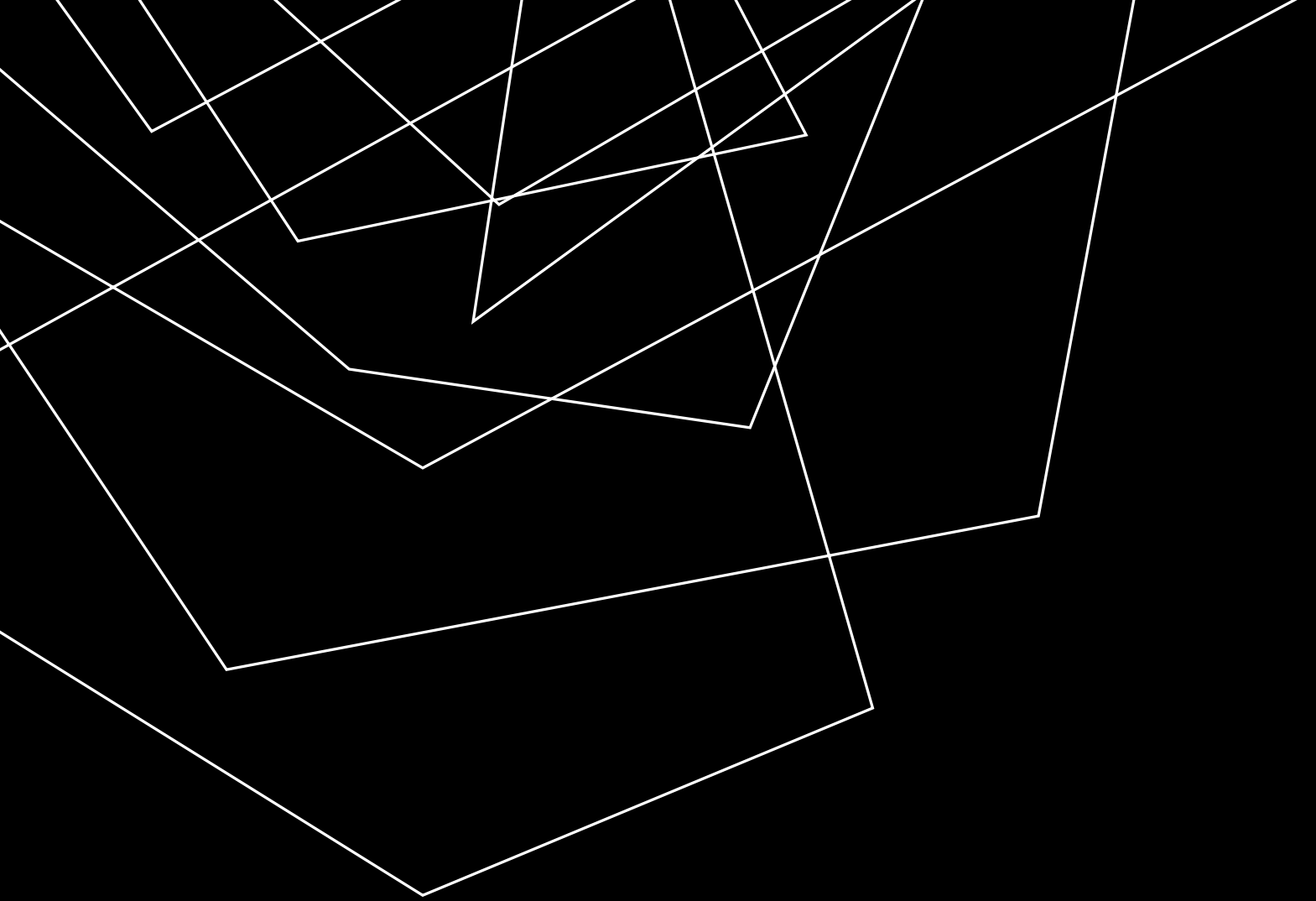


PARALLEL LSTM

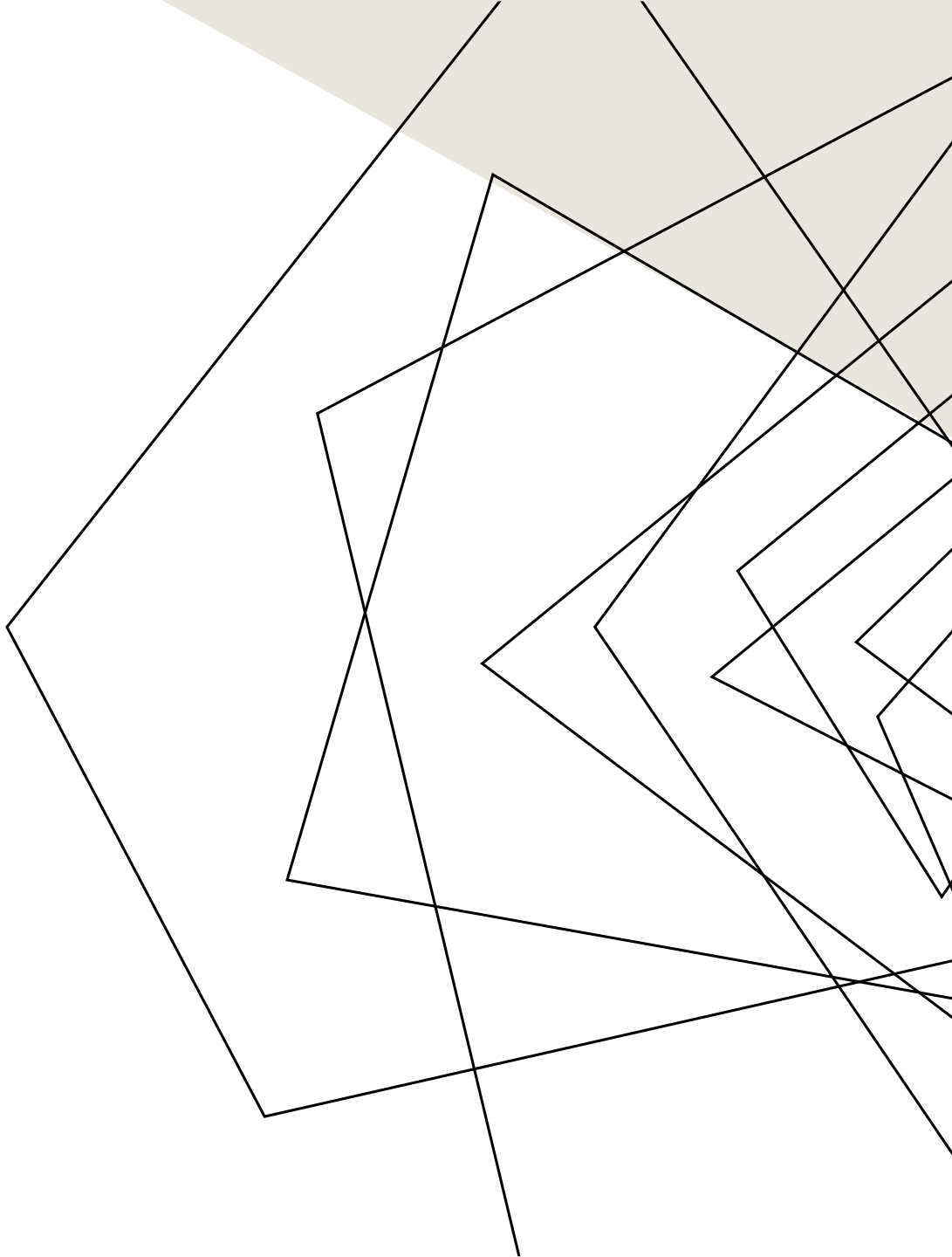
ABOUT US

Vietnam National University - University of Science

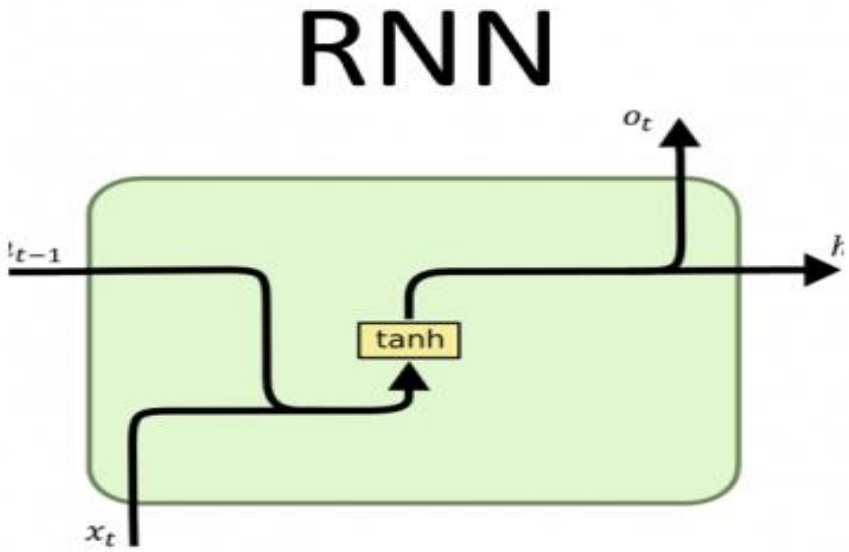
CSC14116 - Applied Parallel Programming

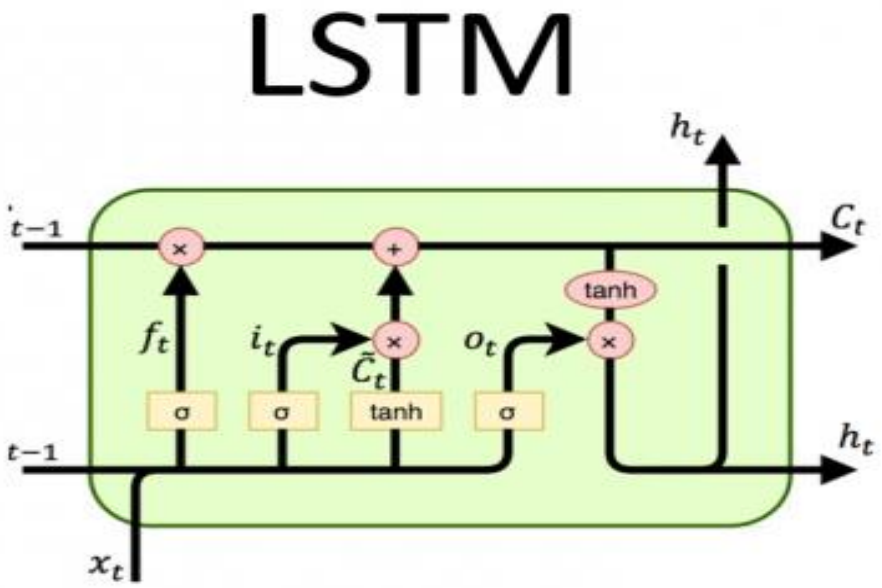
20120105 - Le Hoang Huy

20120120 - Nguyen Viet Khoa



INTRODUCTION



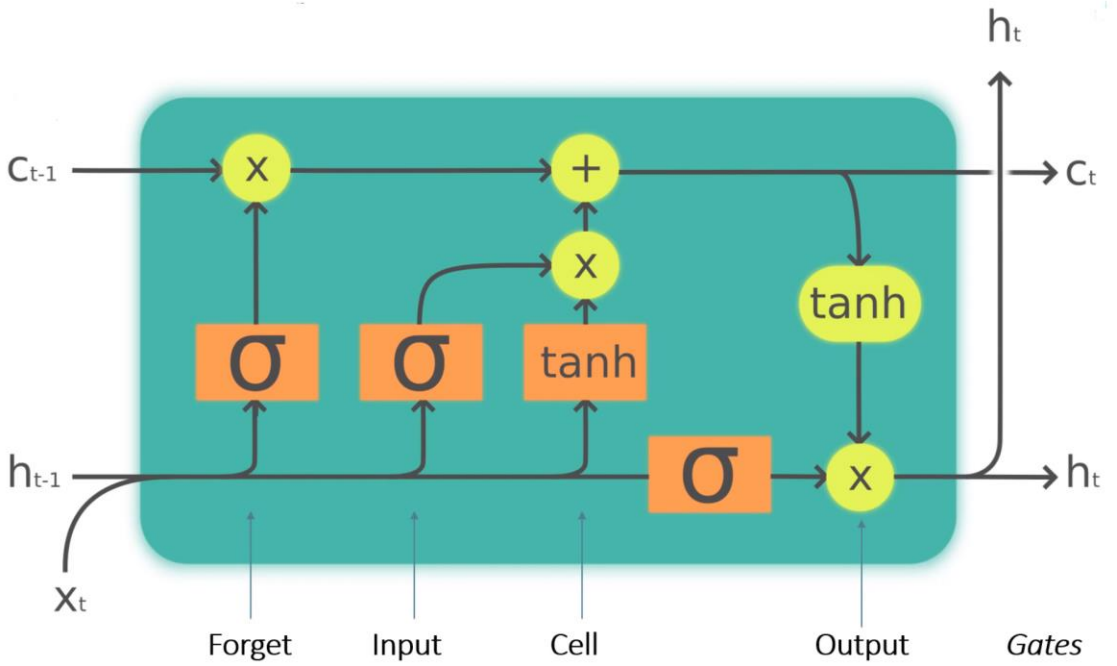


ABOUT RNN AND LSTM

Mạng nơ-ron hồi quy (RNN) huấn luyện trên đầu vào chứa các chuỗi dữ liệu, trong đó nó học về mối quan hệ phụ thuộc vào thời gian giữa các phần khác nhau của đầu vào. Ví dụ, nếu chúng ta đưa vào một chuỗi các từ làm đầu vào, tức là một câu, một RNN có thể học về mối quan hệ giữa các từ khác nhau và từ đó học các quy tắc ngữ pháp như mối quan hệ giữa động từ và trạng từ, v.v.

LSTM là viết tắt của Long Short-Term Memory (bộ nhớ dài ngắn hạn). Đó là một loại mạng nơ-ron hồi quy (RNN) được thiết kế để giải quyết vấn đề của độ gradient bị biến mất trong các RNN truyền thống. Các mạng LSTM có khả năng lựa chọn để ghi nhớ hoặc quên thông tin theo thời gian dài, làm cho chúng đặc biệt hiệu quả cho các nhiệm vụ liên quan đến dữ liệu tuần tự, chẳng hạn như nhận dạng tiếng nói, dịch ngôn ngữ và dự đoán chuỗi thời gian.

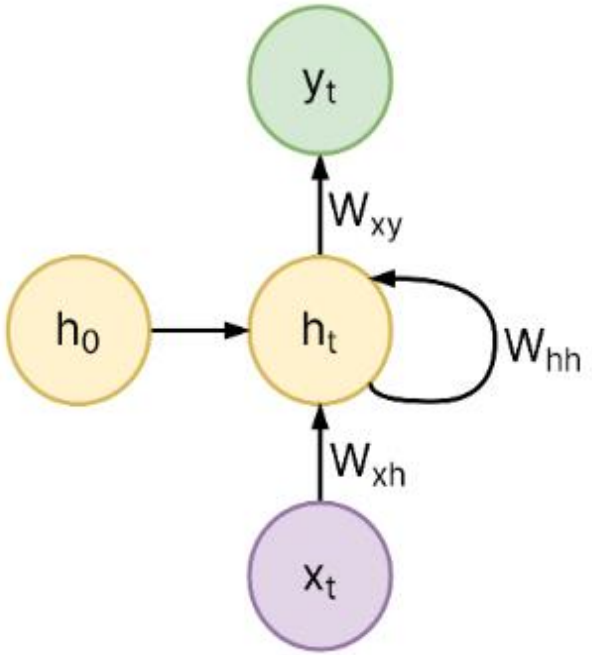
LSTM ARCHITECTURE



Một mạng nơ-ron LSTM (Long Short-Term Memory) cơ bản bao gồm các thành phần sau:

1. **Lớp đầu vào (x_i)**:
   - Mỗi điểm dữ liệu x_i tại bước thời gian t chứa n_x đặc trưng.
   - Lớp đầu vào xử lý các đặc trưng này và gửi chúng đến LSTM.
2. **Lớp ẩn (h_t)**:
   - Lớp ẩn của LSTM có n_h nút.
   - Tại mỗi bước thời gian t, lớp ẩn nhận đầu vào từ lớp đầu vào (x_t) cũng như từ trạng thái ẩn trước đó (h_{t-1}).
3. **Lớp đầu ra (y_t)**:
   - Lớp đầu ra của LSTM sản sinh đầu ra y_t tại mỗi bước thời gian, có thể được sử dụng cho dự đoán hoặc xử lý tiếp theo.
   - Số nút trong lớp đầu ra là n_y.

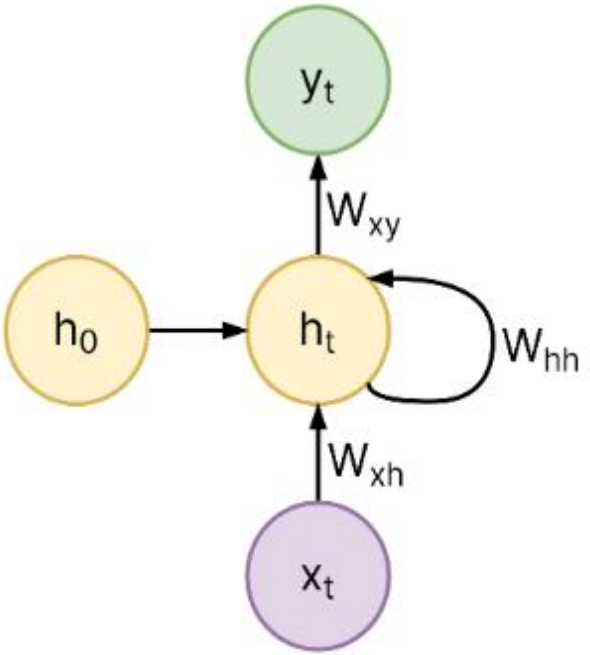
LSTM ARCHITECTURE



4. **Ma trận trọng số**:

- W_{xh} (Từ Đầu vào đến Lớp Ẩn):
  - Ma trận trọng số này có kích thước $n_x \times n_h$ và biểu diễn các tham số kết nối từ lớp đầu vào đến lớp ẩn.
  - Mỗi phần tử $W_{xh}[i, j]$ xác định mức độ kết nối giữa đặc trưng đầu vào i và nút ẩn j.
- W_{hh} (Các Kết Nối Lặp Lại trong Lớp Ẩn):
  - Ma trận trọng số này có kích thước $n_h \times n_h$ và biểu diễn các kết nối lặp lại trong lớp ẩn.
  - Mỗi phần tử $W_{hh}[i, j]$ xác định mức độ kết nối từ nút ẩn i tại bước thời gian t đến nút ẩn j tại bước thời gian $t + 1$.
- W_{hy} (Từ Lớp Ẩn đến Lớp Đầu ra):
  - Ma trận trọng số này có kích thước $n_h \times n_y$ và biểu diễn các kết nối từ lớp ẩn đến lớp đầu ra.
  - Mỗi phần tử $W_{hy}[i, j]$ xác định ảnh hưởng của nút ẩn i lên nút đầu ra j tại bước thời gian t.

LSTM ARCHITECTURE



5. **Xử lý tại Mỗi Bước Thời Gian**:

- Tại mỗi bước thời gian t, LSTM tính toán:
  - Trạng thái ẩn h_t bằng cách sử dụng đầu vào x_t và trạng thái ẩn trước đó h_{t-1}:

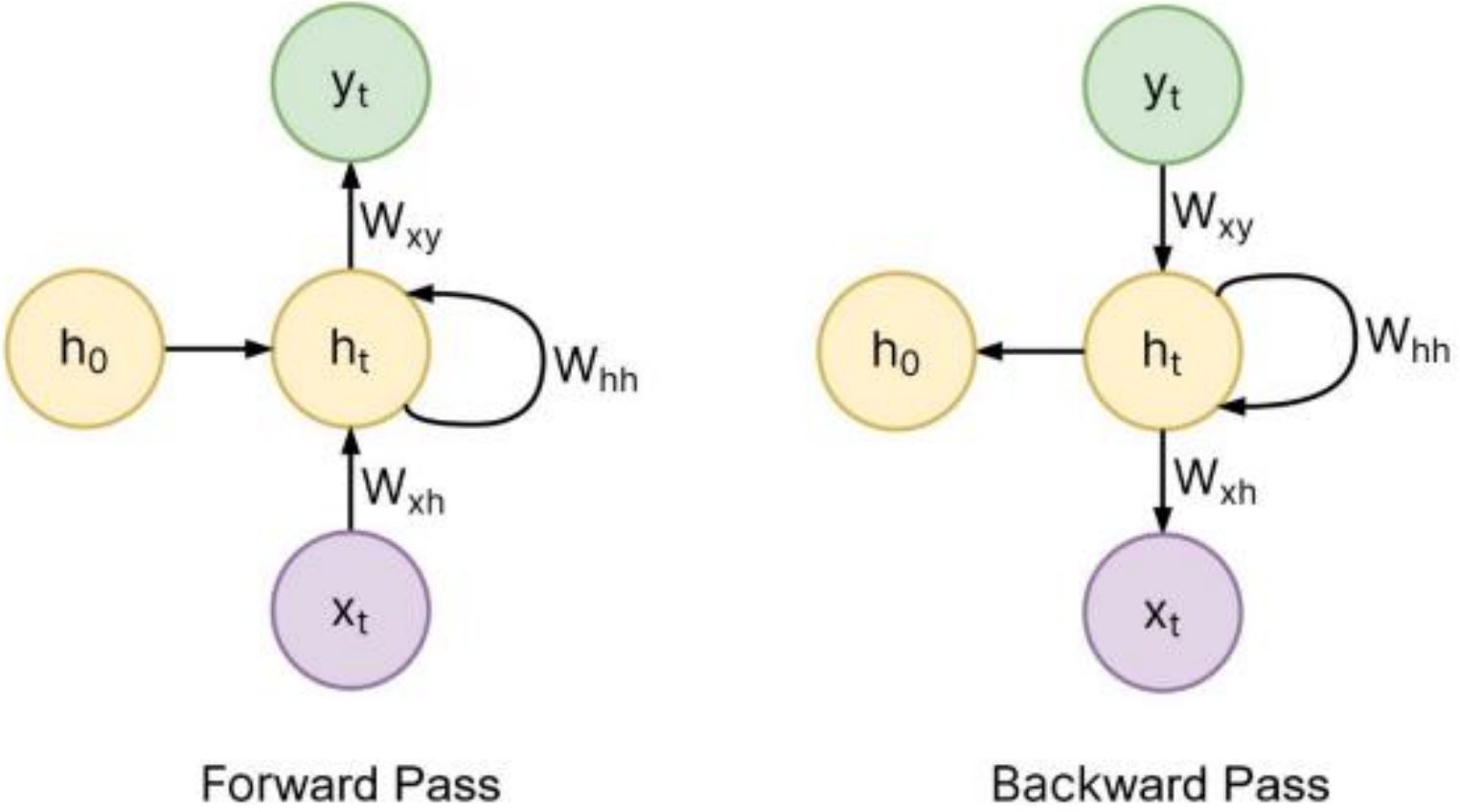
$$h_t = \sigma(W_{xh}^T x_t + W_{hh}^T h_{t-1} + b_h)$$

  - Đây, σ đại diện cho hàm kích hoạt (thường là hàm sigmoid hoặc tanh) được áp dụng từng phần tử, và b_h là thuật ngữ điều chỉnh cho lớp ẩn.
  - Đầu ra y_t sau đó được tính dựa trên h_t:

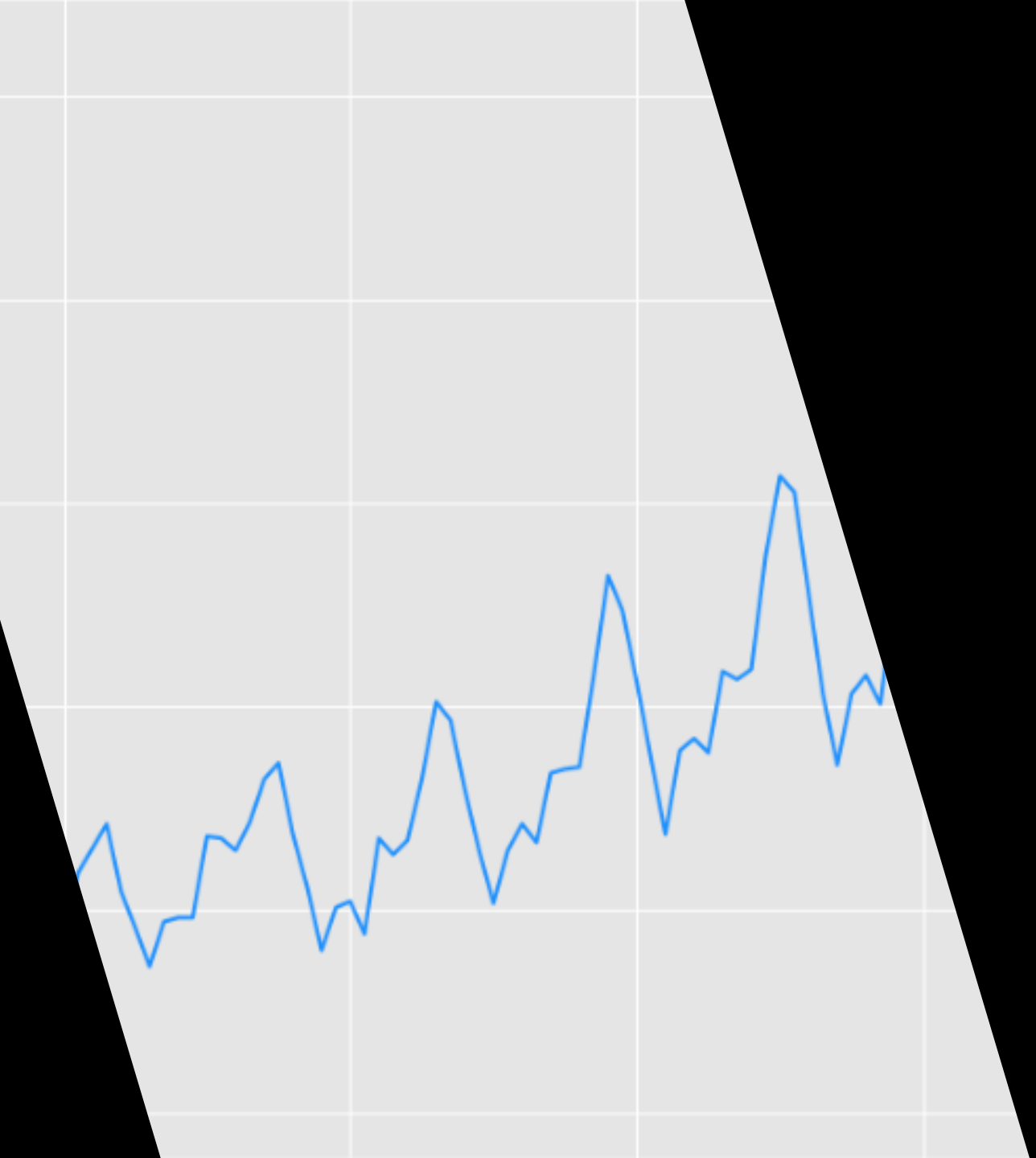
$$y_t = \mathbf{softmax}(W_{hy}^T h_t + b_y)$$

  - **softmax** được sử dụng để chuyển đổi đầu ra thô thành xác suất, và b_y là thuật ngữ điều chỉnh cho lớp đầu ra.

LSTM ARCHITECTURE

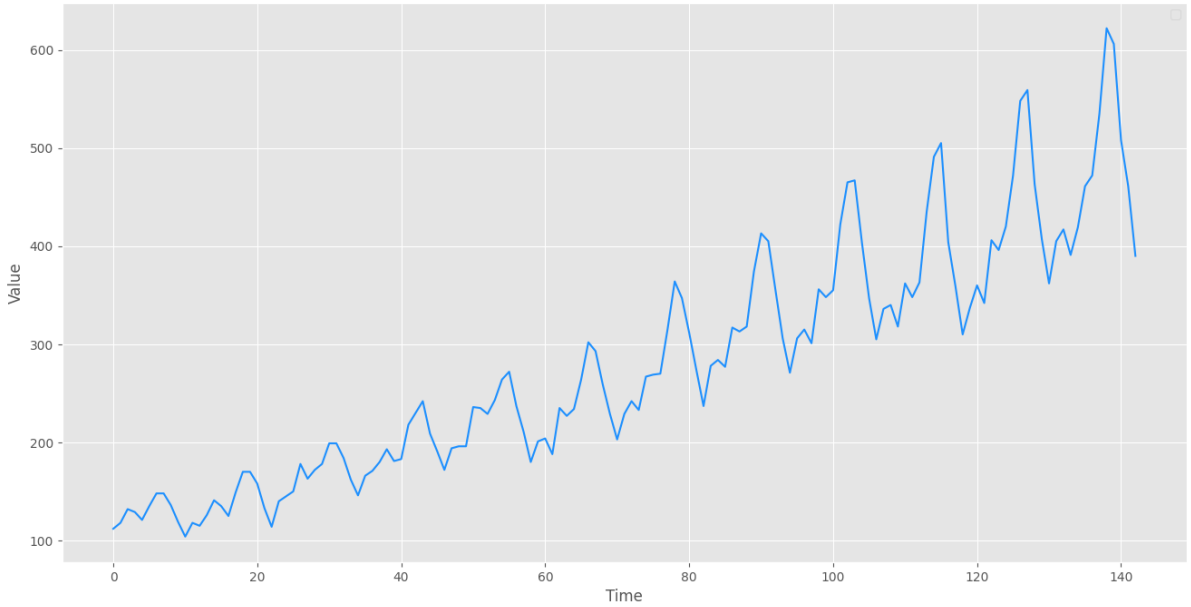


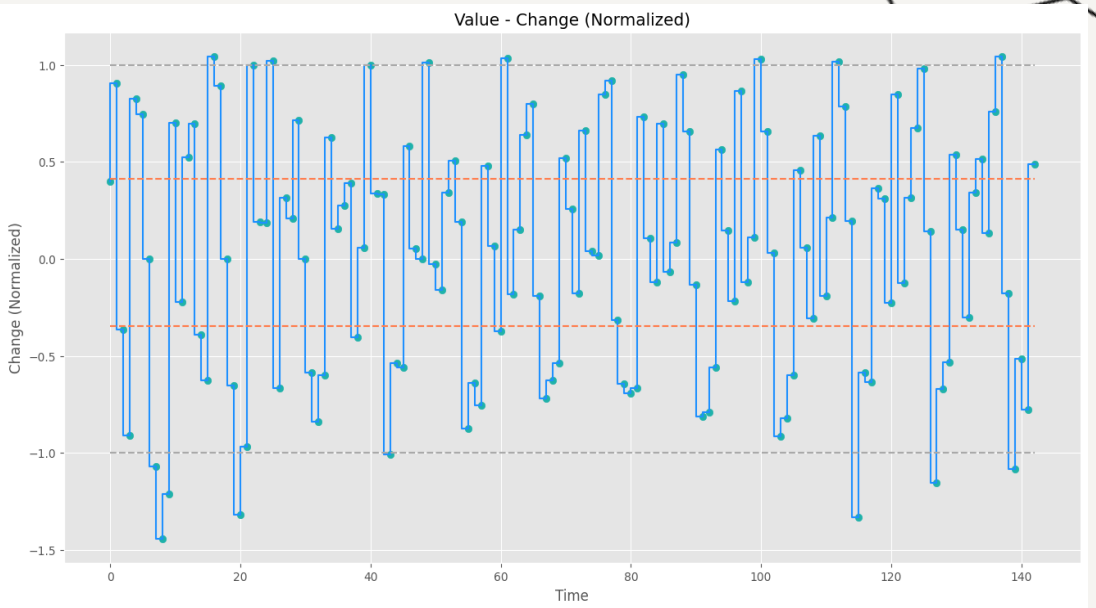
LSTM Forward & Backward Propagation

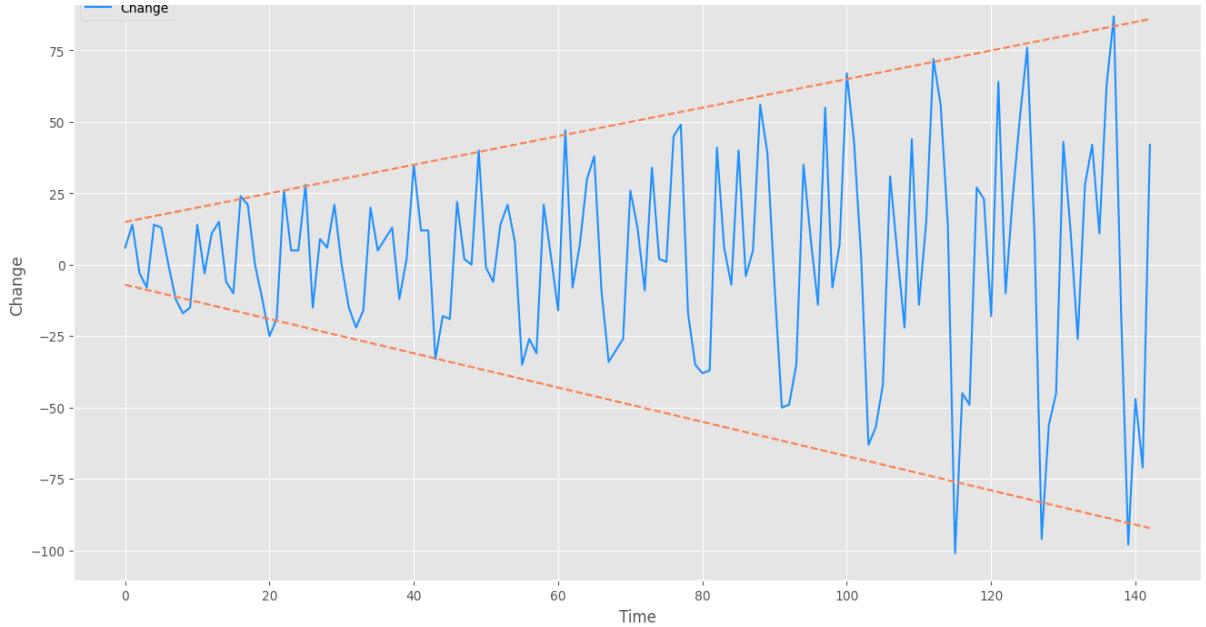


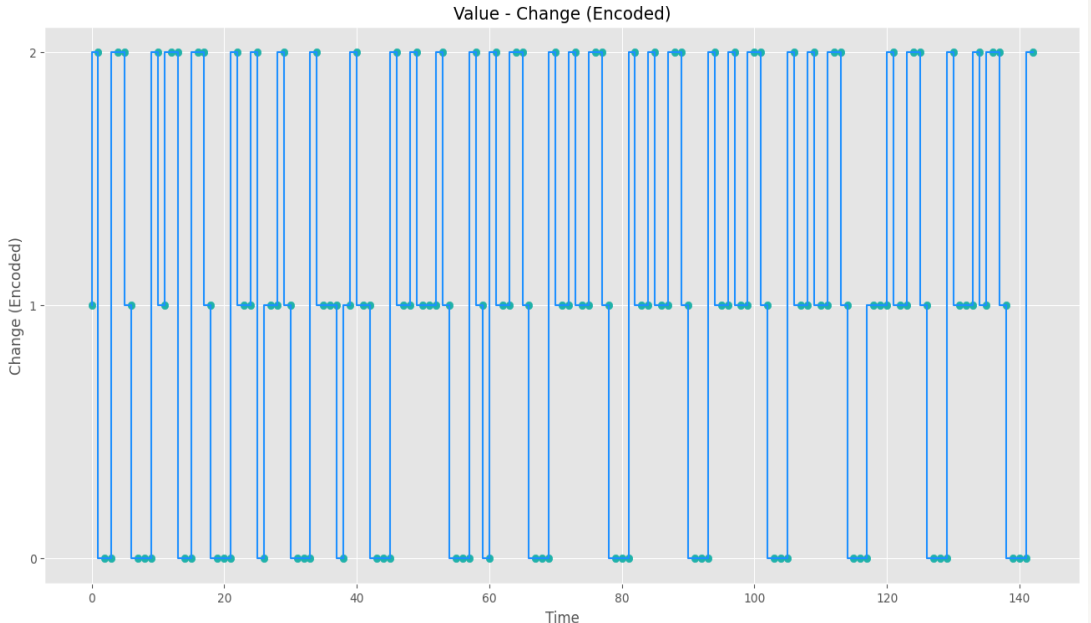
DATASET

US Airline Passengers









DEFINE FUNCTIONS

ACTIVATION FUNCTIONS

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}}$$

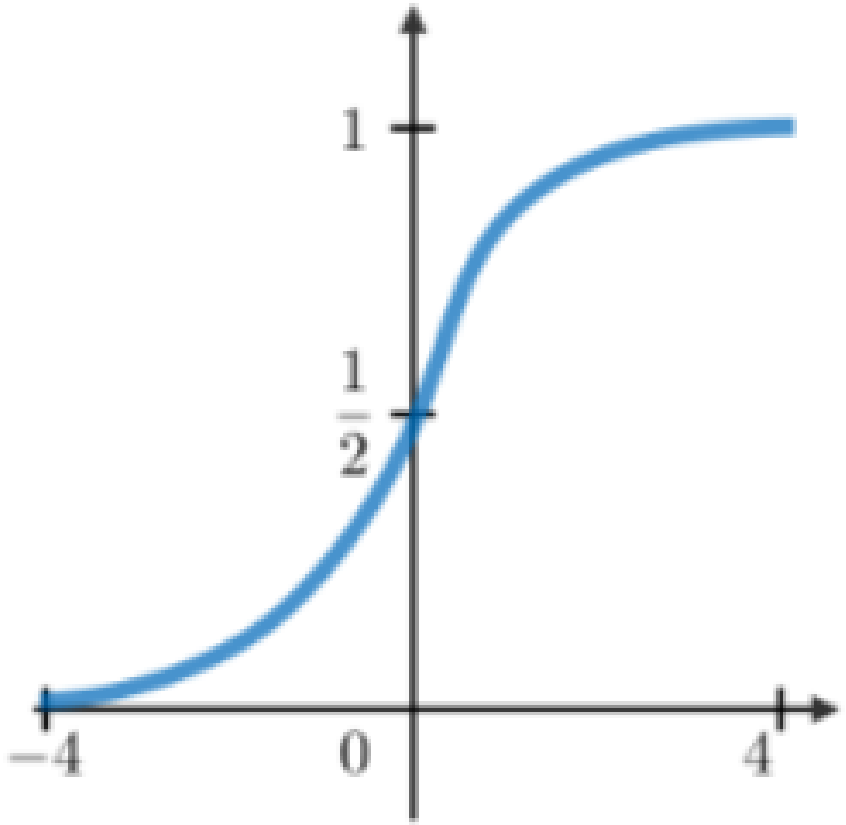
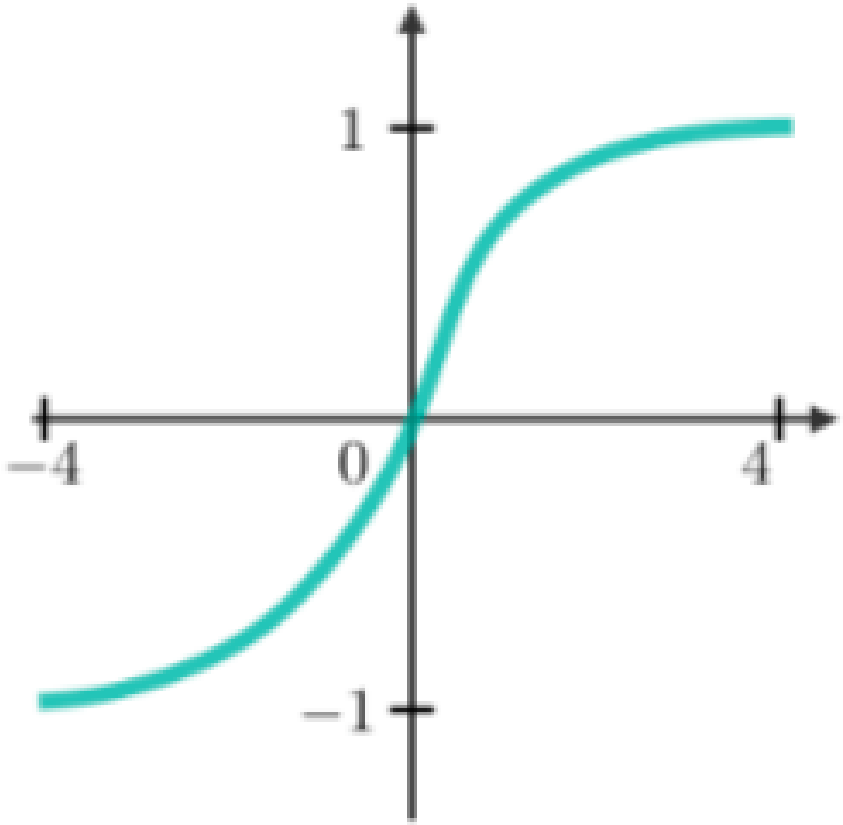
$$\text{sigmoid}'(x) = \text{sigmoid}(x) \cdot (1 - \text{sigmoid}(x))$$

hàm của hàm sigmoid tại `x`, ngược lại trả về giá trị của hàm s

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\tanh'(x) = 1 - \tanh^{2}(x)$$

n tanh tại `x`, ngược lại trả về giá trị của hàm tanh

Sigmoid	Tanh
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$
	

Hàm Softmax (`softmax`):

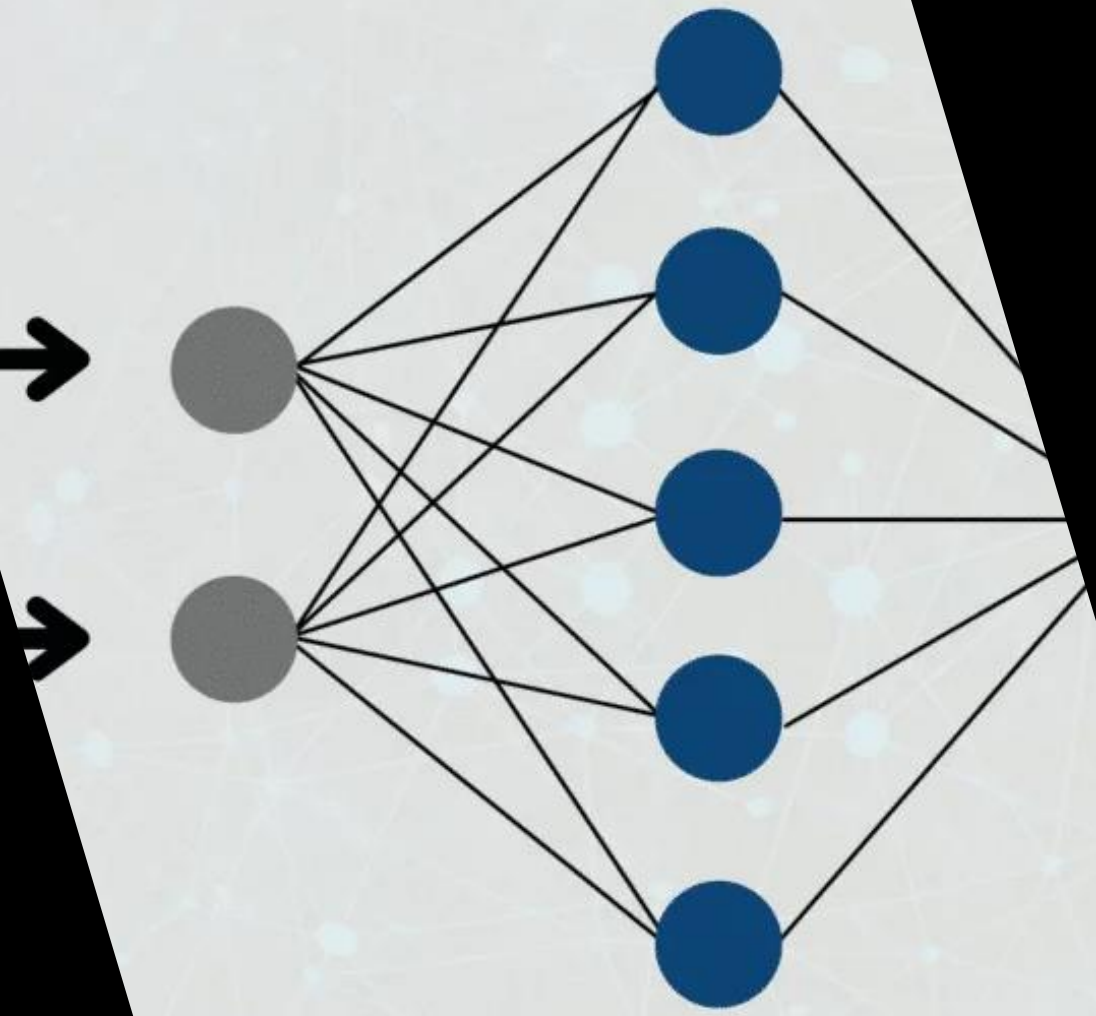
- **Công thức**:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$

với (N) là số lượng phần tử trong vector (x).

- **Đạo hàm**: Đạo hàm của hàm softmax có thể được tính trong quá trình tính toán gradient của hàm mất mát.
- **Tham số**:
  - `x` : Vector đầu vào của hàm softmax.
  - `derivative` : Nếu là True, không trả về giá trị. (Đạo hàm của hàm softmax được tính trong quá trình tính toán gradient của hàm mất mát).

```python
# Activation Function: Softmax
def softmax(x, derivative=False):
    if derivative:
        pass
    else:
        return np.exp(x + 1e-12) / np.sum(np.exp(x + 1e-12))
```



DEFINE FUNCTIONS

NEURAL NETWORK FUNCTIONS

Hàm init_orthogonal được viết để khởi tạo ma trận tham số của mạng nơ-ron theo phương pháp Orthogonal Initialization.

```python
# Function: Initialize the neural network
def init_orthogonal(param):
    if param.ndim < 2:
        raise ValueError("Only parameters with 2 or more dimensions are supported.")
    rows, cols = param.shape
    new_param = randn(rows, cols)
    if rows < cols:
        new_param = new_param.T
    q, r = np.linalg.qr(new_param)
    d = np.diag(r, 0)
    ph = np.sign(d)
    q *= ph
    if rows < cols:
        q = q.T
    new_param = q
    return new_param
```

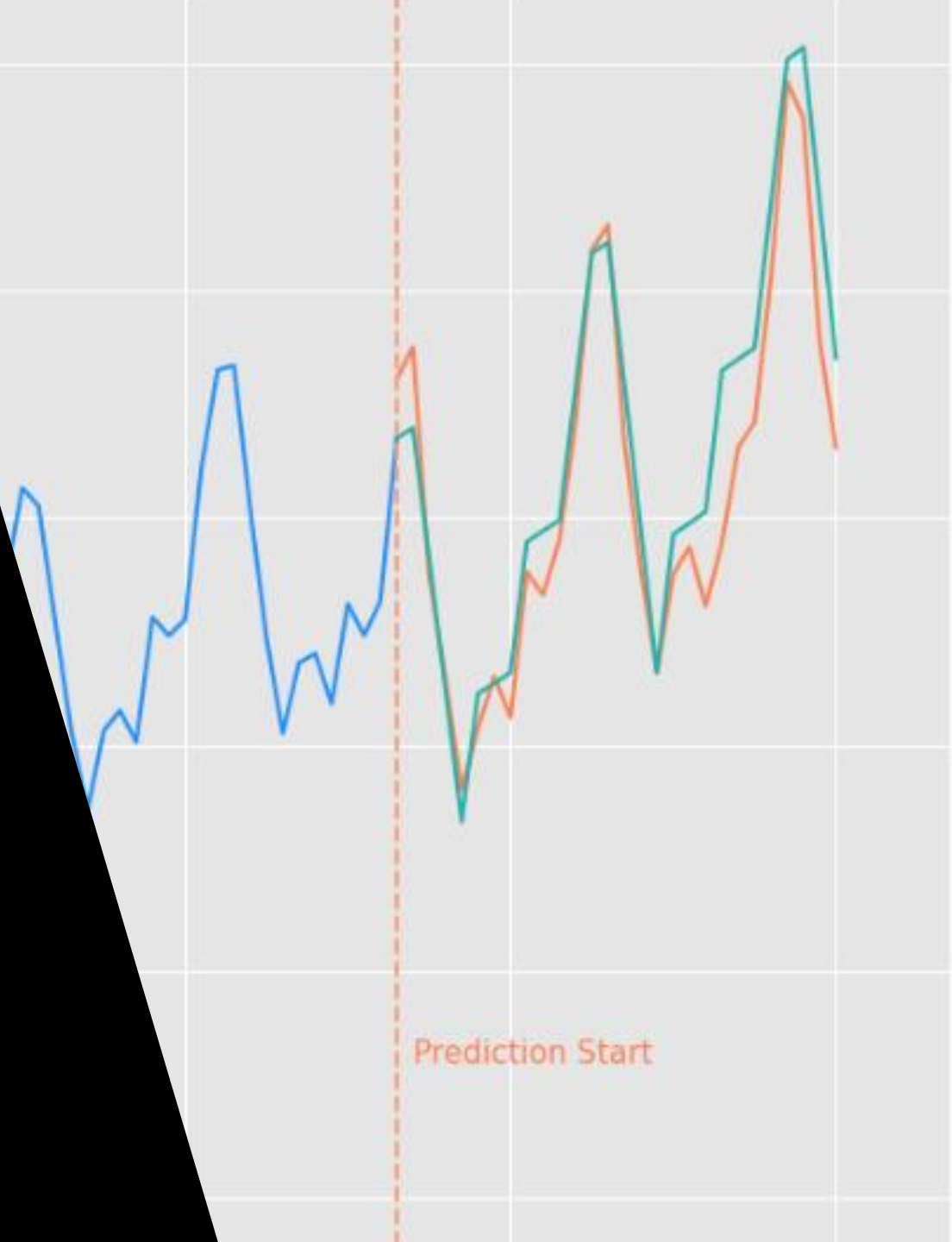
- Phương pháp Orthogonal Initialization nhằm đảm bảo rằng các ma trận tham số của mạng nơ-ron được khởi tạo sao cho không gian biểu diễn của các đặc trưng là độc lập và dễ dàng cho quá trình học của mạng nơ-ron.
- Việc sử dụng hàm này để khởi tạo các ma trận tham số của mạng nơ-ron giúp cải thiện hiệu suất và độ tin cậy của mô hình trong quá trình huấn luyện.

Hàm init_lstm được viết để khởi tạo các ma trận tham số của mạng LSTM (Long Short-Term Memory).

```python
# Function: Initialize the LSTM network
def init_lstm(hidden_size, vocab_size):
    U = np.zeros((hidden_size, vocab_size))
    V = np.zeros((hidden_size, hidden_size))
    W = np.zeros((vocab_size, hidden_size))
    b_hidden = np.zeros((hidden_size, 1))
    b_out = np.zeros((vocab_size, 1))
    U = init_orthogonal(U)
    V = init_orthogonal(V)
    W = init_orthogonal(W)
    return U, V, W, b_hidden, b_out
```

DEFINE FUNCTIONS

INFERENCE FUNCTIONS



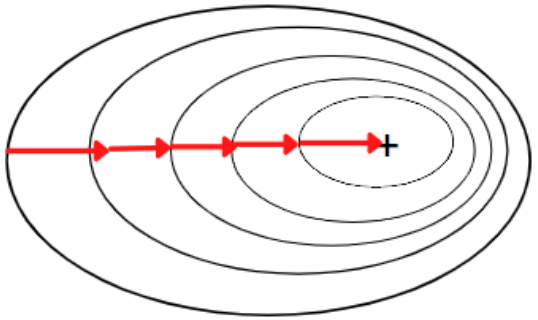
Hàm predict được viết để dự đoán các giá trị trong tập dữ liệu chính (main demo dataset) bằng cách sử dụng một mô hình LSTM đã được huấn luyện trước.

```python
# Function: Inference on the main demo dataset
def predict(df, params, raw_data, centroids, devide_lines):
    prediction_starting_point = int(prediction_start * len(raw_data))
    input_sequence = ' '.join([e for e in raw_data[prediction_starting_point : prediction_starting_point + window_size]])
    num_tokens_to_predict = len(raw_data) - prediction_starting_point
    prediction = inference_next_sequence(params = params, sentence = input_sequence, num_tokens = num_tokens_to_predict)[window_size:]

    prediction_data = raw_data[:prediction_starting_point] + prediction
    prediction_data = [int(e) for e in prediction_data]
    df['Prediction_Encoded'] = prediction_data

    def prediction_decode(row):
        centroid = centroids[-1]
        for i, line in enumerate(sorted(devide_lines)):
            if row['Prediction_Encoded'] == i:
                centroid = centroids[i]
        if row['Change_Normalized'] >= 0:
            return centroid * row['Upper_Bound']
        else:
            return - centroid * row['Lower_Bound']

    df['Change_Prediction'] = df.apply(prediction_decode, axis=1)

    raw = df['Value'].to_list()
    pred = df['Change_Prediction'].to_list()[prediction_starting_point:]
    prediction = raw[:prediction_starting_point]
    for pred in pred:
        prediction.append( prediction[-1] + pred)
    df['Prediction'] =  prediction
    return df
```

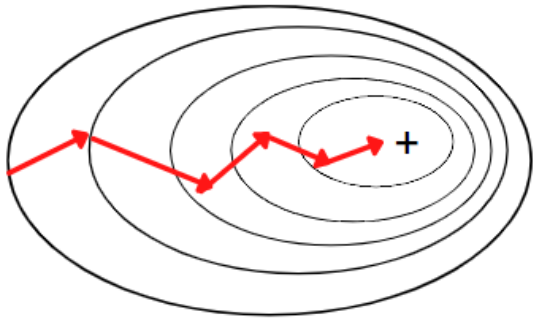
OPTIMIZATION

FOR FUTHER PARALLEL STRATEGY

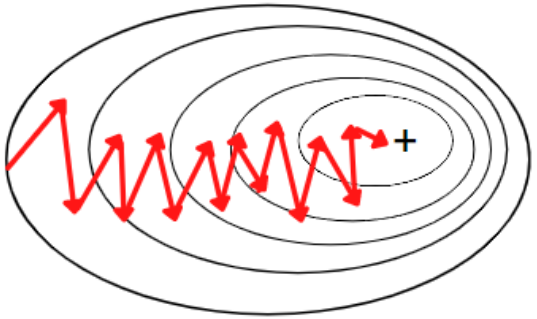
Batch Gradient Descent



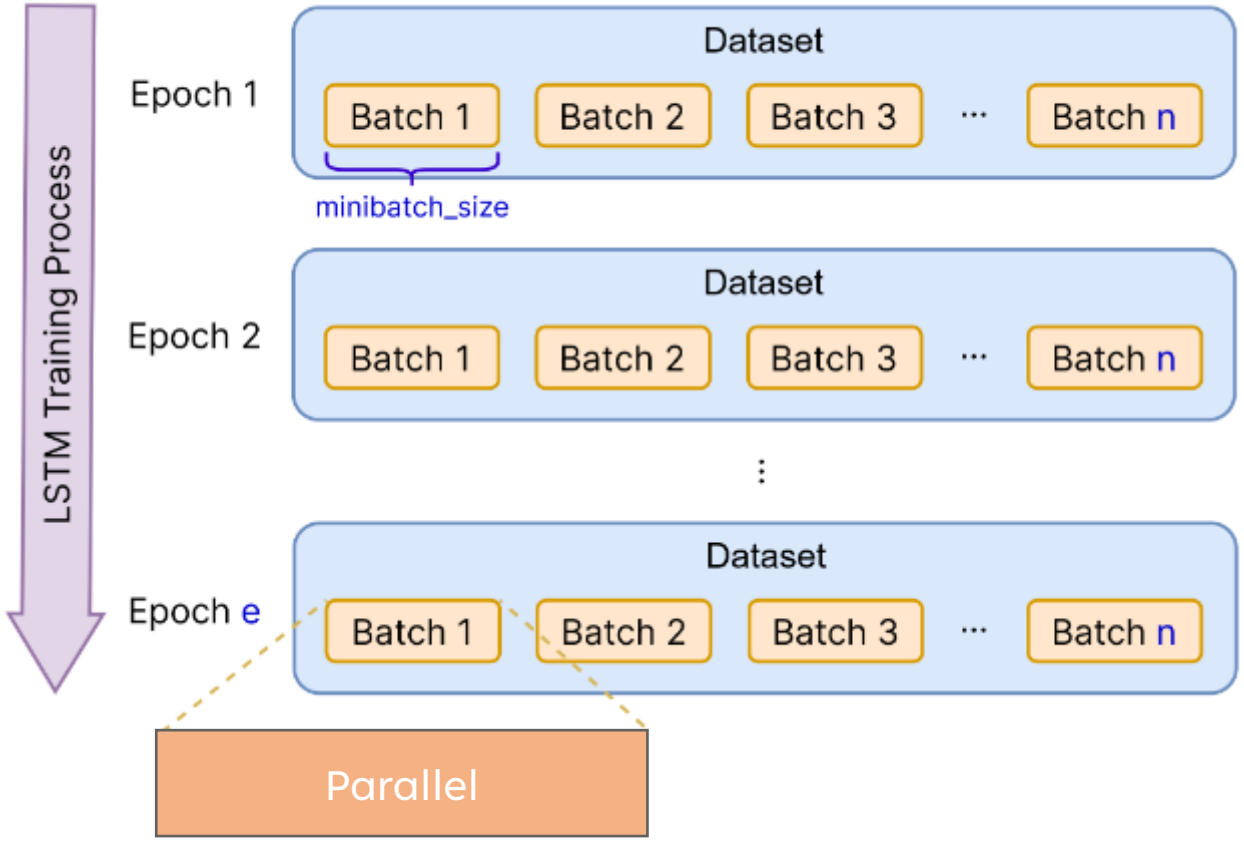
Mini-Batch Gradient Descent



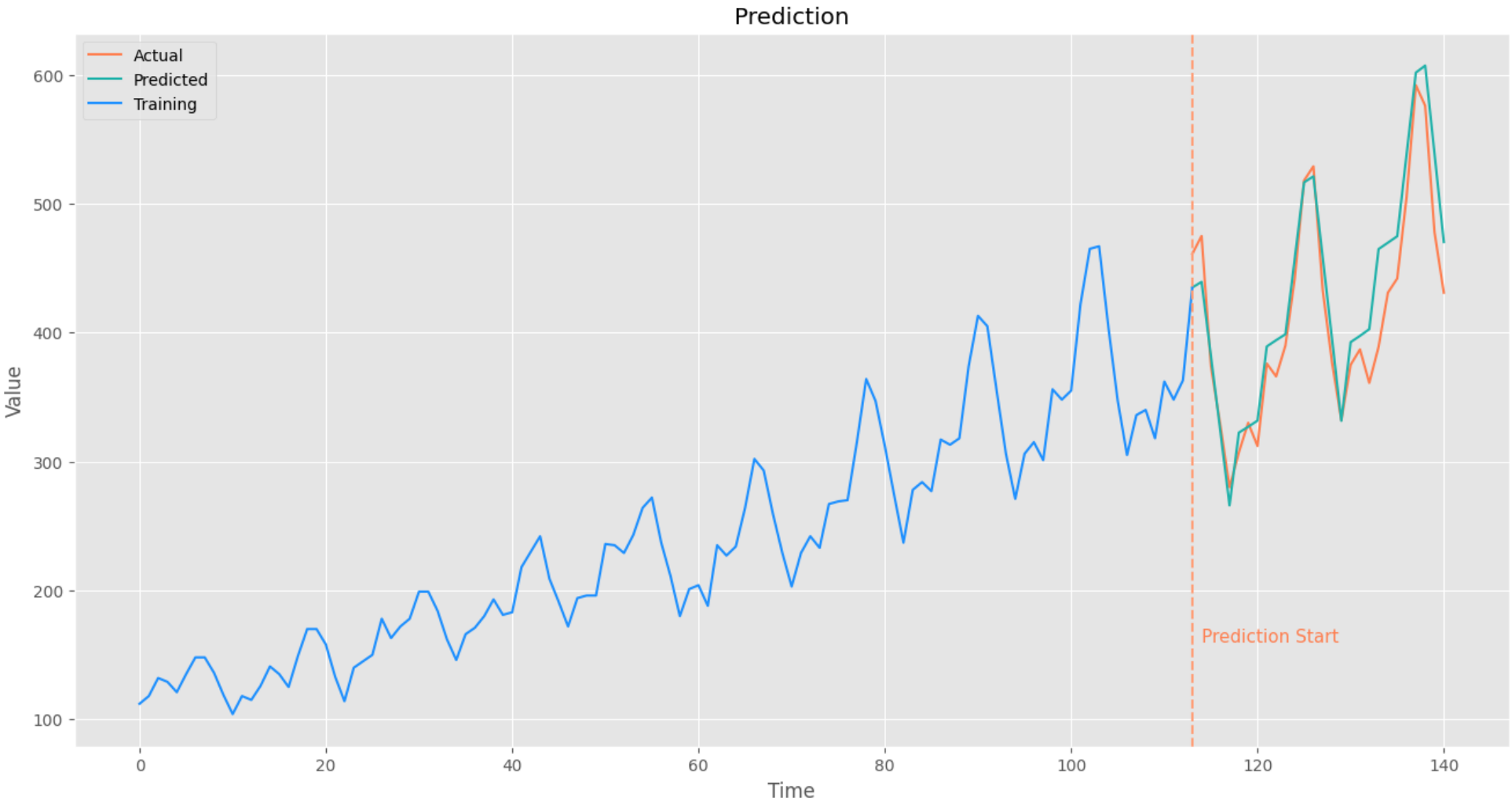
Stochastic Gradient Descent

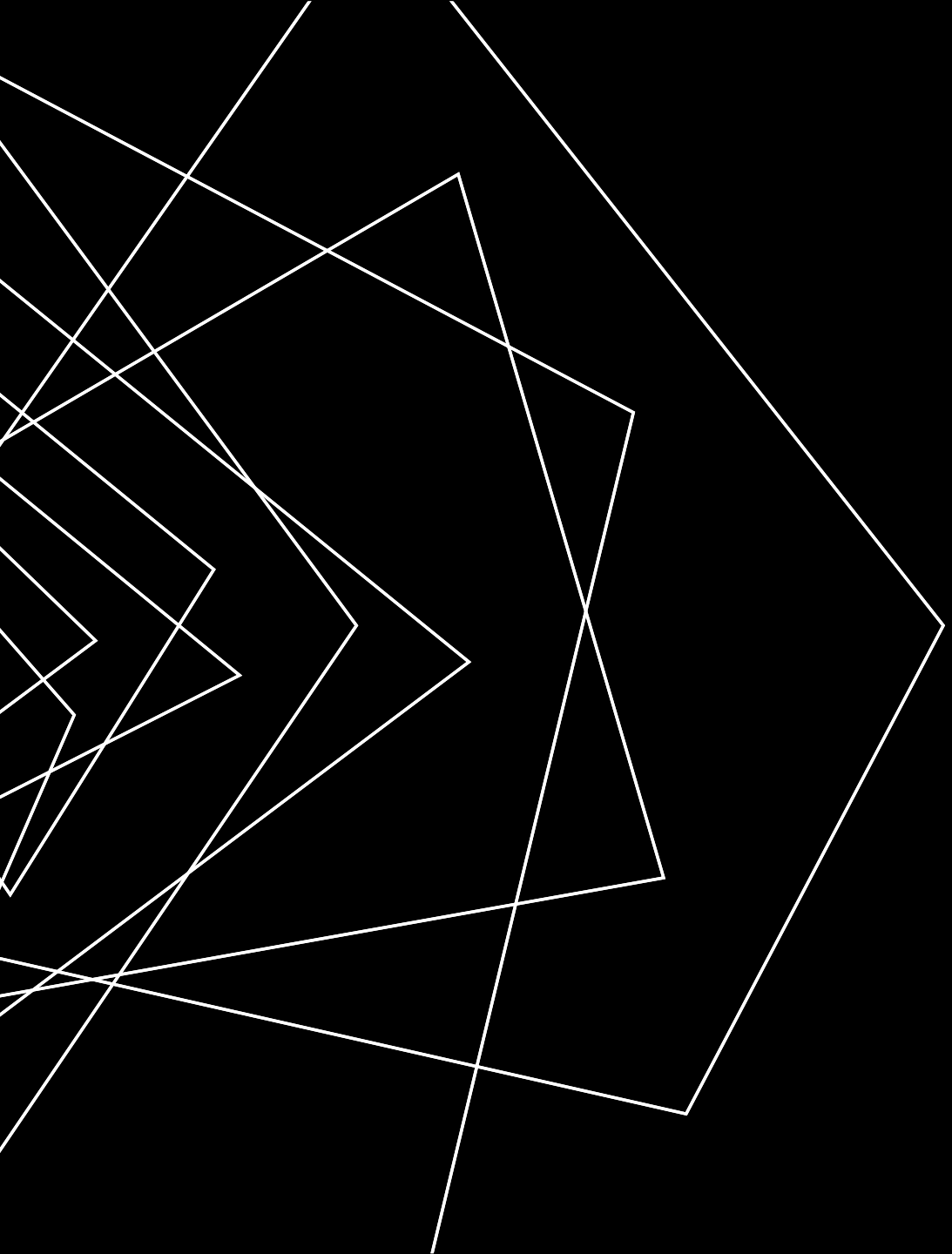


BATCH GRADIENT DESCENT



RESULT


Prediction
Actual
Predicted
Training
Prediction Start
Value
Time
600
500
400
300
200
100
0
20
40
60
80
100
120
140



THANK FOR LISTENING