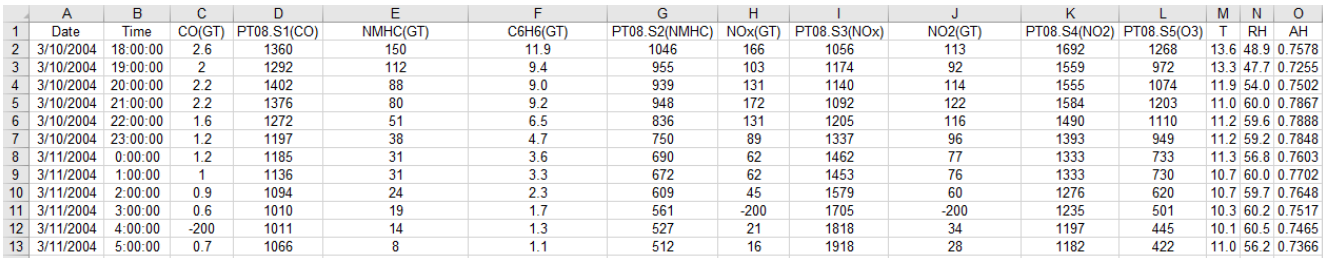
Time Series Air Quality Prediction with Neural Networks

Hai Duong Nguyen

May 23, 2025

1 Introduction

This report documents the development and evaluation of two neural network models—one for classification and one for regression—using a modified version of the "Air Quality" dataset originally available from the UCI Machine Learning Repository UCI Machine Learning Repository (2008). The dataset contains 8,358 instances of hourly averaged responses collected from an air quality monitoring device deployed at a polluted roadside location in an Italian city between March 2004 and February 2005. Ground truth concentrations for various air pollutants (e.g., CO, NOx, NMHC, Benzene) were obtained from a certified co-located reference analyser. Missing values are indicated using the placeholder value `-200`.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Date	Time	CO(GT)	PT08.S1(CO)	NMHC(GT)	C6H6(GT)	PT08.S2(NMHC)	NOx(GT)	PT08.S3(NOx)	NO2(GT)	PT08.S4(NO2)	PT08.S5(O3)	T	RH	AH
2	3/10/2004	18:00:00	2.6	1360	150	11.9	1046	166	1056	113	1692	1268	13.6	48.9	0.7578
3	3/10/2004	19:00:00	2	1292	112	9.4	955	103	1174	92	1559	972	13.3	47.7	0.7255
4	3/10/2004	20:00:00	2.2	1402	88	9.0	939	131	1140	114	1555	1074	11.9	54.0	0.7502
5	3/10/2004	21:00:00	2.2	1376	80	9.2	948	172	1092	122	1584	1203	11.0	60.0	0.7867
6	3/10/2004	22:00:00	1.6	1272	51	6.5	836	131	1205	116	1490	1110	11.2	59.6	0.7888
7	3/10/2004	23:00:00	1.2	1197	38	4.7	750	89	1337	96	1393	949	11.2	59.2	0.7848
8	3/11/2004	0:00:00	1.2	1185	31	3.6	690	62	1462	77	1333	733	11.3	56.8	0.7603
9	3/11/2004	1:00:00	1	1136	31	3.3	672	62	1453	76	1333	730	10.7	60.0	0.7702
10	3/11/2004	2:00:00	0.9	1094	24	2.3	609	45	1579	60	1276	620	10.7	59.7	0.7648
11	3/11/2004	3:00:00	0.6	1010	19	1.7	561	-200	1705	-200	1235	501	10.3	60.2	0.7517
12	3/11/2004	4:00:00	-200	1011	14	1.3	527	21	1818	34	1197	445	10.1	60.5	0.7465
13	3/11/2004	5:00:00	0.7	1066	8	1.1	512	16	1918	28	1182	422	11.0	56.2	0.7366

Figure 1: Dataset

1.1 Dataset Description

The dataset comprises sensor responses from five metal oxide chemical sensors, along with ground truth pollutant concentrations and meteorological variables. A summary of the key variables is shown in Table 1.

Table 1: Key variables in the dataset

Variable	**Description**
CO(GT)	True hourly average of carbon monoxide (mg/m^3)
NMHC(GT)	True hourly average of non-methane hydrocarbons (µg/m^3)
C6H6(GT)	True hourly average of benzene (µg/m^3)
NOx(GT)	True hourly average of nitrogen oxides (ppb)
NO2(GT)	True hourly average of nitrogen dioxide (µg/m^3)
PT08.S1–S5	Sensor responses for various gases
T, RH, AH	Temperature, Relative Humidity, Absolute Humidity

1.2 Task Overview

Two machine learning tasks are addressed in this assignment:

- **Classification Task:** Develop a neural network to classify whether the concentration of carbon monoxide (CO(GT)) exceeds its mean value (excluding missing values). This is a binary classification problem: the model must predict whether each instance falls above or below the calculated threshold.
- **Regression Task:** Construct a neural network to estimate the concentration of nitrogen oxides (NOx(GT)) using other available features in the dataset. This involves learning a continuous mapping from input variables to NOx concentrations.

Both models must be capable of handling missing values within the data.

2 Literature Review

2.1 Neural Network Basics

A neural network is made of layers of interconnected nodes (neurons). Each neuron computes a weighted sum of its inputs, adds a bias, and passes the result through an activation function.
Neuron operation (forward pass):

$$z = \sum_{i=1}^{n} w_i x_i + b = \mathbf{w}^\top \mathbf{x} + b$$

$$a = \sigma(z)$$

Where:

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ are the inputs
- $\mathbf{w} = [w_1, w_2, \ldots, w_n]$ are the weights
- b is the bias
- z is the linear combination
- σ is the activation function (e.g., ReLU, Sigmoid)
- a is the neuron output

More details: `https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/`

2.2 Activation Functions

Activation functions add non-linearity, allowing the network to learn complex patterns. Common ones include:

- **Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$ — used for binary classification
- **ReLU (Rectified Linear Unit):** $\text{ReLU}(z) = \max(0, z)$ — used in hidden layers
- **Softmax:** Converts output into probability distribution (used in multi-class classification)

2.3 Loss Functions

Loss functions measure how far off the model's predictions are from actual values.
For classification (binary):

$$\text{Binary Cross-Entropy} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

Where y_i is the actual label and $\hat{y}_i$ is the predicted probability.
For regression:

- **Mean Squared Error (MSE):**
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
- **Mean Absolute Error (MAE):**
$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$
- **Root Mean Squared Error (RMSE):**
$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

2.4 Training the Network

Training means adjusting the weights and biases to minimize the loss using a method called gradient descent.
Backpropagation:

- Compute the gradient of the loss with respect to each parameter
- Update weights using:

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w}$$

  Where η is the learning rate and $\mathcal{L}$ is the loss

More on backprop: `https://www.geeksforgeeks.org/backpropagation-in-neural-network/`

2.5 Evaluation Metrics

After training, the model needs to be evaluated to ensure it's generalizing well.

2.5.1 Classification Metrics

- **Accuracy:**

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

- **Precision:**

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Confusion Matrix:** Shows TP, TN, FP, FN in a table.

2.5.2 Regression Metrics

- **MAE, MSE, RMSE** — already defined above
- These help measure how accurate the predicted numbers are compared to actual values.

More metrics: `https://scikit-learn.org/stable/modules/model_evaluation.html`

2.6 Model Implementation

The neural networks for both classification and regression are built using `Keras` (with `TensorFlow` backend). This framework makes it easier to:

- Define layers and activation functions
- Compile the model with loss function and optimizer
- Train using model.fit()
- Evaluate using built-in metrics or custom evaluation scripts

Learning Keras: `https://www.geeksforgeeks.org/keras-a-deep-learning-library/`

2.7 Data Preprocessing

Before training, the dataset was cleaned and preprocessed to ensure data quality and improve model performance.

1. Handling Missing Values

The dataset contains missing values marked as `-200`. These were replaced with `NaN` and filled using forward fill, which replaces each missing value with the last valid value before it.

$$x_t = \begin{cases} x_t, & \text{if } x_t \neq \text{NaN} \\ x_{t-1}, & \text{if } x_t = \text{NaN} \end{cases}$$

Reference: `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html`

2. Outlier Detection

Outliers were removed using the Z-score method. The Z-score measures how far a value is from the mean in terms of standard deviations:

$$Z = \frac{x - \mu}{\sigma}$$

Where:

- x is the data point
- μ is the mean of the column
- σ is the standard deviation

Points with $|Z| > 3$ were considered outliers and removed.
Reference: `https://en.wikipedia.org/wiki/Standard_score`

3. Feature Scaling

Feature values were normalized using **StandardScaler**, which transforms each feature to have zero mean and unit variance:

$$x' = \frac{x - \mu}{\sigma}$$

This is important to ensure all input features are on the same scale, especially when using gradient-based models like neural networks.
Reference: `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`

4. Data Splitting

The cleaned dataset was split into training and testing sets using an 80/20 split:

- **Training set:** used to train the model
- **Testing set:** used to evaluate performance on unseen data

Reference: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

3 Methodology

3.1 Data Preprocessing

Before training any model, it's important to clean and understand the data. Here's what preprocessing usually involves:

- **Check variable ranges:** Look at the minimum and maximum values of each feature (input and output) to understand their scale.
- **Plot each variable:** Use graphs to spot trends, seasonality, or weird values.
- **Handle missing or weird data:** In this dataset, missing values are marked as `-200`. These need to be replaced or removed.
- **Split the dataset:** Divide it into a training set and a test set (usually 80/20 or 70/30 split) so you can evaluate how well the model generalizes.

3.2 Neural Network Design

Once the data is cleaned, you need to design a neural network that can do the job. There are two tasks:

- **Classification:** Predict if CO concentration is above or below the average.
- **Regression:** Predict NOx concentration using the rest of the features.

For both tasks, you need to decide:

- Number of layers (e.g., input layer, hidden layers, output layer).
- Number of neurons per layer.
- Activation functions (e.g., ReLU, Sigmoid).
- Make sure the total number of parameters in your model is less than total samples / 10.
- Use Keras and TensorFlow to build the model.

3.3 Training the Model

Next step is to train the network so it learns from the data. Important training choices include:

- **Loss function:** For classification, use binary cross-entropy. For regression, use MSE or MAE.
- **Optimizer:** Adam is a popular choice.
- **Batch size:** How many samples the model sees at once. Common values: 32, 64.
- **Learning rate:** Controls how fast the model updates. Common starting value: 0.001.
- **Epochs:** Number of times the model goes through the whole dataset. Try starting with 100.
- Monitor the training loss to avoid **overfitting**—where the model is too good at training data but fails on new data.

3.4 Validation and Evaluation

After training, test how well the model performs on unseen data.

3.4.1 Classification Task

- Plot training and validation loss over epochs.
- Plot training and validation accuracy.
- Compute the **confusion matrix** with:
- Use `sklearn.metrics` to compute:
  - Accuracy
  - Precision
- Target accuracy and precision should be **above 85%** to get full marks.

3.4.2 Regression Task

- Plot training and validation loss over epochs.
- Plot predictions vs actual values for test data.
- Report these performance metrics:
  - **RMSE** (Root Mean Squared Error): average of squared prediction errors.
  - **MAE** (Mean Absolute Error): average of absolute prediction errors.
- Use `sklearn.metrics` to calculate them.
- Target: **RMSE** $<$ **280** and **MAE** $<$ **220** on test data to pass.

4 Code

4.1 Data Preprocessing

The dataset required multiple preprocessing steps before it could be used for training. These included identifying missing values, normalizing input features, and detecting outliers.

```python
# Replace -200 with NaN
df.replace(-200, np.nan, inplace=True)

# Identify variation range for each variable
variation_range = df.agg(['min', 'max']).T
variation_range.columns = ['Min', 'Max']
print(variation_range)

# Fill missing values using forward fill (time-series logic)
df.fillna(method='ffill', inplace=True)

# Outlier removal using Z-score method
from scipy.stats import zscore
z_scores = np.abs(zscore(df.select_dtypes(include=np.number)))
df_no_outliers = df[(z_scores < 3).all(axis=1)]
```

Listing 1: Data Cleaning and Preprocessing

4.2 Classification Model (CO Exceeds Mean)

The classification model predicts whether the CO concentration exceeds the mean. The output is binary (0 or 1).

```python
# Prepare input features and binary target
X_class = df_no_outliers.drop(columns=['Date', 'Time', 'CO(GT)', 'NOx(GT)'])
y_class = df_no_outliers['CO(GT)']

# Train-test split
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
    X_class, y_class, test_size=0.2, random_state=42)

# Normalize features
scaler_class = StandardScaler()
X_train_class_scaled = scaler_class.fit_transform(X_train_class)
X_test_class_scaled = scaler_class.transform(X_test_class)

# Create binary target
threshold = y_train_class.mean()
y_train_class_bin = (y_train_class > threshold).astype(int)
y_test_class_bin = (y_test_class > threshold).astype(int)

# Build MLP classification model
model_class = Sequential([
    Dense(16, activation='relu', input_shape=(X_train_class_scaled.shape[1],)),
    Dropout(0.1),
    Dense(8, activation='relu'),
    Dropout(0.1),
    Dense(1, activation='sigmoid')  # Binary output
])

model_class.compile(optimizer='adam',
                    loss='binary_crossentropy',
                    metrics=['accuracy'])

# Train the model
history_class = model_class.fit(
    X_train_class_scaled, y_train_class_bin,
    epochs=50, batch_size=16, validation_split=0.1, verbose=1
)
```

4.3 Regression Model (Predict NOx)

This regression model predicts the NOx concentration based on other sensor data.

```python
# Prepare input features and continuous target
X_reg = df_no_outliers.drop(columns=['Date', 'Time', 'NOx(GT)'])
y_reg = df_no_outliers['NOx(GT)']

# Train-test split
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(
    X_reg, y_reg, test_size=0.2, random_state=42)

# Normalize features
scaler_reg = StandardScaler()
X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)
X_test_reg_scaled = scaler_reg.transform(X_test_reg)

# Build MLP regression model
model_reg = Sequential([
    Dense(16, activation='relu', input_shape=(X_train_reg_scaled.shape[1],)),
    Dropout(0.1),
    Dense(8, activation='relu'),
    Dropout(0.1),
    Dense(1)  # Linear output
])

model_reg.compile(optimizer='adam',
                  loss='mean_squared_error',
                  metrics=['mae'])

# Train the model
history_reg = model_reg.fit(
    X_train_reg_scaled, y_train_reg,
    epochs=50, batch_size=16, validation_split=0.1, verbose=1
)
```

4.4 Saving and Loading Models for Future Use

After training, each model is saved to disk using the Keras `.h5` format. This ensures the network architecture and weights are stored and can be reloaded later for testing on the generalization dataset.

```python
# Save classification model
model_class.save("classification_model.h5")

# Save regression model
model_reg.save("regression_model.h5")
```

Listing 2: Save Trained Models

To use the models later, load them and re-run predictions:

```python
from tensorflow.keras.models import load_model

# Load models
loaded_class_model = load_model("classification_model.h5")
loaded_reg_model = load_model("regression_model.h5")

# Load and preprocess generalization dataset
gen_df = pd.read_excel("Generalization Dataset.xlsx")
gen_df.replace(-200, np.nan, inplace=True)
gen_df.fillna(method='ffill', inplace=True)

# Preprocess inputs
X_gen_class = gen_df[feature_columns_for_class]
X_gen_class_scaled = scaler_class.transform(X_gen_class)

```

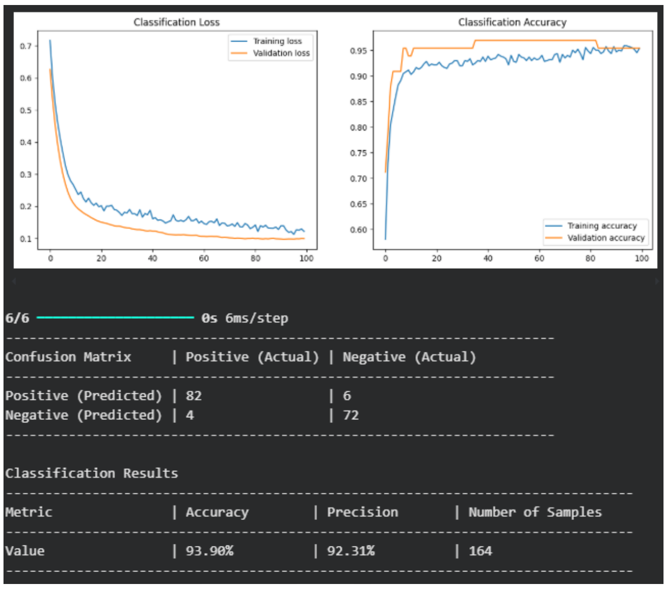
```python
X_gen_reg = gen_df[feature_columns_for_reg]
X_gen_reg_scaled = scaler_reg.transform(X_gen_reg)

# Make predictions
pred_class = (loaded_class_model.predict(X_gen_class_scaled) > 0.5).astype(int)
pred_reg = loaded_reg_model.predict(X_gen_reg_scaled)
```

Listing 3: Load Saved Models for Generalization Dataset

5 Results and Discussion

5.1 Classification Task: CO(GT) Threshold Exceedance



```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
----------------------------------------------------------------------
Confusion Matrix      | Positive (Actual) | Negative (Actual)
----------------------------------------------------------------------
Positive (Predicted) | 82                | 6
Negative (Predicted) | 4                 | 72
----------------------------------------------------------------------

Classification Results
--------------------------------------------------------------------------------
Metric               | Accuracy          | Precision         | Number of Samples
--------------------------------------------------------------------------------
Value                | 93.90%            | 92.31%            | 164
--------------------------------------------------------------------------------
```

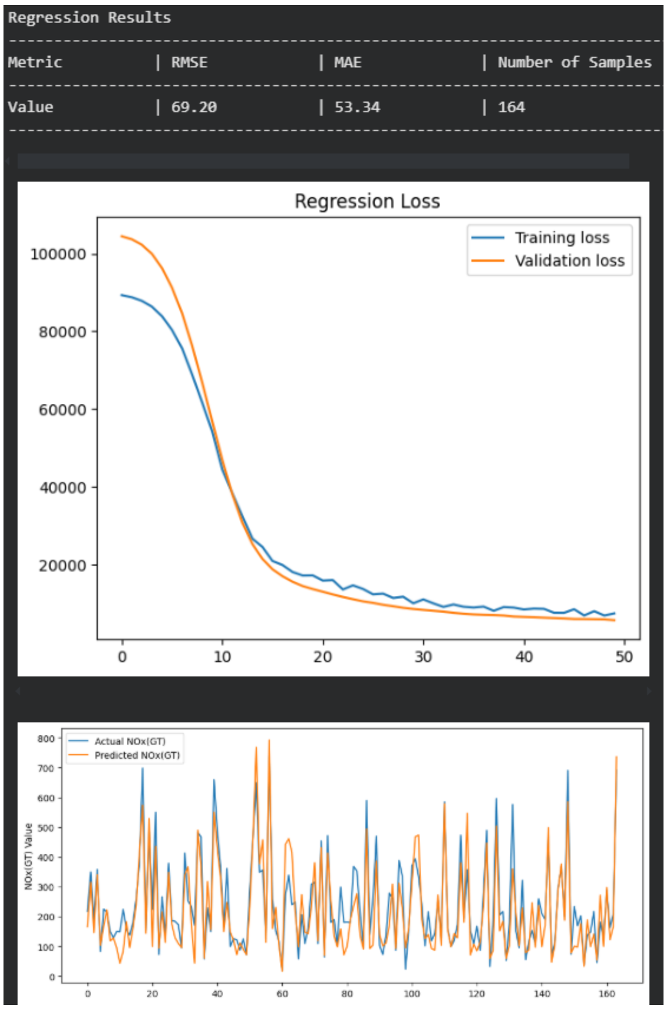
Figure 2: Classification Result

Observations:

- The validation accuracy is consistently high and tracks the training accuracy well, showing the model generalizes effectively.
- Loss steadily decreases without large spikes, indicating stable learning.
- Confusion matrix results show balanced classification with few false predictions:

	Positive (Actual)	**Negative (Actual)**
Positive (Predicted)	82	6
Negative (Predicted)	4	72

5.2 Regression Task: Predicting NOx(GT)



```
Regression Results
------------------------------------------------------------------------
Metric              | RMSE                | MAE                 | Number of Samples
------------------------------------------------------------------------
Value               | 69.20               | 53.34               | 164
------------------------------------------------------------------------
```

Figure 3: Regression Result

Observations:

- Training and validation loss decreased steadily, indicating a well-fitted model with no signs of overfitting.
- The actual vs predicted plot shows a strong overlap in trend, though some high-variance areas are under-predicted or slightly offset.
- RMSE and MAE values are well below the maximum allowed limits (280 and 220), showing excellent predictive accuracy.

5.3 Limitations and Future Work

- **Model Simplicity:** Only shallow MLPs were used. Future work could explore CNNs, RNNs, or attention mechanisms for temporal patterns.
- **Missing Data Handling:** Forward-fill may propagate noise. Interpolation or imputation methods (e.g., KNN) could be more robust. Using Z-scores remove too many samples.
- **Feature Selection:** All variables were used. Feature importance or selection (e.g., via SHAP or PCA) could improve performance.
- **Time Dependency:** The model does not directly consider time-series order. Adding timestamps or lag features may enhance predictions.

The model was trained when I was new to this. This report is purely for learning.

References

UCI Machine Learning Repository. (2008). *Air Quality Data Set.* `https://archive.ics.uci.edu/dataset/360/air+quality`. (Accessed: 2025-05-23)