Autoregressive Models: Unraveling Sequential Patterns

IIT Ropar Minor in AI

3rd March, 2025

1 A Tale of Predictive Insight

Emma stood by the window of her small meteorological research station, studying the intricate weather charts that covered her walls. For years, she had been fascinated by a simple yet profound question: Could the past truly predict the future?

Her breakthrough came on a crisp autumn morning. While most meteorologists relied on complex, disconnected measurements, Emma noticed something different. The temperature wasn't random—it told a story. Each day whispered secrets about the next.

> "It's not about individual data points," she would tell her colleagues,
> "it's about the conversation between yesterday and tomorrow."

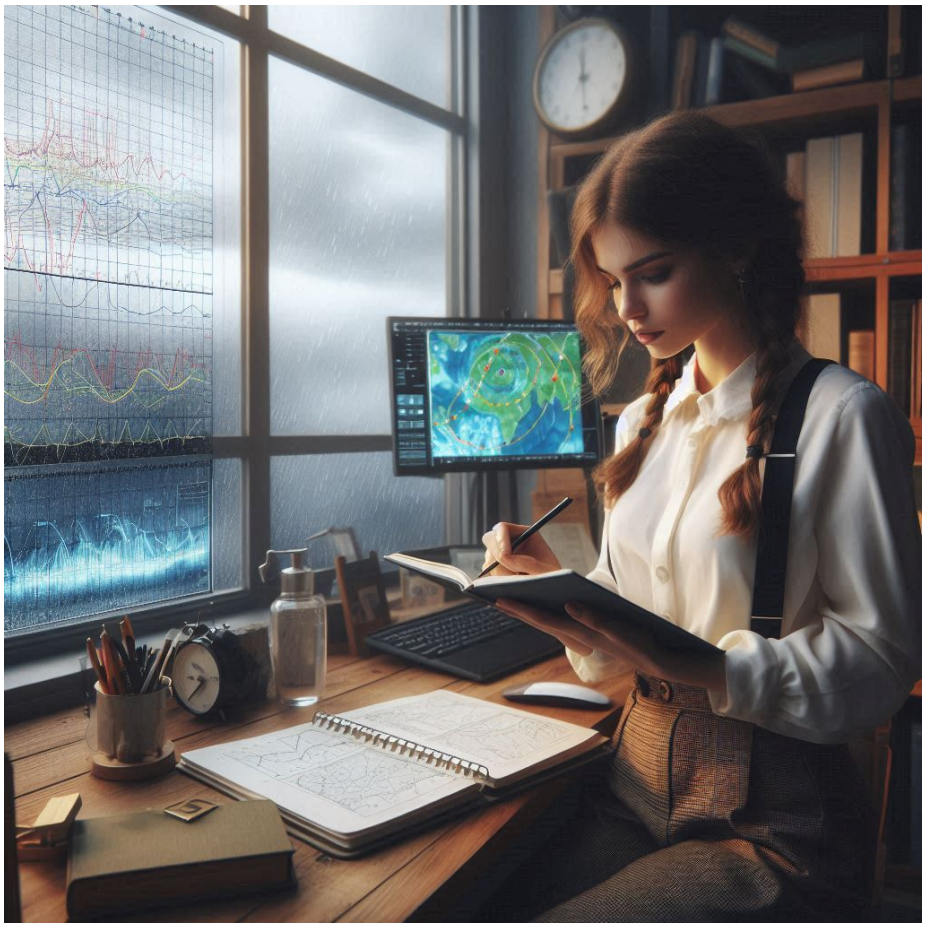

Figure 1: Emma analyzing weather patterns

2 Understanding Autoregressive Models

2.1 Sequential Data Fundamentals

- Collections of ordered observations: $D = \{t_1, t_2, ..., t_n\}$
- Applications: Weather prediction, medical records analysis, NLP
- Key challenge: Handle varying-length inputs (e.g. text, medical time series)

2.2 Comparing Sequence Modeling Approaches

Method	Input Type	Typical Use
AR Models	Fixed-length sequences	Time series forecasting
MLP	Fixed-length vectors	Tabular data
CNN	Varying resolution grids	Image processing

2.3 The Temperature Prediction Case

From whiteboard data (past 2 days window):

Day	Temp (°C)
1	30
2	32
3	33
4	31
5	?

Prediction equation for day 5:

$$t_5 = b + w_1 t_4 + w_2 t_3 \tag{1}$$

3 Mathematical Foundation

3.1 General AR(p) Model

For order p:

$$X_t = b + \sum_{i=1}^{p} w_i X_{t-i} + \epsilon_t \tag{2}$$

Special cases:

- AR(1): $X_t = b + w_1 X_{t-1}$
- AR(2): $X_t = b + w_1 X_{t-1} + w_2 X_{t-2}$

3.2 State Transition Dynamics

Autoregressive models leverage a state transition matrix to represent their sequential relationships efficiently. This matrix captures how previous observations influence the current state.

- **Matrix Representation:** AR models can be expressed in matrix form to reveal temporal dependencies. For example:

$$\begin{bmatrix} X_t \\ X_{t-1} \\ \vdots \\ X_{t-p+1} \end{bmatrix} = \begin{bmatrix} w_1 & w_2 & \cdots & w_p \\ 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & 0 \end{bmatrix} \begin{bmatrix} X_{t-1} \\ X_{t-2} \\ \vdots \\ X_{t-p} \end{bmatrix} + \begin{bmatrix} b \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

- **Explanation:**
  - Each column in the matrix represents a lagged variable's coefficient.
  - The lower triangular structure ensures that earlier observations propagate through the system in an orderly fashion.

4 Practical Implementation

4.1 Generating Synthetic Temperature Data

Emma developed a Python script to simulate and analyze temperature patterns:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Set random seed for reproducibility
np.random.seed(182)

# Generate synthetic temperature data
days = np.arange(1, 101)
temperature = 25 + 0.5 * np.sin(0.1*days) + np.random.normal
    (0, 0.5, size=100)

# Visualize the temperature data
plt.figure(figsize=(10, 5))
plt.plot(days, temperature)
plt.title('Synthetic Temperature Time Series')
plt.xlabel('Days')
plt.ylabel('Temperature')
plt.show()
```

4.2 Modified Temperature Prediction Code

Using 2-day window (p=2) from whiteboard example:

```
# Set the autoregressive lag
p = 2  # looking back 2 days

# Create feature matrix and target variable
X = []
y = []

for i in range(len(temperature) - p):
    X.append(temperature[i:i+p])
    y.append(temperature[i+p])
```

```
11
12 X = np.array(X)
13 y = np.array(y)
14
15 # Split the data into training and testing sets
16 X_train, X_test, y_train, y_test = train_test_split(X, y,
       test_size=0.2, random_state=42)
17
18 # Add bias term to the feature matrix
19 X_train_with_bias = np.c_[np.ones(X_train.shape[0]), X_train
       ]
20 X_test_with_bias = np.c_[np.ones(X_test.shape[0]), X_test]
21
22 # Estimate coefficients using least squares
23 coeff = np.linalg.lstsq(X_train_with_bias, y_train, rcond=
       None)[0]
24 print("Model Coefficients:", coeff)
25
26 # Make predictions
27 y_pred = X_test_with_bias @ coeff
28
29 # Evaluate model performance
30 mse = mean_squared_error(y_test, y_pred)
31 print("Mean Squared Error:", mse)
```

5 Extended Applications

5.1 Text Sequence Processing

Tokenization in Natural Language Processing (NLP) is the process of breaking down text into smaller units, or tokens. These tokens can be words, phrases, sentences, or characters. Tokenization is a key step in NLP because it helps to structure text data for machine processing.

- **Purpose:** Tokenization converts raw text into meaningful numerical sequences for machine understanding.
- **Preprocessing:** Involves cleaning text (e.g., removing punctuation, converting to lowercase).
- **Example:**

```
1 "I love my country"      [0, 1, 2, 3]
2 "Country"      ID 42 (Oxford English Dictionary-based,
     for example)
```

5.2 Vocabulary Building

Constructing a vocabulary is critical to assigning unique identifiers to each token:

- **Steps:**
  - Collect all unique words from the training corpus.
  - Assign each word a unique integer identifier.
  - Include special tokens like `<UNK>` (unknown), `<PAD>` (padding).
- **Output:**
  - A mapping of each word to its unique ID.
  - Example vocabulary:

```
Vocabulary = {
    "I": 0, "love": 1, "my": 2, "country":
        3,
    "<UNK>": 4, "<PAD>": 5
}
```

- **Applications:**
  - **Efficient Storage:** Maps text data to numerical representations for fast computation.
  - **Facilitates NLP Tasks:** Enables algorithms to process text for tasks like classification, prediction, and generation.
  - **Special Tokens Utility:**
    - `<UNK>` handles out-of-vocabulary words during test time.
    - `<PAD>` helps create uniform sequence lengths for batch processing.

6 Conclusion

Autoregressive models bridge diverse domains - from weather patterns to medical diagnostics. By understanding temporal dependencies through parameters like p, we unlock predictive power in sequential data while addressing challenges like varying input lengths through techniques like padding and truncation.