher than the final training loss at 0.6220. This means that there is a significant difference between how well it performs in training data versus test data. If there's a higher test loss, that means that the model is probably underfitting, which is why we decided to tune the model to try fitting again.

Our first attempt to improve the accuracy and minimize the test loss in our model was by setting and testing different learning rates on a log scale from 10e-2 to 10e-5 to see if we could get higher accuracy rates for the maximum temperature. The results of the test loss compared to the learning rate are attached below.

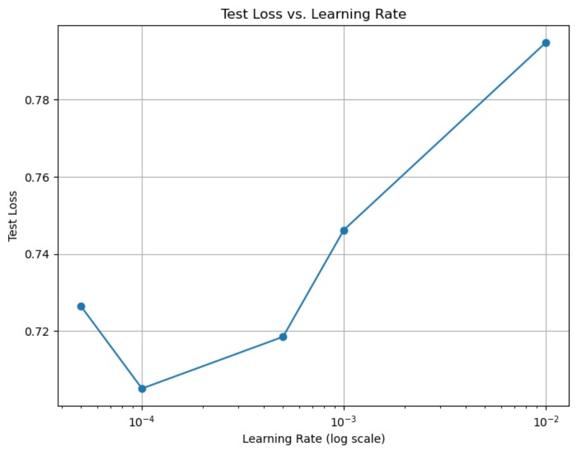


Figure 5: Test Loss vs Learning Rate

These results tell us that there is a better learning rate at 0.0001.

Before our tuning, we wanted to plot 3 ways of verification to see the training loss versus the test loss, the predicted values vs. the actual values, and the residual plot. Below are the results.

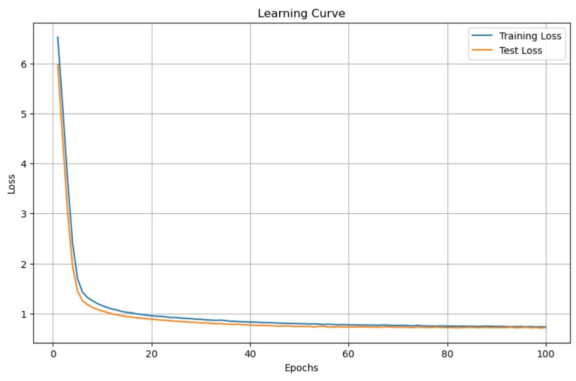


Figure 6: Learning Curve - test and training loss vs epoch

The learning curve shows the loss for both the training and test sets over 100 epochs. Since both of them decrease rapidly initially and then stabilize, it means that this model is probably not overfitting and is generalizing decently to unseen data.

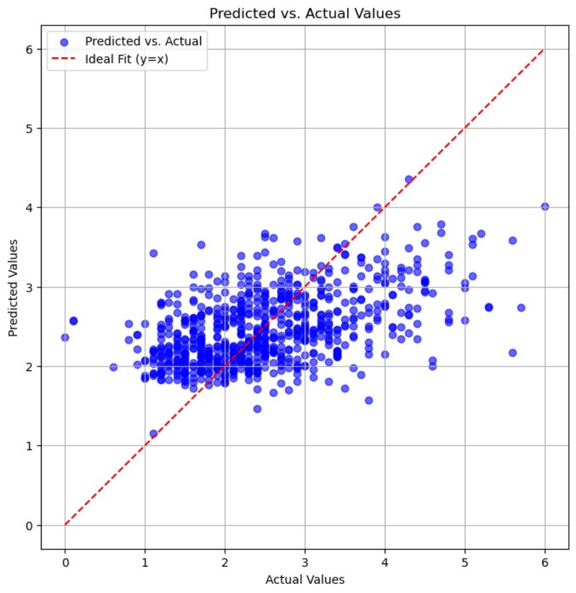


Figure 7: Predicted vs. Actual Values

This second plot demonstrates the predicted versus actual values. Ideally, the data is supposed to follow the red line which would mean that the data is matching perfectly. This shows us that the points are mostly around 2 to 4, but they do not change with respect to the ideal fit. Our model was evidently not well-trained here.

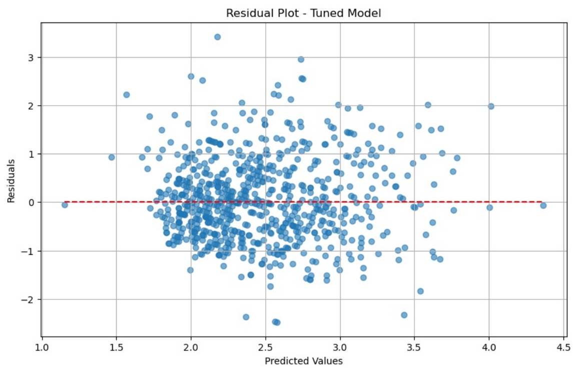


Figure 8: Residual Plot - Tuned Model

This plot shows the residuals. Ideally, they should be randomly scattered around zero because that would mean that the model is not biased, so it doesn't fall in the category of overfitting. However, since there is a really big spread of the residuals with lots of outliers, it means this model has significant errors.

6) We tried to tune the ANN in the next step by using two methods. First, we increased the numbers of layers and neurons. Our intent was for the model to learn to recognize more complex relationships in the data. We also added dropout, a regularization method intended to help us

prevent overfitting. In each additional layer, the model dropout will randomly set a fraction of the input units to 0 during training so that the network will learn more about different features. We chose a dropout rate of 20%, which is what we had seen in other research papers. We also adjusted the learning rate based on the previous learning rate optimization we had done. The results, however, were slightly disappointing. Below are the predicted versus actual values.

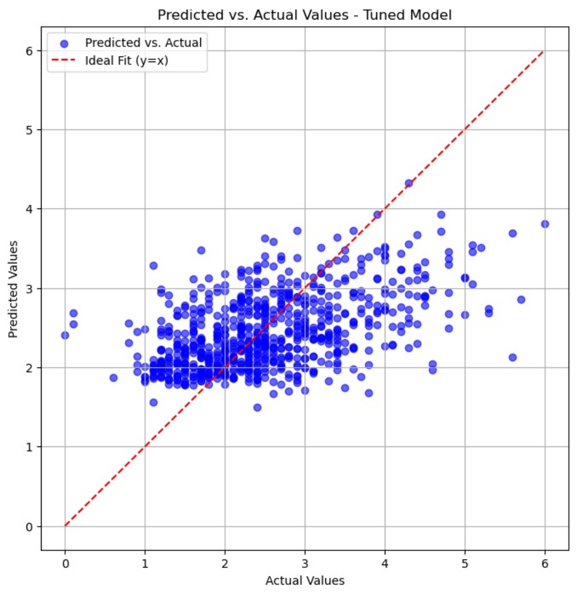
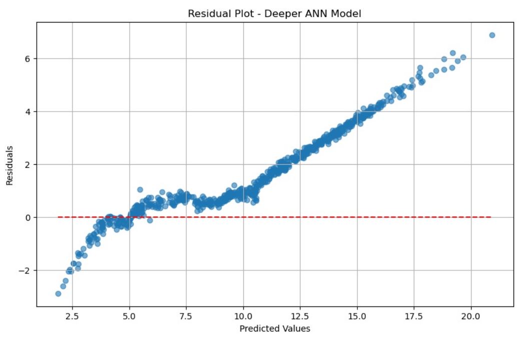
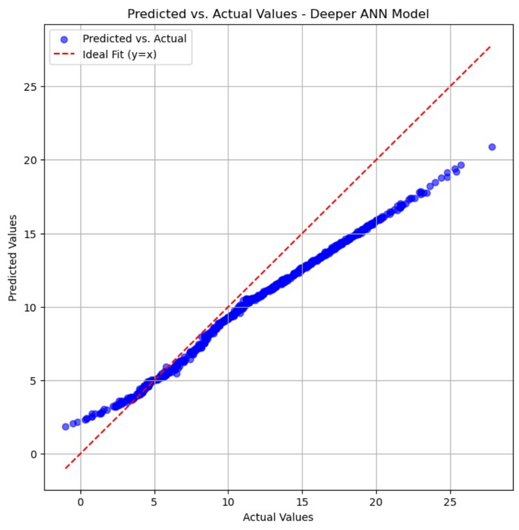


Figure 9: Predicted vs. Actual Values - Tuned Model

The results don't seem to be much better, which is why we decided to pivot and try a deeper ANN which upped the number of layers from 4 to 5. Because overfitting becomes a greater concern with deeper networks, we increased the dropout rate to 30%. We still used 100 epochs with the Adam optimizer. We see better results here.



(a) Residual Plot - Deeper ANN Model

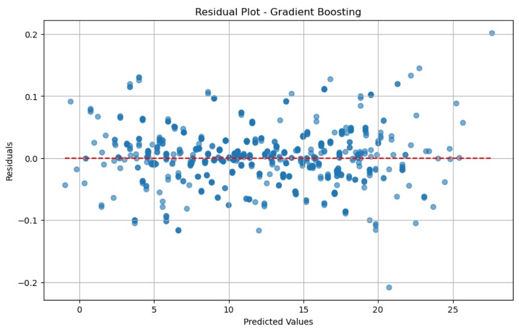


(b) Predicted vs. Actual Values - Deeper ANN Model

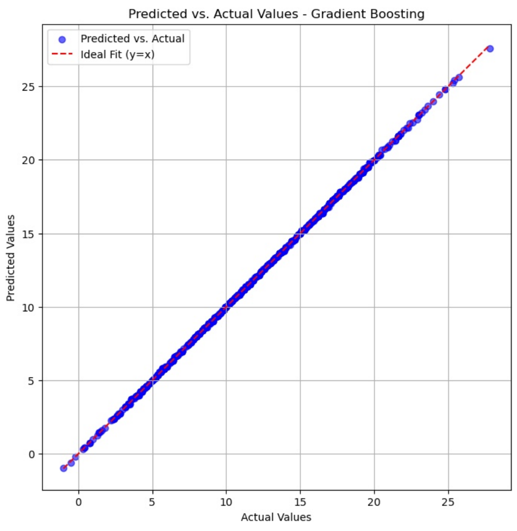
Figure 10: Overall caption for both images

At a first glance, it seems as if the predicted and actual graph looks much better. However, upon analyzing the residual plot of the deeper ANN results, we realize that the residual error increases greatly when the predicted values get higher systematically. The residuals are also NOT randomly distributed, which is what we would expect. Rather, the residuals show that the model is likely underfitting the higher temperature range.

In the final part, we used a gradient boosting method. It used decision trees to form a strong predictor. We initialized the gradient booster with a max depth of 3 to prevent overfitting. We chose to use gradient boosting in hopes of achieving a higher predictive accuracy. We used gradient boosting to predict the maximum temperature at Heathrow based on the other features. Then, we evaluated the performance using MSE.



(a) Residual Plot - Gradient Boosting



(b) Predicted vs. Actual Values - Gradient Boosting

These results show that gradient boosting is very good at predicting the maximum weather in Heathrow because it follows the line almost exactly. On top of that, the residual plot being randomly scattered around 0 is also very promising because it likely indicates that we aren't overtraining the data. Another way that we mitigated overtraining is because we are using real data that's hidden in the test split at the beginning to test our model. The mean cross-validation MSE is 0.0044, which is extremely low, meaning this model has been highly accurate.

5 What did we learn?

This project has been a good experience for the whole group. Catherine came into the project wanting to learn about neural networks and from this project, she gained experience working with CNNs, and ANNs, especially after the CNN didn't work. Henry found it interesting that the gradient boosting algorithm was more effective than the ANN which came as a surprise for all of us. His takeaway is that sometimes for smaller datasets, we should stick to a simpler model. Divik found it interesting that a complex model is not always the better solution, and that simpler models can be far more effective, reduce training time, and simultaneously increase performance and efficiency. Furthermore, Divik found it interesting that the best models capture structure that exists within the data, and that adding architectures for dependencies not present in the data will worsen performance. Paul found it interesting learning about how neural nets could be used to supplement traditional methods of numerical weather prediction. He also enjoyed learning about the ways in which Gru neural networks work.

Sources:
https://www.kaggle.com/datasets/thedevastator/weather-prediction/code
https://www.sciencedirect.com/science/article/pii/S0360132321000160#sec3

Author Contributions:
Catherine Chu: authored ANN code
Divik Verma: authored RNN code
Henry Moore: assisted with ANN code, report
Paul Chirkov: assisted with RNN code, report