Creating Regression Models using LSTM

Bahare Nasir

Context

Creating Regression Models using LSTM

Introduction:

- Regression models are essential tools for analyzing and predicting continuous numerical outcomes.
- LSTM (Long Short-Term Memory) models are a type of recurrent neural network (RNN) known for their ability to model sequential data.

Why LSTM for Regression?

- LSTM models are particularly well-suited for regression tasks that involve time series or sequence data.
- Unlike traditional feedforward neural networks, LSTM models can capture long-term dependencies and patterns in the data.
- They can handle input sequences of varying lengths and retain information over extended time intervals.

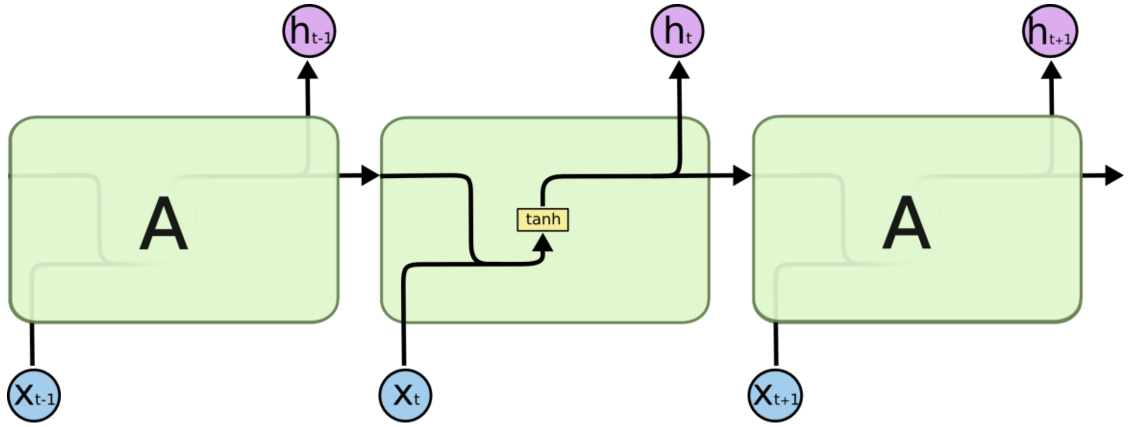
Creating Regression Models using LSTM

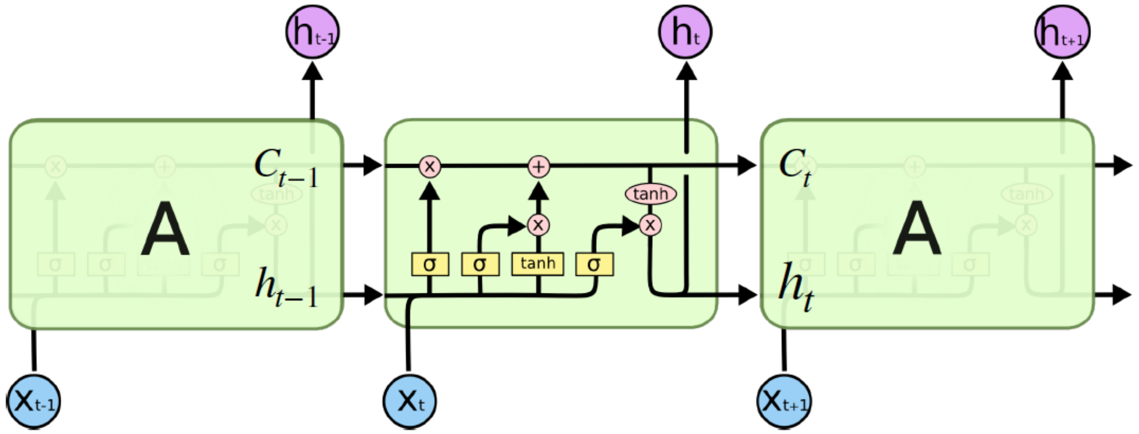
Key Advantages of LSTM Models:

- Ability to capture temporal dynamics
- Handling of variable-length sequence
- Robustness to noise and missing data
- Flexibility in architecture

Simple RNN vs LSTM:

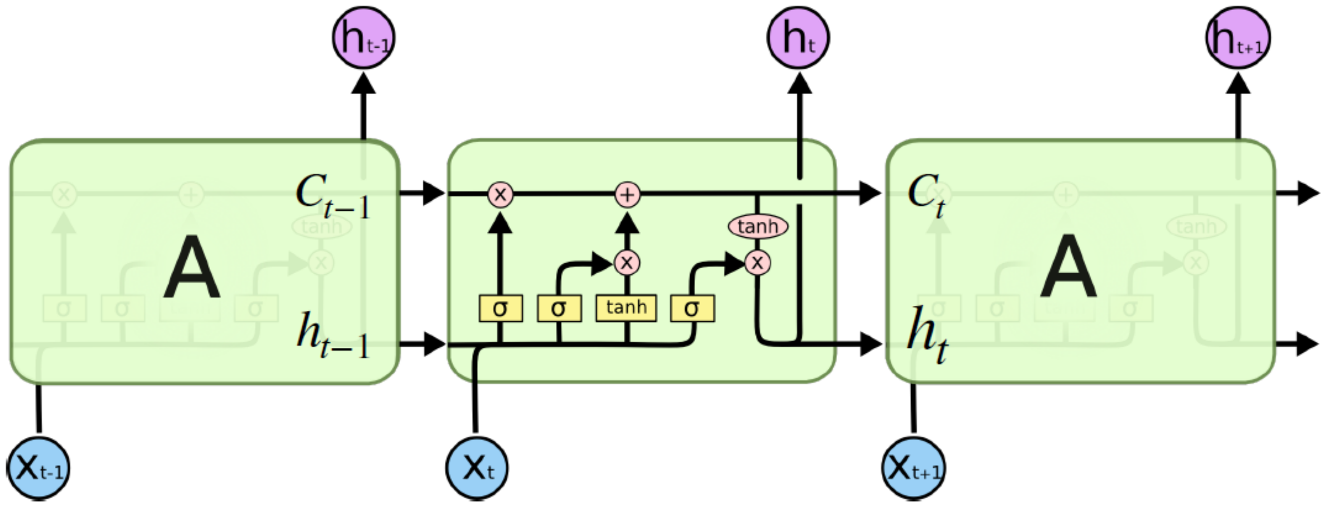
All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.





Long Short Term Memory (LSTM)

LSTM RNN (Hochreiter, '97) implements a "software trick": instead of having a single neural layer, it has four, which interact in such a way to implement a sort of parallel data-flow which at each step t makes the previous data available to each layer of the network being affected by gradient dilution



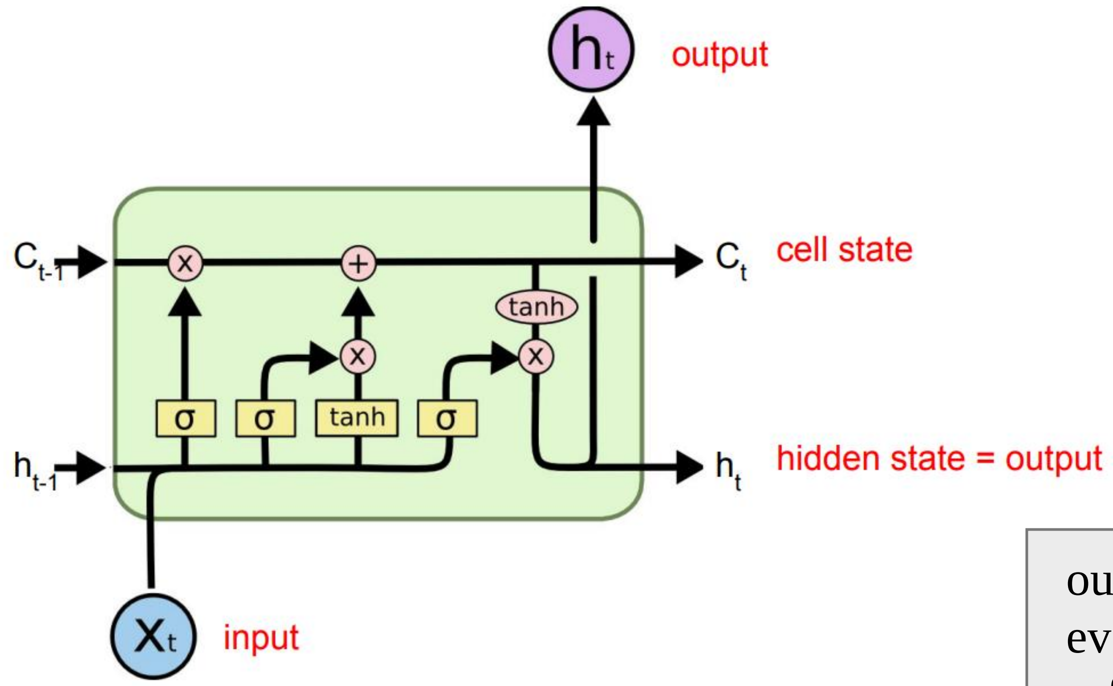
key element: cell-state C_t is a memory units (“conveyor belt”) to which is possible to add or subtract information using “gate” structures

the cell state holds long-term memory

the hidden states hold short-term memory

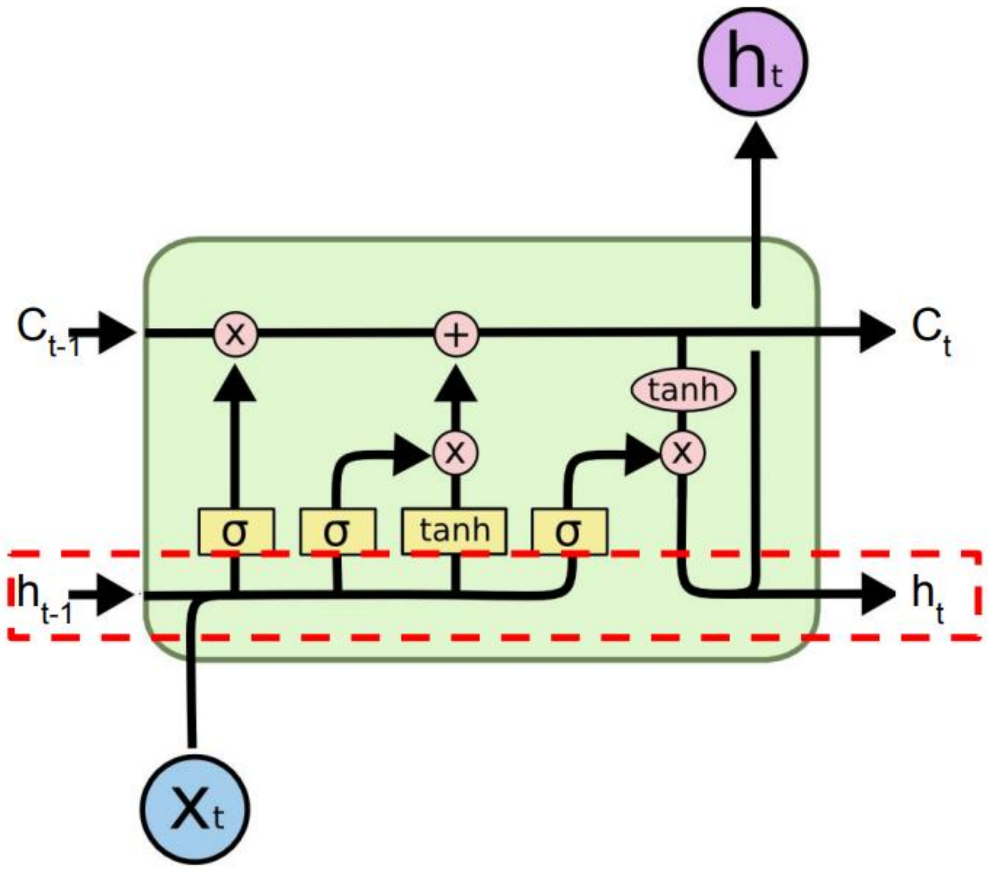
Long Short Term Memory (LSTM)

gate: NN-layer with sigmoid activation and a point-wise multiplication

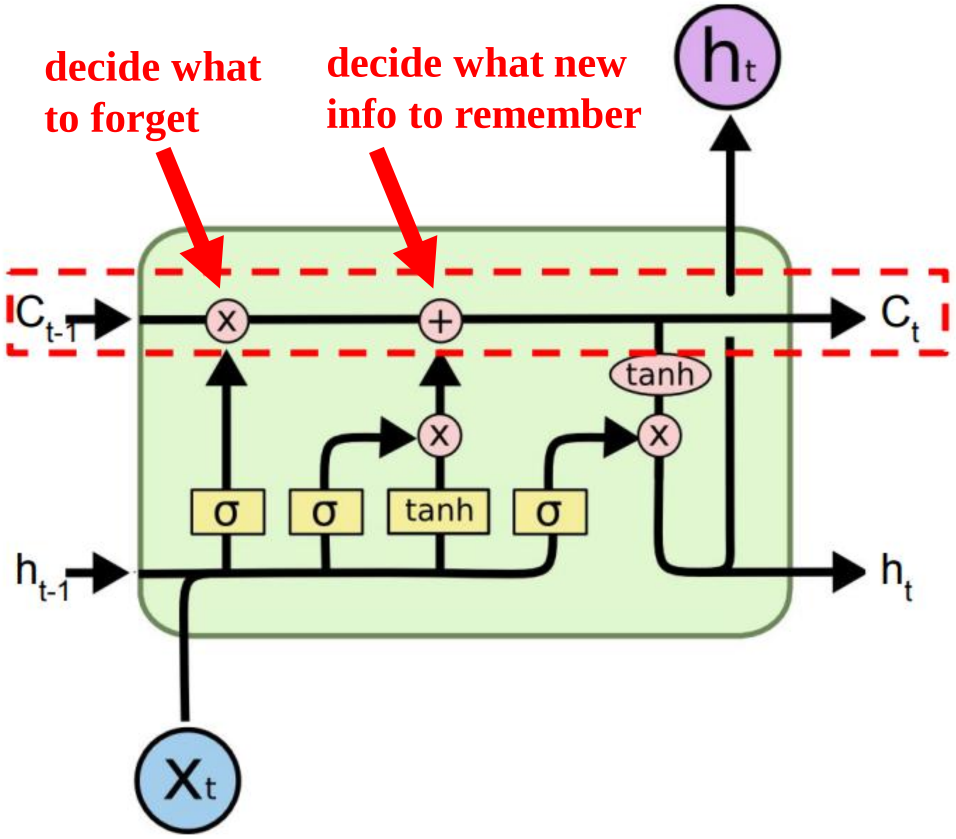


output ∈[0,1]
every LSTM has 3 gates:
- f: forget gate (controls deleting from the cell-state)
- i: input gate (controls writing on the cell-state)
- o: output gate (controls the output on ht)

Long Short Term Memory (LSTM): Hidden state



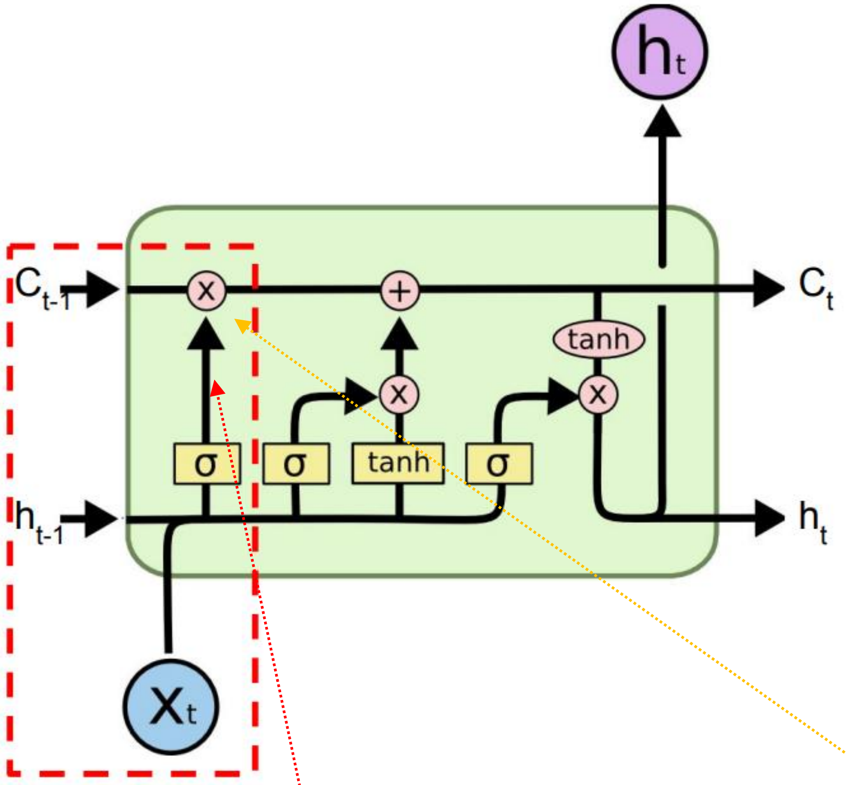
Long Short Term Memory (LSTM): Cell State



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

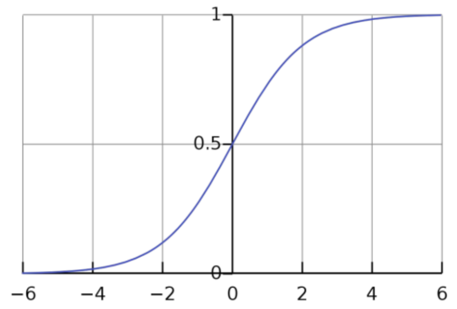
- Cell state updated twice
- Few computations -> stabilise gradients

Long Short Term Memory (LSTM): Forget components

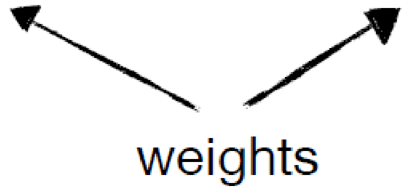


Decides which part of the previous information should be removed from the cell-state looks at h_{t-1} and x_t and produces a number in [0,1] for each value of the cell-state C_{t-1}

- 0: completely eliminates the previous information
- 1: completely retains the previous information

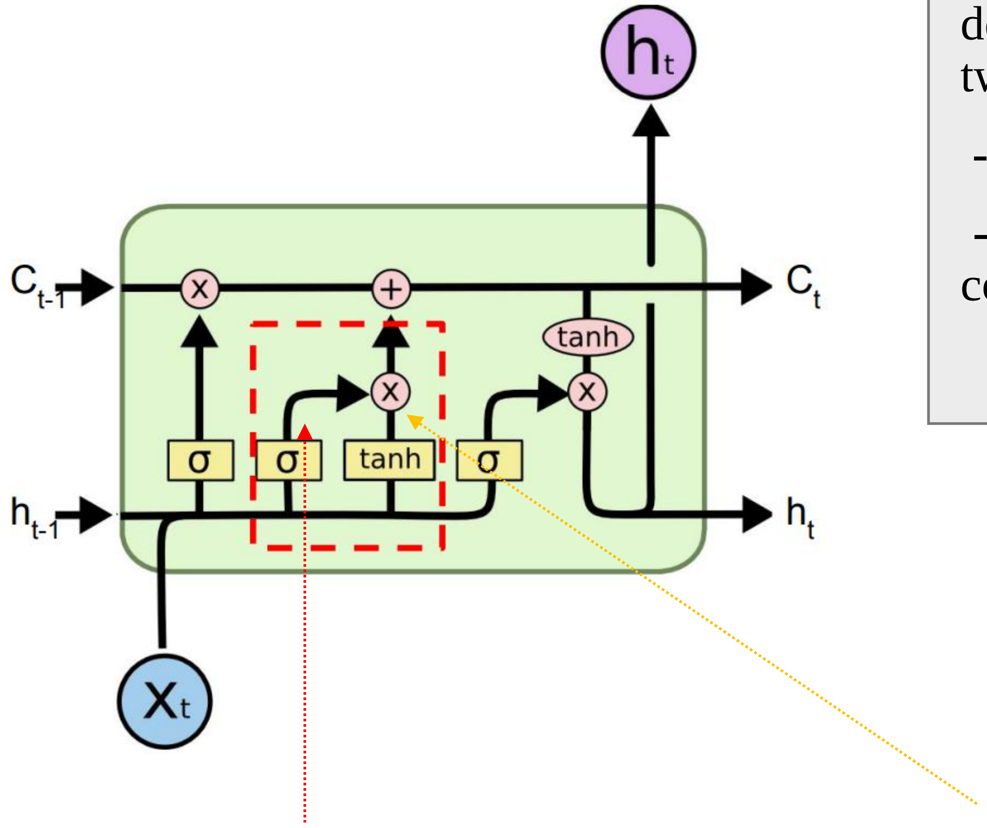


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$



$$C_t^f = C_{t-1} * f_t$$

Long Short Term Memory (LSTM): Input components



decides what new information to attach to the cell-state, acts in two steps:

- σ-layer "input gate layer" decides which value to update

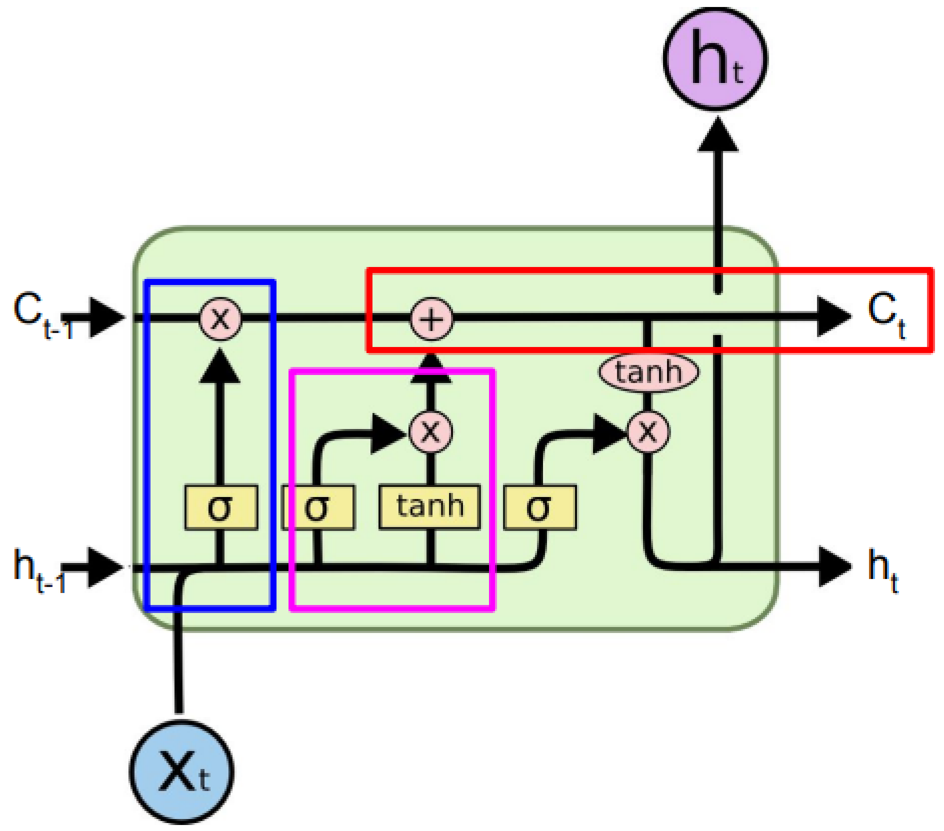
-a tanh layer creates a new vector of candidates C_t that potentially could be added to the cell-state

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

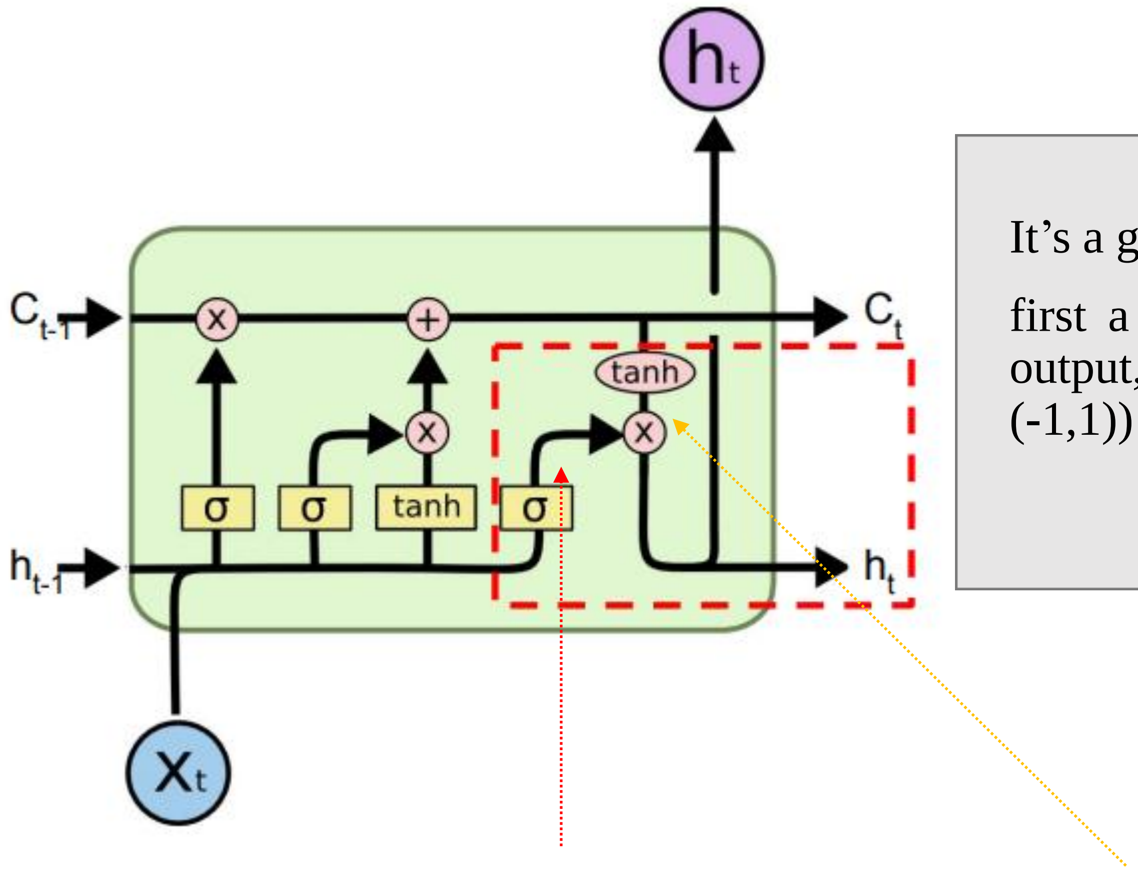
$$C_t^i = C_t' * i_t$$

Long Short Term Memory (LSTM): Cell state



$$\boxed{C_t} = \boxed{C_t^f} + \boxed{C_t^i}$$

Long Short Term Memory (LSTM): Output components



It's a gate based on a filtered version of the cell-state:

first a σ-layer decides which part of the cell-state to write to output, then the cell-state is evaluated via a tanh (compressed in (-1,1)) and multiplied by the gate output

$$o_t = \sigma \left(W_o \; [h_{t-1}, x_t] \; + \; b_o \right)$$

$$h_t = o_t * \tanh \left(C_t \right)$$

Data Preparation

Before building an LSTM regression model, it is crucial to properly prepare the data for training and evaluation.

Data preparation involves several steps to ensure the data is in a suitable format and represents the underlying problem accurately.

- Data Collection
- Data Cleaning
- Feature Selection and Engineering
- Train-Test Split
- Sequence Creation

Code explanation

(related to the data preparation section)

```
% Load the data
load('data2.mat')

% Data preparation
windowSize = 8;
numSamples = 2187 - windowSize;
X = zeros(5, windowSize, numSamples);
Y = zeros(2, numSamples);

for i = 1:numSamples
    X(:,:,i) = data2(i:i+windowSize-1, 1:5).';
    Y(:,i) = data2(i+windowSize, 6:7).';
end

cvVal = cvpartition(size(XTrainVal, 3), 'HoldOut', 0.2); % 80% for training, 20%
for validation
XTrain = XTrainVal(:, :, cvVal.training);
YTrain = YTrainVal(:, cvVal.training);

XValidation = XTrainVal(:, :, cvVal.test);
YValidation = YTrainVal(:, cvVal.test);
```

Model Architecture

The model architecture plays a crucial role in the performance and effectiveness of an LSTM regression model.

It defines the structure and connectivity of the LSTM layers, as well as the overall flow of information within the model.

- Sequence Input Layer
- Flatten Layer
- LSTM Layers
- Dropout Layer
- Activation Functions
- Fully Connected Layers
- Regression Layer

Activation Functions

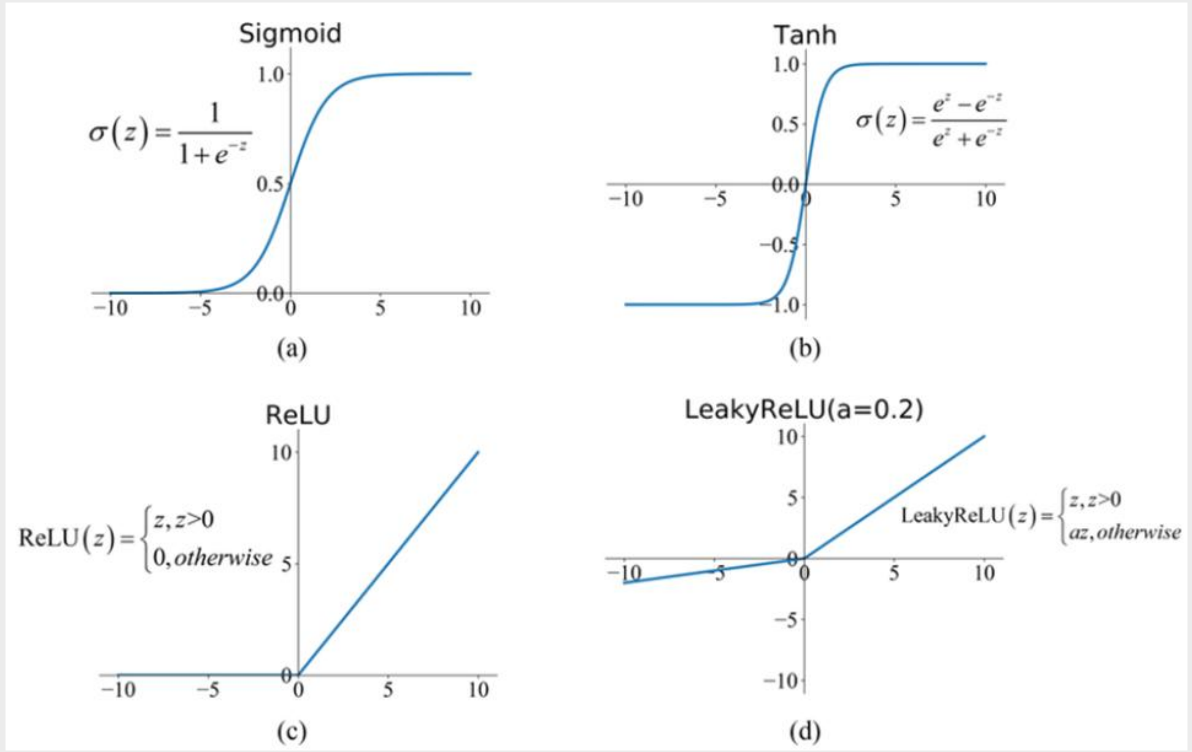
Activation functions are generally two types, These are:

Linear or Identity Activation Function

2. Non-Linear Activation Function.

Various non-linear activations in use:

- Sigmoid
- ReLU
- Leaky ReLU
- Tanh



Code explanation
(related to the Model Architecture)

```
% Define the activation function
activationFcn = 'leakyReLU';

% Define the layers with the activation function and dropout layer
layers = [
    sequenceInputLayer(inputSize, "Normalization","none")
    flattenLayer()
    lstmLayer(64, 'OutputMode', 'sequence')
    lstmLayer(32, 'OutputMode', 'sequence')
    dropoutLayer(0.01)
    fullyConnectedLayer(64)
    reluLayer('Name', activationFcn)
    fullyConnectedLayer(2)
    regressionLayer
];
```

Training

Training is a crucial step in building an LSTM regression model. During training, the model learns to map input sequences to corresponding output values by adjusting its internal parameters.

- Loss Function:
  - mean squared error (MSE)
  - mean absolute error (MAE)
- Optimization Algorithm:
  - Root mean square propagation (RMSprop)
  - Adaptive moment estimation (Adam)
  - Stochastic gradient descent (SGD)
- Mini-Batch Training
- Sequence Length
- Epochs
- Regularization

Code explanation
(related to Training)

```
inputSize = [5 windowSize];
miniBatchSize = 16;

% Training options
options = trainingOptions("adam", ...
    "ExecutionEnvironment", "cpu", ...
    "GradientThreshold", 1, ...
    "MaxEpochs", 100, ...
    "MiniBatchSize", miniBatchSize, ...
    "SequenceLength", "longest", ...
    "Shuffle", "never", ...
    "Plots", "training-progress", ...
    "ValidationData", {XValidation, YValidation}, ...
    "ValidationFrequency", 10, ...
    "ValidationPatience", 5, ... % Early stopping: Stop training if validation
loss doesn't improve for 5 epochs
    "L2Regularization", 0.001); % Apply L2 regularization

% Train the model
net = trainNetwork(XTrain, YTrain, layers, options);
```

Evaluation

In the context of LSTM regression, evaluation refers to assessing the performance and quality of the trained model. It involves measuring how well the model predicts the target outputs for unseen data. Here are some evaluation metrics commonly used for LSTM regression models:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- R-squared (Coefficient of Determination)
- Root mean squared error (RMSE)

Note:

During the evaluation phase, it is important to assess the model's performance on both the training and test datasets. This helps to understand if the model has learned the underlying patterns in the data and whether it can generalize well to unseen samples. Comparing the evaluation metrics between the training and test datasets can also indicate if the model is overfitting or underfitting.

Code explanation
(related to Evaluation)

```
% Calculate evaluation metrics
mseValues = mean((YPred - YTest).^2);
maeValues = mean(abs(YPred - YTest));
rSquaredValues = 1 - sum((YPred - YTest).^2) / sum((YTest -
mean(YTest)).^2);

% Average evaluation metrics over folds
mse = mean(mseValues);
mae = mean(maeValues);
rSquared = mean(rSquaredValues);

% Display evaluation metrics
disp("MSE: " + num2str(mse));
disp("MAE: " + num2str(mae));
disp("R-squared: " + num2str(rSquared));
```

Model Analysis and Fine-tuning

Once the LSTM regression model has been trained and evaluated, it's important to perform model analysis and fine-tuning to further improve its performance or address any potential issues. Here are some key aspects of model analysis and fine-tuning:

1- Model Analysis:

- Layer-by-Layer Evaluation
- Feature Importance

2- Fine-tuning:

- Hyperparameter Tuning:
  - learning rate
  - mini-batch size
  - number of hidden units
- Regularization Techniques:
  - regularization
  - Dropout

Prediction

After training the model using the provided training data and validating it using the validation data, we evaluate its performance on the test data. The predictions made by the model are compared with the actual values to assess the accuracy of the model's predictions.

The top part of the slide displays a line plot comparing the actual values (represented in blue) with the predicted values (represented by the red dashed line) for the first output. This plot gives a visual representation of how well the model's predictions align with the ground truth values. Similarly, the bottom part of the slide shows a similar line plot for the second output.

Code explanation
(Prediction)

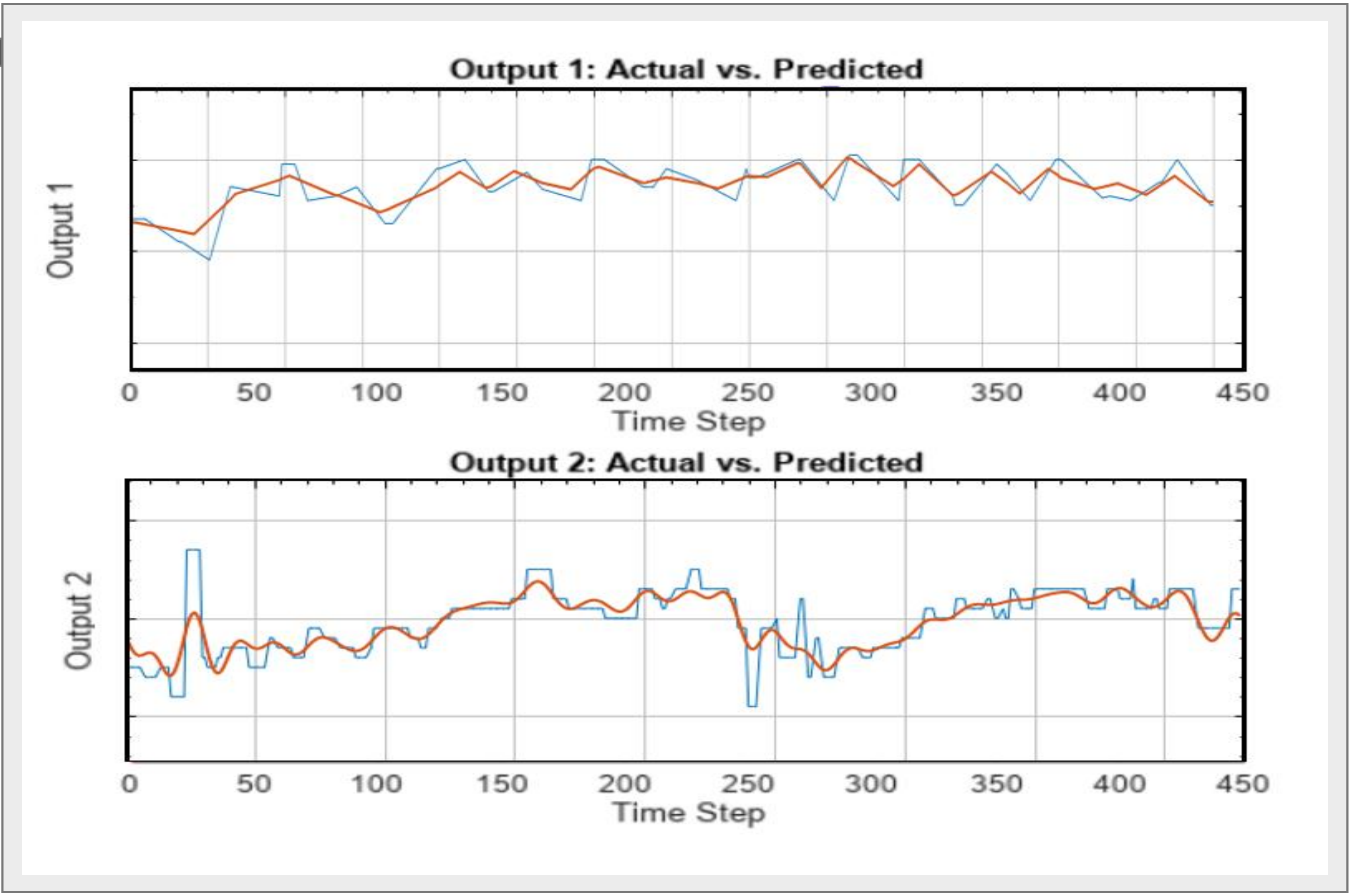
```
% Test the model
YPred = predict(net, XTest, ...
    "MiniBatchSize", miniBatchSize, ...
    "SequenceLength", "longest");
```

Code explanation (Prediction)

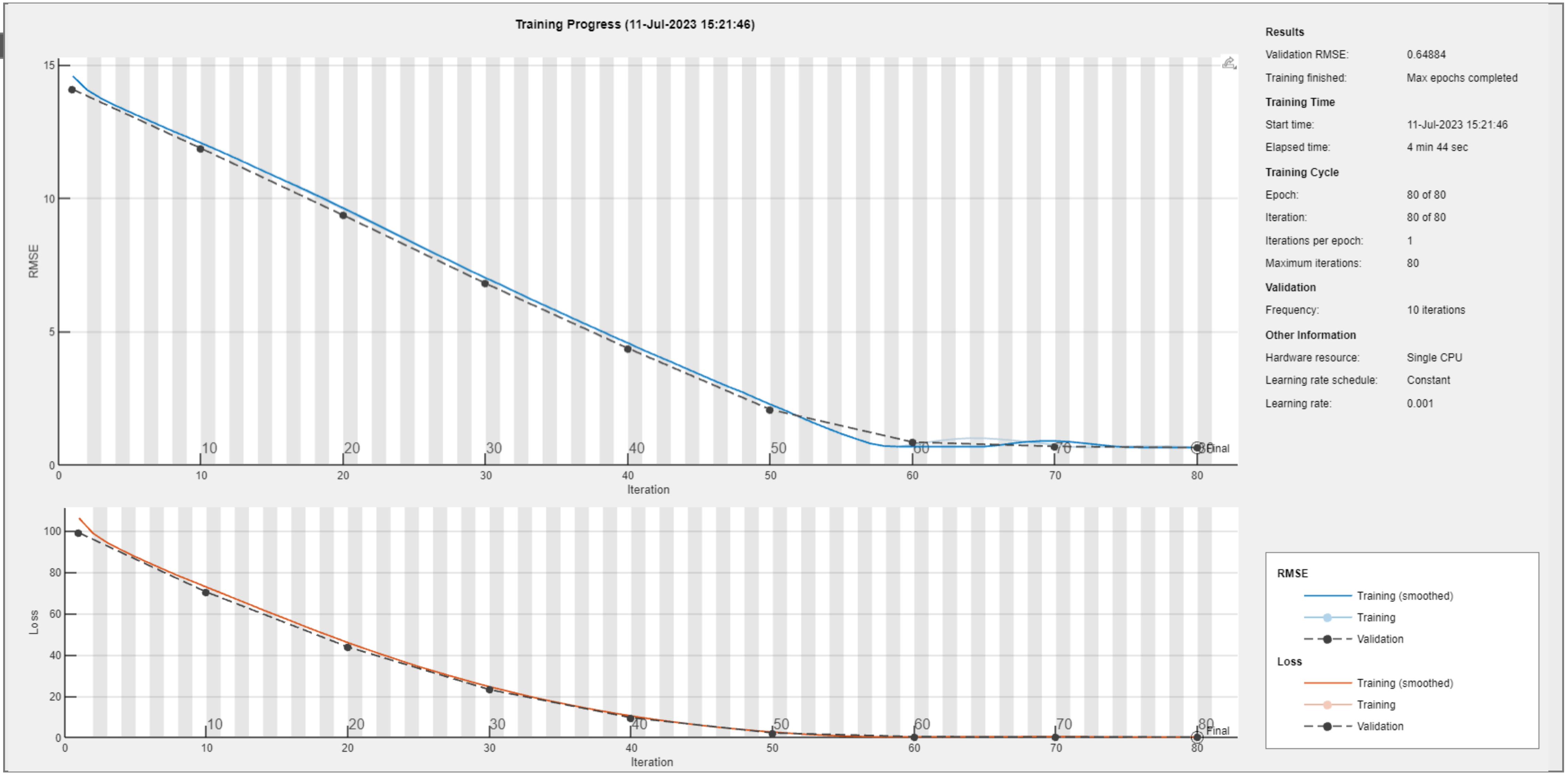
```
% Plot the results
figure;
subplot(2, 1, 1);
plot(YTest(1, :), 'b', 'LineWidth', 1.5);
hold on;
plot(YPred(1, :), 'r--', 'LineWidth', 1.5);
xlabel('Time Step');
ylabel('Output 1');
legend('Actual', 'Predicted');
title('Output 1: Actual vs. Predicted');

subplot(2, 1, 2);
plot(YTest(2, :), 'b', 'LineWidth', 1.5);
hold on;
plot(YPred(2, :), 'r--', 'LineWidth', 1.5);
xlabel('Time Step');
ylabel('Output 2');
```

Prediction:


Output 1: Actual vs. Predicted
Output 1
0
50
100
150
200
250
300
350
400
450
Time Step
Output 2: Actual vs. Predicted
Output 2
0
50
100
150
200
250
300
350
400
450
Time Step

Result:



Results

Validation RMSE: 0.64884

Training finished: Max epochs completed

Training Time

Start time: 11-Jul-2023 15:21:46

Elapsed time: 4 min 44 sec

Training Cycle

Epoch: 80 of 80

Iteration: 80 of 80

Iterations per epoch: 1

Maximum iterations: 80

Validation

Frequency: 10 iterations

Other Information

Hardware resource: Single CPU

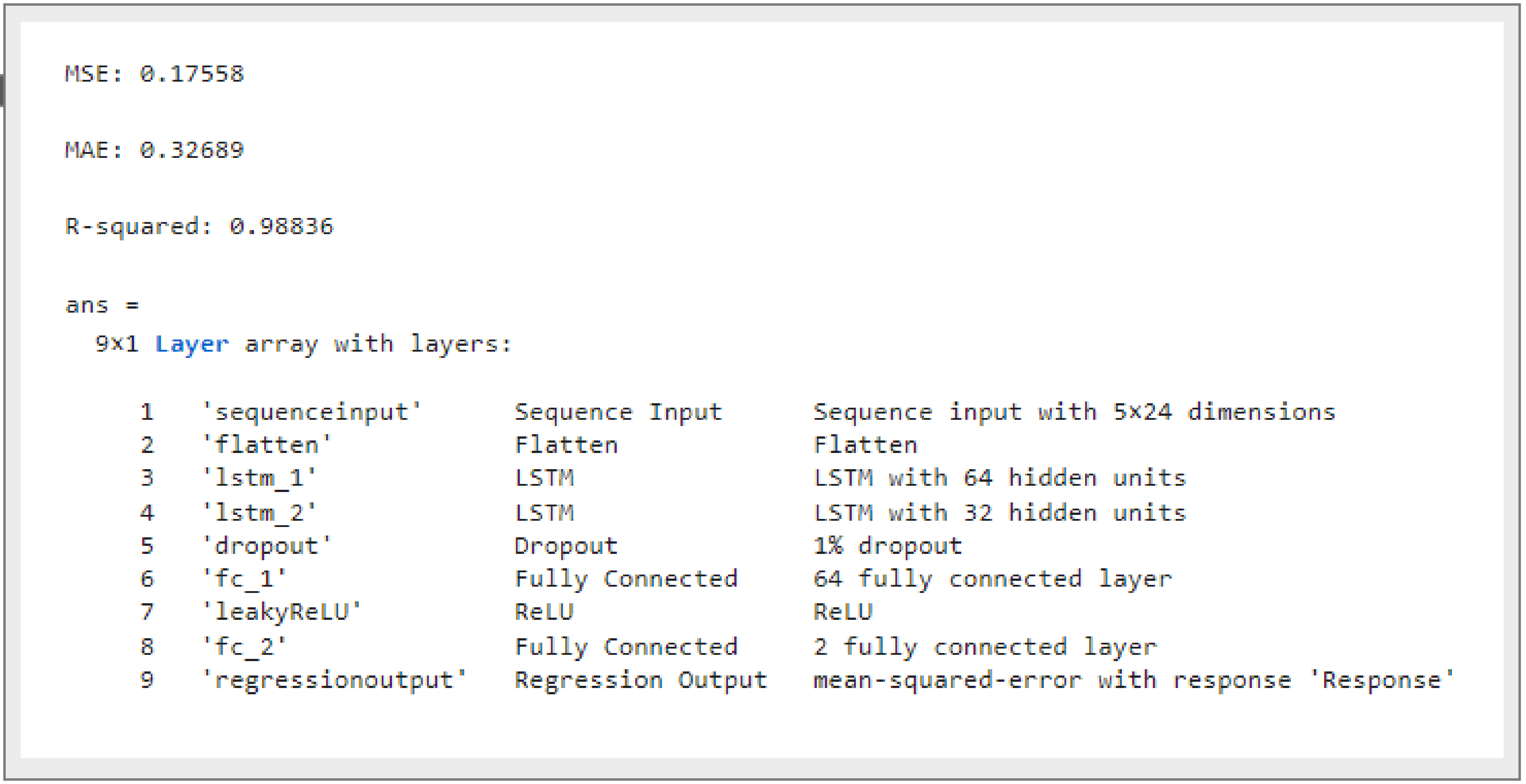
Learning rate schedule: Constant

Learning rate: 0.001

RMSE
- Training (smoothed)
- Training
- Validation

Loss
- Training (smoothed)
- Training
- Validation

Result:



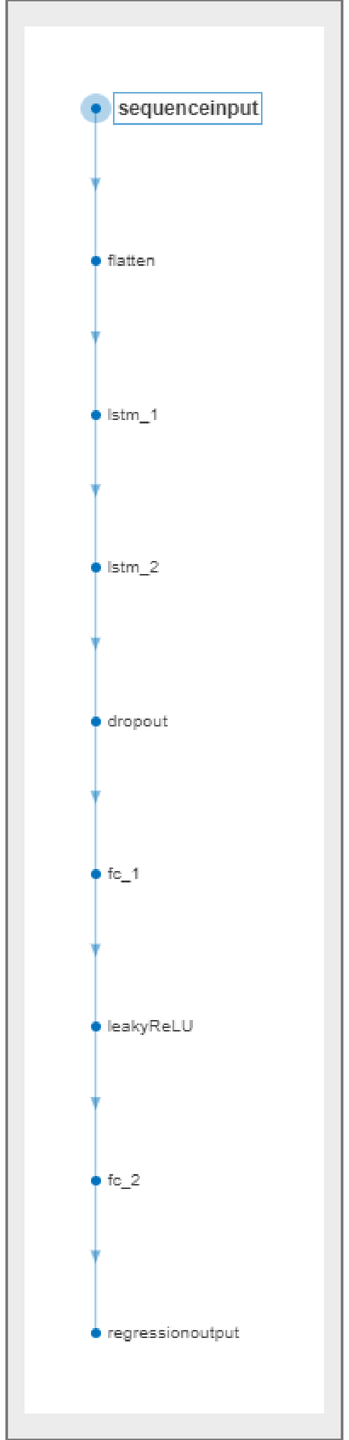
```
MSE: 0.17558

MAE: 0.32689

R-squared: 0.98836

ans =
  9x1 Layer array with layers:

     1   'sequenceinput'      Sequence Input      Sequence input with 5x24 dimensions
     2   'flatten'            Flatten             Flatten
     3   'lstm_1'             LSTM                LSTM with 64 hidden units
     4   'lstm_2'             LSTM                LSTM with 32 hidden units
     5   'dropout'            Dropout             1% dropout
     6   'fc_1'               Fully Connected     64 fully connected layer
     7   'leakyReLU'          ReLU                ReLU
     8   'fc_2'               Fully Connected     2 fully connected layer
     9   'regressionoutput'   Regression Output   mean-squared-error with response 'Response'
```



Conclusion:

Our LSTM regression model achieved excellent performance in predicting the target outputs.

- The mean squared error (MSE) was 0.17558, the mean absolute error (MAE) was 0.32689, and the R-squared value was 0.98836, indicating high accuracy and a strong relationship between input features and target outputs.
- The model's validation root mean squared error (RMSE) of 0.64884 demonstrated consistent accuracy on unseen data.
- Through rigorous validation techniques, we ensured the model's ability to generalize well to new data.
- The LSTM model holds great potential for various time series prediction and regression tasks.

Code:

```
% Load the data
load('data2.mat')

% Data preparation
windowSize = 8;
numSamples = 2187 - windowSize;
X = zeros(5, windowSize, numSamples);
Y = zeros(2, numSamples);

for i = 1:numSamples
    X(:,:,i) = data2(i:i+windowSize-1, 1:5).';
    Y(:,i) = data2(i+windowSize, 6:7).';
end

cvVal = cvpartition(size(XTrainVal, 3), 'HoldOut', 0.2); % 80%
for training, 20% for validation
XTrain = XTrainVal(:, :, cvVal.training);
YTrain = YTrainVal(:, cvVal.training);

XValidation = XTrainVal(:, :, cvVal.test);
YValidation = YTrainVal(:, cvVal.test);
```

```
% Feature scaling (min-max normalization)
XTrain = (XTrain - min(XTrain(:))) / (max(XTrain(:)) -
min(XTrain(:)));
XTest = (XTest - min(XTest(:))) / (max(XTest(:)) - min(XTest(:)));

% Data normalization
XTrain = normalize(XTrain, 'zscore');
XTest = normalize(XTest, 'zscore’);

inputSize = [5 windowSize];
miniBatchSize = 16;

% Define the activation function
activationFcn = 'leakyReLU';

% Define the layers with the activation function and dropout layer
layers = [
    sequenceInputLayer(inputSize, "Normalization","none")
    flattenLayer()
    lstmLayer(64, 'OutputMode', 'sequence')
    lstmLayer(32, 'OutputMode', 'sequence')
    dropoutLayer(0.01)
    fullyConnectedLayer(64)
    reluLayer('Name', activationFcn)
    fullyConnectedLayer(2)
    regressionLayer
];
```

Code:

```
% Xavier Initialization for fully connected layers
for i = 7:numel(layers)-2
    if isa(layers(i), 'fullyConnectedLayer')
        layers(i).Weights = sqrt(2 / (layers(i).InputSize(1) +
layers(i).OutputSize)) * randn(layers(i).OutputSize,
layers(i).InputSize(1));
    end
end

% Create a validation partition
cvVal = cvpartition(size(XTrain, 3), 'HoldOut', 0.2); % 80% for
training, 20% for validation
XValidation = XTrain(:, :, cvVal.test);
YValidation = YTrain(:, cvVal.test);
XTrain = XTrain(:, :, cvVal.training);
YTrain = YTrain(:, cvVal.training);

% Training options
options = trainingOptions("adam", ...
    "ExecutionEnvironment", "cpu", ...
    "GradientThreshold", 1, ...
    "MaxEpochs", 80, ...
    "MiniBatchSize", miniBatchSize, ...
    "SequenceLength", "longest", ...
    "Shuffle", "never", ...
    "Verbose", 0, ...
```

```
"Plots", "training-progress", ...
    "ValidationData", {XValidation, YValidation}, ...
    "ValidationFrequency", 10, ...
    "ValidationPatience", 5, ... % Early stopping: Stop training if
validation loss doesn't improve for 5 epochs
    "L2Regularization", 0.001); % Apply L2 regularization

% Train the model
net = trainNetwork(XTrain, YTrain, layers, options);

% Layer-by-layer evaluation
numLayers = numel(net.Layers);
layerOutputs = cell(1, numLayers);
X = XTest; % Use test data for evaluation

for i = 1:numLayers
    layerOutputs{i} = activations(net, X, i);
end


% Test the model
YPred = predict(net, XTest, ...
    "MiniBatchSize", miniBatchSize, ...
    "SequenceLength", "longest");
```

Code:

```
% Calculate evaluation metrics
mseValues = mean((YPred - YTest).^2);
maeValues = mean(abs(YPred - YTest));
rSquaredValues = 1 - sum((YPred - YTest).^2) / sum((YTest -
mean(YTest)).^2);

% Average evaluation metrics over folds
mse = mean(mseValues);
mae = mean(maeValues);
rSquared = mean(rSquaredValues);

% Display evaluation metrics
disp("MSE: " + num2str(mse));
disp("MAE: " + num2str(mae));
disp("R-squared: " + num2str(rSquared));

% Network analysis
analyzeNetwork(net)
```

```
% Examine the details of the network architecture contained in the
Layers property of net.
net.Layers

% Plot the results
figure;
subplot(2, 1, 1);
plot(YTest(1, :), 'b', 'LineWidth', 1.5);
hold on;
plot(YPred(1, :), 'r--', 'LineWidth', 1.5);
xlabel('Time Step');
ylabel('Output 1');
legend('Actual', 'Predicted');
title('Output 1: Actual vs. Predicted');
```

Code:

```
subplot(2, 1, 2);
plot(YTest(2, :), 'b', 'LineWidth', 1.5);
hold on;
plot(YPred(2, :), 'r--', 'LineWidth', 1.5);
xlabel('Time Step');
ylabel('Output 2');
legend('Actual', 'Predicted');
title('Output 2: Actual vs. Predicted');
```