Deep Learning Models for Text Processing - LSTM

- Classify Text with LSTM Models

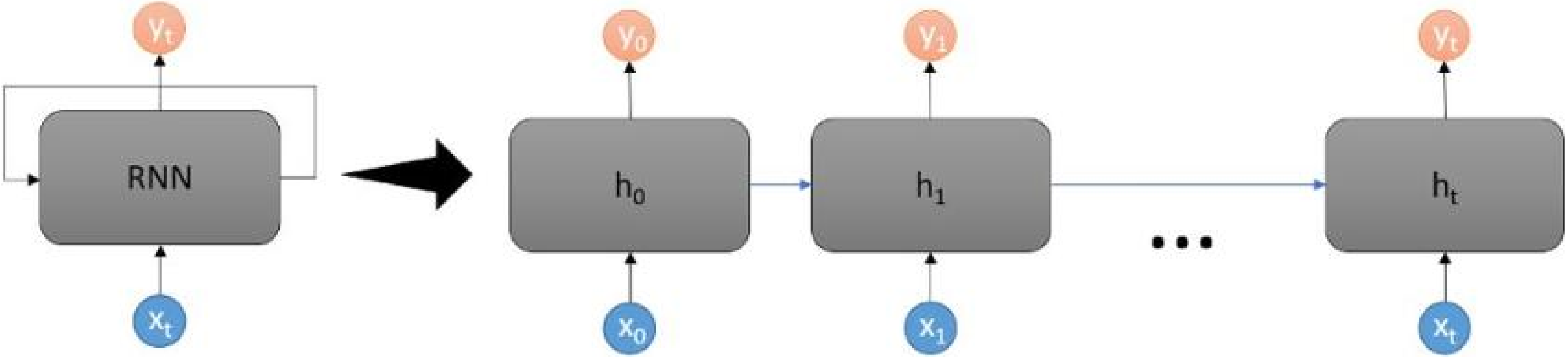
Outline

- About RNN and LSTM
- Long Short-Term Memory Networks
- Classify Text Data Using LSTM

RNN

- A recurrent neural network (RNN) is a deep learning network structure that uses information of the past to improve the performance of the network on current and future inputs. What makes RNNs unique is that the network contains a hidden state and loops. The looping structure allows the network to store past information in the hidden state and operate on sequences.

RNN



Unrolling a single cell of an RNN, showing how information moves through the network for a data sequence. Inputs are acted on by the hidden state of the cell to produce the output, and the hidden state is passed to the next time step.

RNN

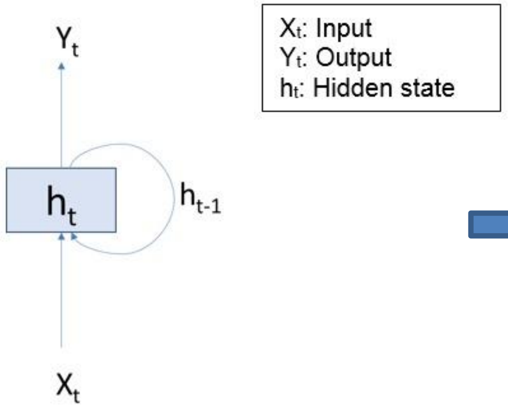
- How does the RNN know how to apply the past information to the current input? The network has two sets of weights, one for the hidden state vector and one for the inputs. During training, the network learns weights for both the inputs and the hidden state. When implemented, the output is based on the current input, as well as the hidden state, which is based on previous inputs.

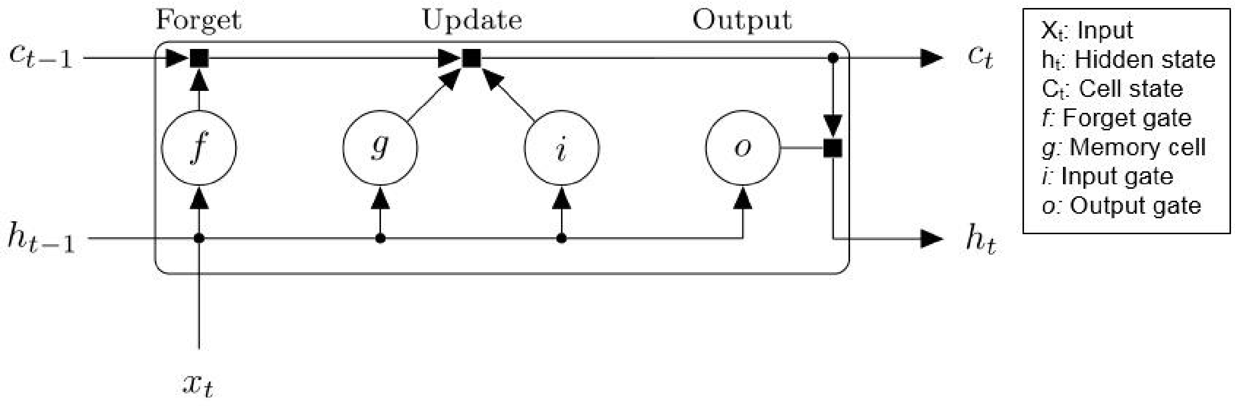
RNN

- In practice, simple RNNs experience a problem with learning longer-term dependencies. RNNs are commonly trained through backpropagation, where they can experience either a ‘vanishing’ or ‘exploding’ gradient problem. These problem cause the network weights to either become very small or very large, limiting the effectiveness of learning the long-term relationships.

RNN and LSTM

- A long short-term memory network is a type of recurrent neural network (RNN). LSTM networks use additional gates to control what information in the hidden cell makes it to the output and the next hidden state. This allows the network to more effectively learn long-term relationships in the data. LSTMs are a commonly implemented type of RNN.





Long Short-Term Memory Networks

- This section explains how to work with sequence and time series data for classification and regression tasks using long short-term memory (LSTM) networks.
- An LSTM network is a type of recurrent neural network (RNN) that can learn long-term dependencies between time steps of sequence data.

LSTM Network Architecture

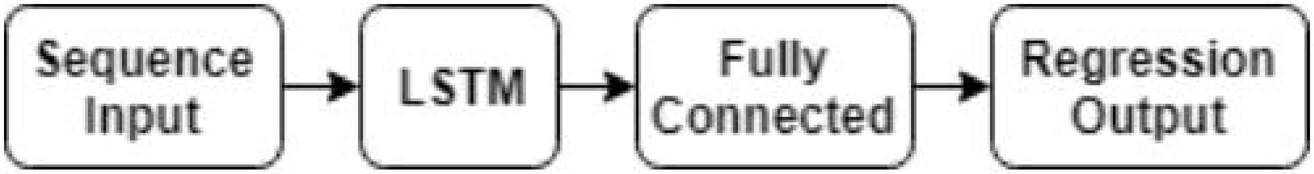
- The core components of an LSTM network are a sequence input layer and an LSTM layer. A sequence input layer inputs sequence or time series data into the network. An LSTM layer learns long-term dependencies between time steps of sequence data.

LSTM Network Architecture

- This diagram illustrates the architecture of a simple LSTM network for classification. The network starts with a sequence input layer followed by an LSTM layer. To predict class labels, the network ends with a fully connected layer, a softmax layer, and a classification output layer.

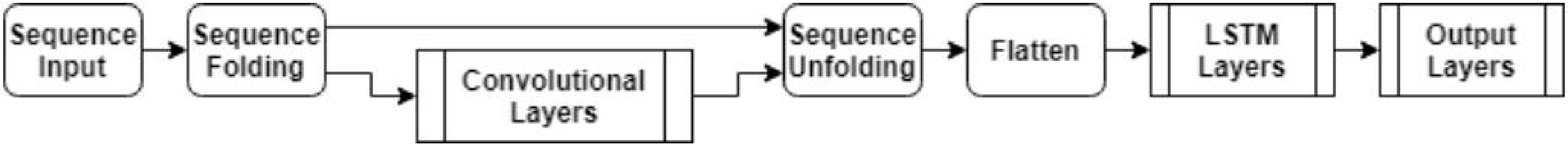
LSTM Network Architecture

- This diagram illustrates the architecture of a simple LSTM network for regression. The network starts with a sequence input layer followed by an LSTM layer. The network ends with a fully connected layer and a regression output layer.



LSTM Network Architecture

- This diagram illustrates the architecture of a network for video classification. To input image sequences to the network, use a sequence input layer. To use convolutional layers to extract features, use a sequence folding layer followed by the convolutional layers, and then a sequence unfolding layer. To use the LSTM layers to learn from sequences of vectors, use a flatten layer followed by the LSTM and output layers.



Classification LSTM Networks

- To create an LSTM network for sequence-to-label classification, create a layer array containing a sequence input layer, an LSTM layer, a fully connected layer, a softmax layer, and a classification output layer.
- Set the size of the sequence input layer to the number of features of the input data. Set the size of the fully connected layer to the number of classes. You do not need to specify the sequence length.
- For the LSTM layer, specify the number of hidden units and the output mode 'last'.

Classification LSTM Networks

```
numFeatures = 12;
numHiddenUnits = 100;
numClasses = 9;
layers = [ ...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits,'OutputMode','last')
 fullyConnectedLayer(numClasses)
 softmaxLayer
 classificationLayer];
```

Classification LSTM Networks

- To create an LSTM network for sequence-to-sequence classification, use the same architecture as for sequence-to-label classification, but set the output mode of the LSTM layer to 'sequence'.

```
numFeatures = 12;
numHiddenUnits = 100;
numClasses = 9;
layers = [ ...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits,'OutputMode','sequence')
 fullyConnectedLayer(numClasses)
 softmaxLayer
 classificationLayer];
```

Regression LSTM Networks

- To create an LSTM network for sequence-to-one regression, create a layer array containing a sequence input layer, an LSTM layer, a fully connected layer, and a regression output layer.
- Set the size of the sequence input layer to the number of features of the input data. Set the size of the fully connected layer to the number of responses. You do not need to specify the sequence length.
- For the LSTM layer, specify the number of hidden units and the output mode 'last'.

Regression LSTM Networks

```
numFeatures = 12;
numHiddenUnits = 125;
numResponses = 1;
layers = [ ...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits,'OutputMode','last')
 fullyConnectedLayer(numResponses)
 regressionLayer];
```

Regression LSTM Networks

- To create an LSTM network for sequence-to-sequence regression, use the same architecture as for sequence-to-one regression, but set the output mode of the LSTM layer to 'sequence'.

```
numFeatures = 12;
numHiddenUnits = 125;
numResponses = 1;
layers = [ ...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits,'OutputMode','sequence')
 fullyConnectedLayer(numResponses)
 regressionLayer];
```

Video Classification Network

- To create a deep learning network for data containing sequences of images such as video data and medical images, specify image sequence input using the sequence input layer.
- To use convolutional layers to extract features, that is, to apply the convolutional operations to each frame of the videos independently, use a sequence folding layer followed by the convolutional layers, and then a sequence unfolding layer. To use the LSTM layers to learn from sequences of vectors, use a flatten layer followed by the LSTM and output layers.

```
inputSize = [28 28 1];
filterSize = 5;
numFilters = 20;
numHiddenUnits = 200;
numClasses = 10;
layers = [ ...
 sequenceInputLayer(inputSize,'Name','input')
 sequenceFoldingLayer('Name','fold')
 convolution2dLayer(filterSize,numFilters,'Name','conv')
 batchNormalizationLayer('Name','bn')
 reluLayer('Name','relu')
 sequenceUnfoldingLayer('Name','unfold')
 flattenLayer('Name','flatten')
 lstmLayer(numHiddenUnits,'OutputMode','last','Name','lstm')
 fullyConnectedLayer(numClasses, 'Name','fc')
 softmaxLayer('Name','softmax')
 classificationLayer('Name','classification')];
```

Video Classification Network

- Convert the layers to a layer graph and connect the miniBatchSize output of the sequence folding layer to the corresponding input of the sequence unfolding layer.

```
lgraph = layerGraph(layers);
lgraph = connectLayers(lgraph,'fold/miniBatchSize','unfold/miniBatchSize');
```

Deeper LSTM Networks

- You can make LSTM networks deeper by inserting extra LSTM layers with the output mode 'sequence' before the LSTM layer. To prevent overfitting, you can insert dropout layers after the LSTM layers.
- For sequence-to-label classification networks, the output mode of the last LSTM layer must be 'last'.

Deeper LSTM Networks

```
numFeatures = 12;
numHiddenUnits1 = 125;
numHiddenUnits2 = 100;
numClasses = 9;
layers = [ ...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits1,'OutputMode','sequence')
 dropoutLayer(0.2)
 lstmLayer(numHiddenUnits2,'OutputMode','last')
 dropoutLayer(0.2)
 fullyConnectedLayer(numClasses)
 softmaxLayer
 classificationLayer];
```

- For sequence-to-sequence classification networks, the output mode of the last LSTM layer must be 'sequence'.

```
numFeatures = 12;
numHiddenUnits1 = 125;
numHiddenUnits2 = 100;
numClasses = 9;
layers = [ ...
 sequenceInputLayer(numFeatures)
 lstmLayer(numHiddenUnits1,'OutputMode','sequence')
 dropoutLayer(0.2)
 lstmLayer(numHiddenUnits2,'OutputMode','sequence')
 dropoutLayer(0.2)
 fullyConnectedLayer(numClasses)
 softmaxLayer
 classificationLayer];
```

Sequence Padding, Truncation, and Splitting

- LSTM networks support input data with varying sequence lengths. When passing data through the network, the software pads, truncates, or splits sequences so that all the sequences in each minibatch have the specified length. You can specify the sequence lengths and the value used to pad the sequences using the *SequenceLength* and *SequencePaddingValue* name-value pair arguments in trainingOptions.

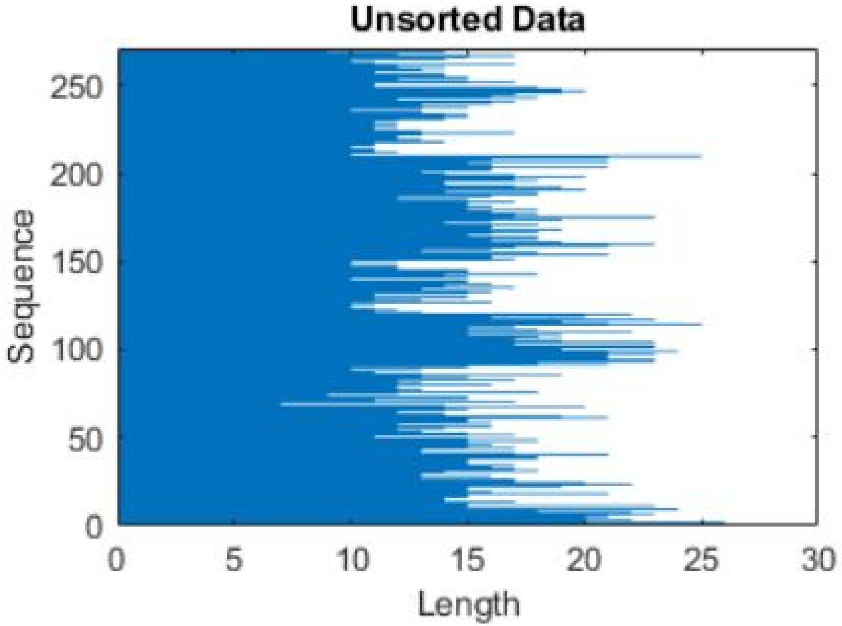
Sort Sequences by Length

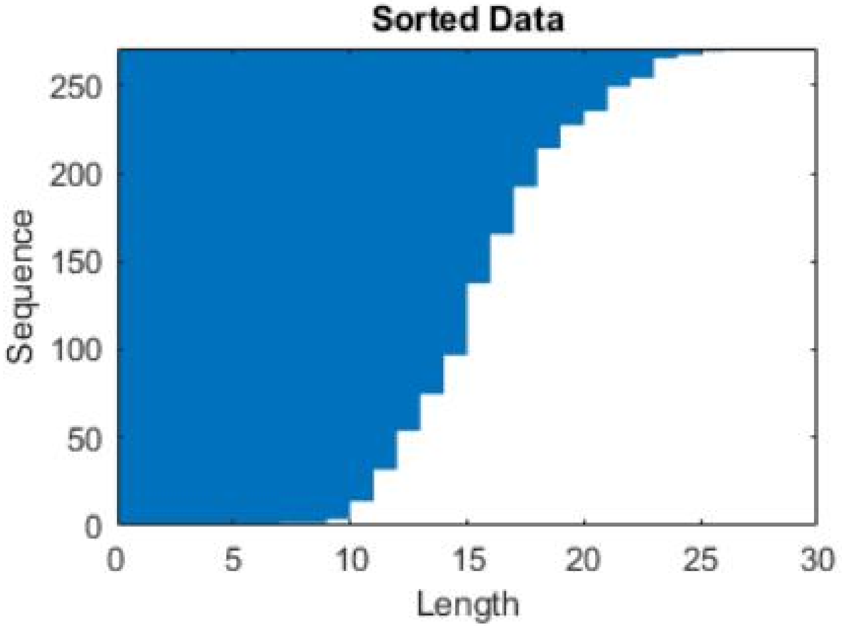
- To reduce the amount of padding or discarded data when padding or truncating sequences, try sorting your data by sequence length. To sort the data by sequence length, first get the number of columns of each sequence by applying size(X,2) to every sequence using cellfun. Then sort the sequence lengths using sort, and use the second output to reorder the original sequences.

```
sequenceLengths = cellfun(@(X) size(X,2), XTrain);
[sequenceLengthsSorted,idx] = sort(sequenceLengths);
XTrain = XTrain(idx);
```

Sort Sequences by Length

- The following figures show the sequence lengths of the sorted and unsorted data in bar charts.



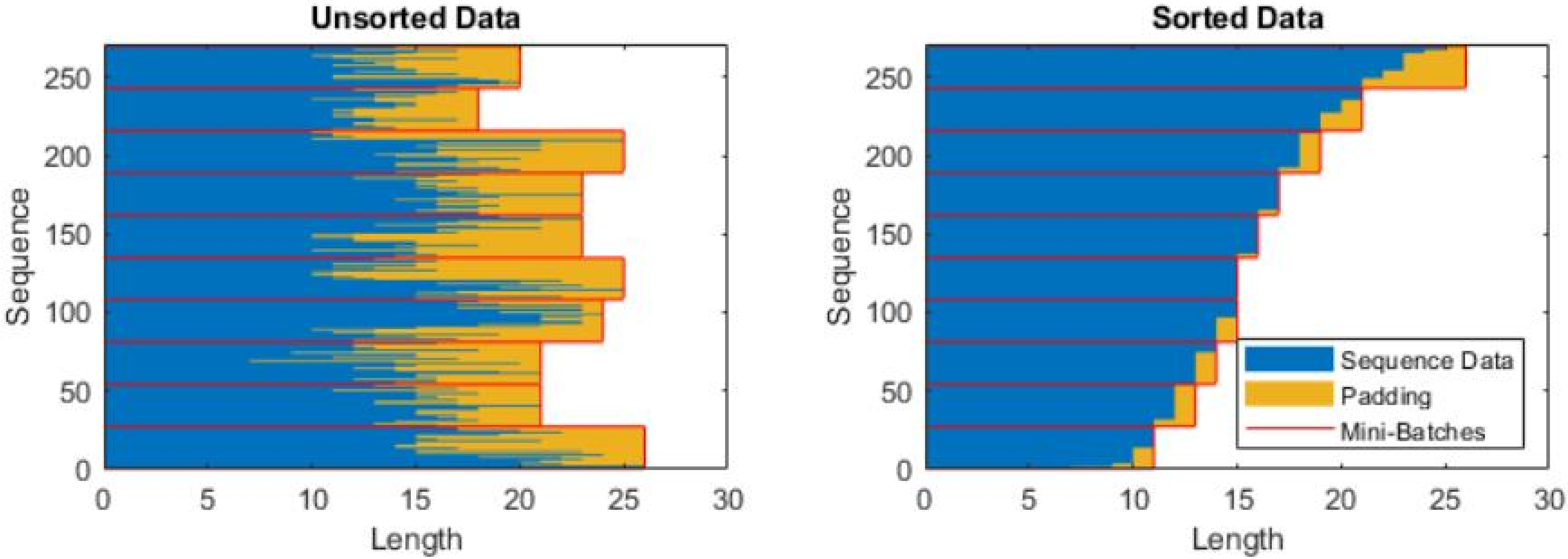


Pad Sequences

- If you specify the sequence length 'longest', then the software pads the sequences so that all the sequences in a mini-batch have the same length as the longest sequence in the mini-batch. This option is the default.

Pad Sequences

- The following figures illustrate the effect of setting 'SequenceLength' to 'longest'.

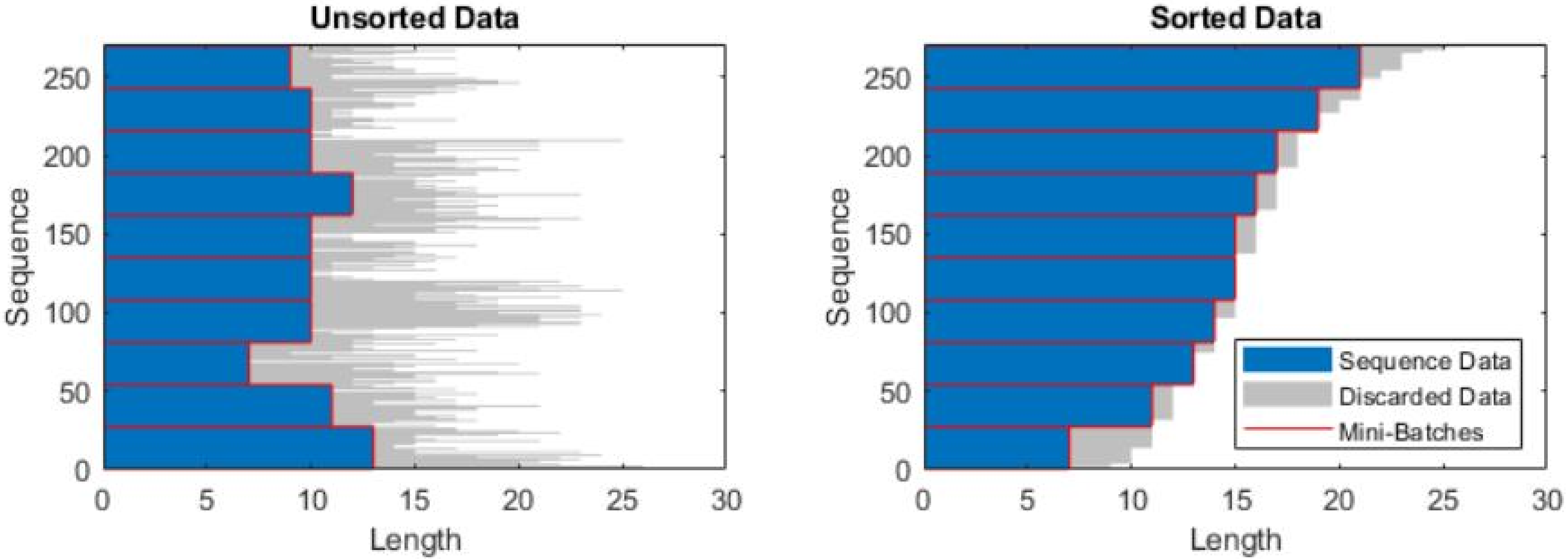


Truncate Sequences

- If you specify the sequence length 'shortest', then the software truncates the sequences so that all the sequences in a mini-batch have the same length as the shortest sequence in that mini-batch. The remaining data in the sequences is discarded.

Truncate Sequences

- The following figures illustrate the effect of setting 'SequenceLength' to 'shortest'.



Split Sequences

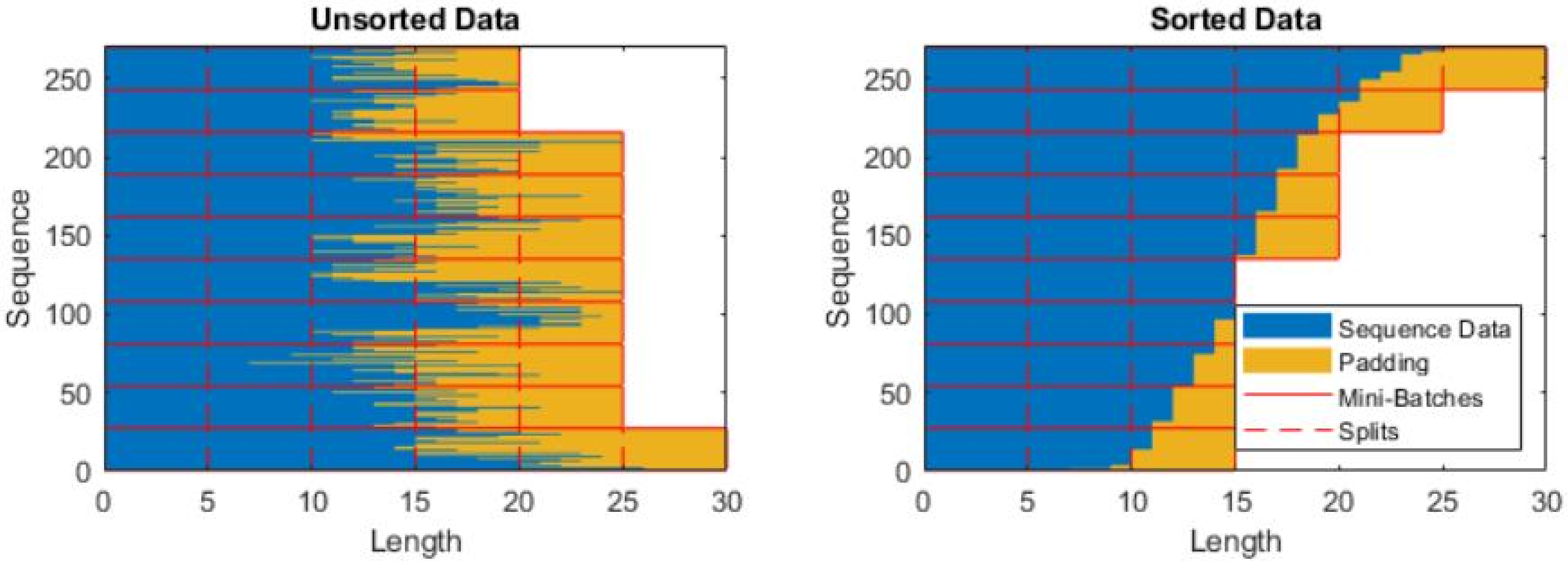
- If you set the sequence length to an integer value, then software pads all the sequences in a minibatch to the nearest multiple of the specified length that is greater than the longest sequence length in the mini-batch. Then, the software splits each sequence into smaller sequences of the specified length. If splitting occurs, then the software creates extra mini-batches.

Split Sequences

- Use this option if the full sequences do not fit in memory. Alternatively, you can try reducing the number of sequences per mini-batch by setting the 'MiniBatchSize' option in trainingOptions to a lower value.
- If you specify the sequence length as a positive integer, then the software processes the smaller sequences in consecutive iterations. The network updates the network state between the split sequences.

Split Sequences

- The following figures illustrate the effect of setting 'SequenceLength' to 5.



Specify Padding Direction

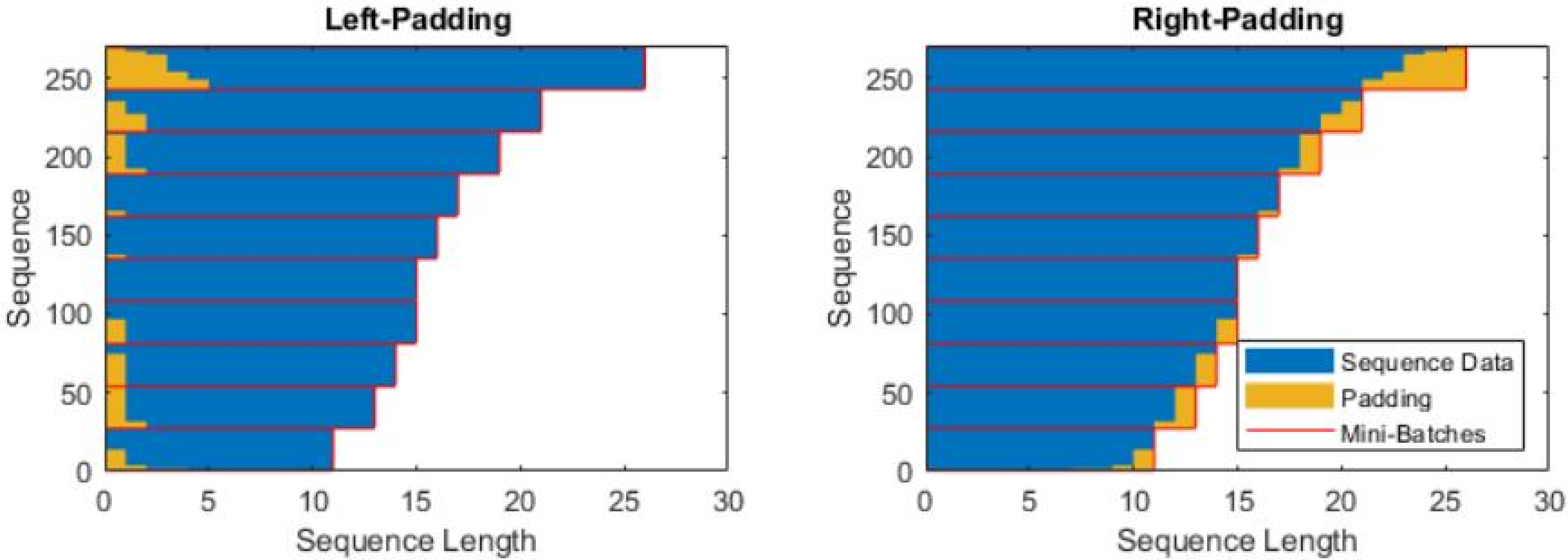
- The location of the padding and truncation can impact training, classification, and prediction accuracy. Try setting the 'SequencePaddingDirection' option in trainingOptions to 'left' or 'right' and see which is best for your data.
- Because LSTM layers process sequence data one time step at a time, when the layer OutputMode property is 'last', any padding in the final time steps can negatively influence the layer output. To pad or truncate sequence data on the left, set the 'SequencePaddingDirection' option to 'left'.

Specify Padding Direction

- For sequence-to-sequence networks (when the OutputMode property is 'sequence' for each LSTM layer), any padding in the first time steps can negatively influence the predictions for the earlier time steps. To pad or truncate sequence data on the right, set the 'SequencePaddingDirection' option to 'right'.

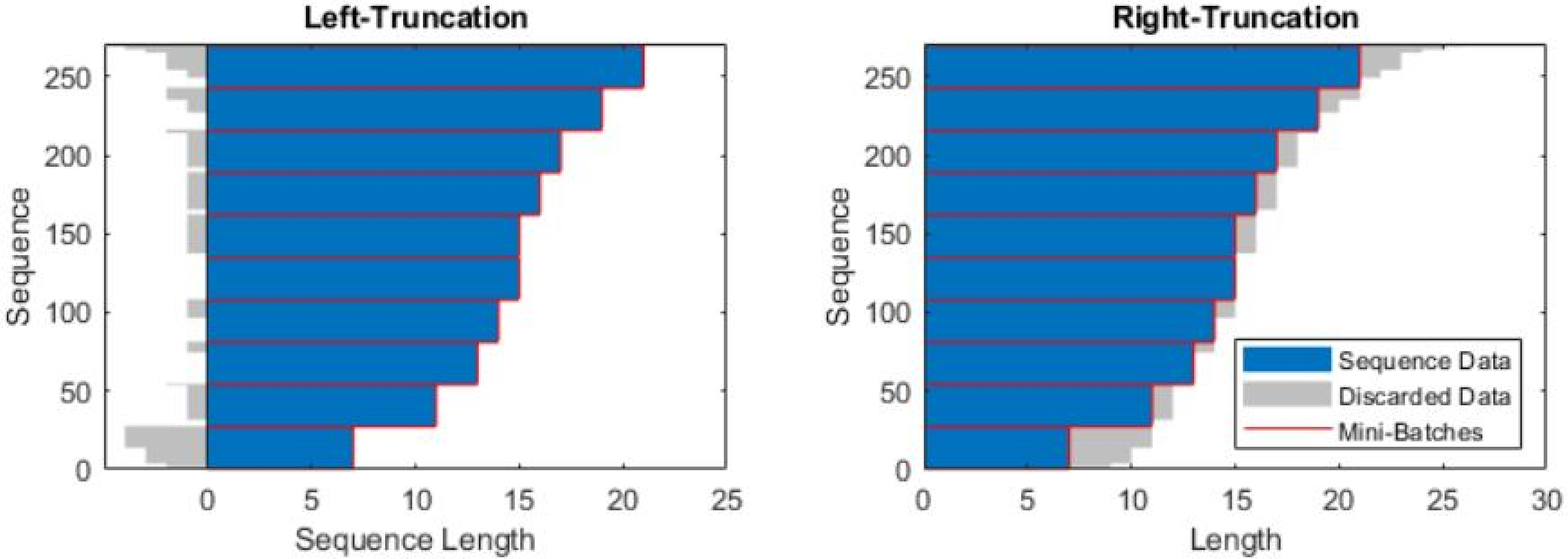
Specify Padding Direction

- The following figures illustrate padding sequence data on the left and on the right.

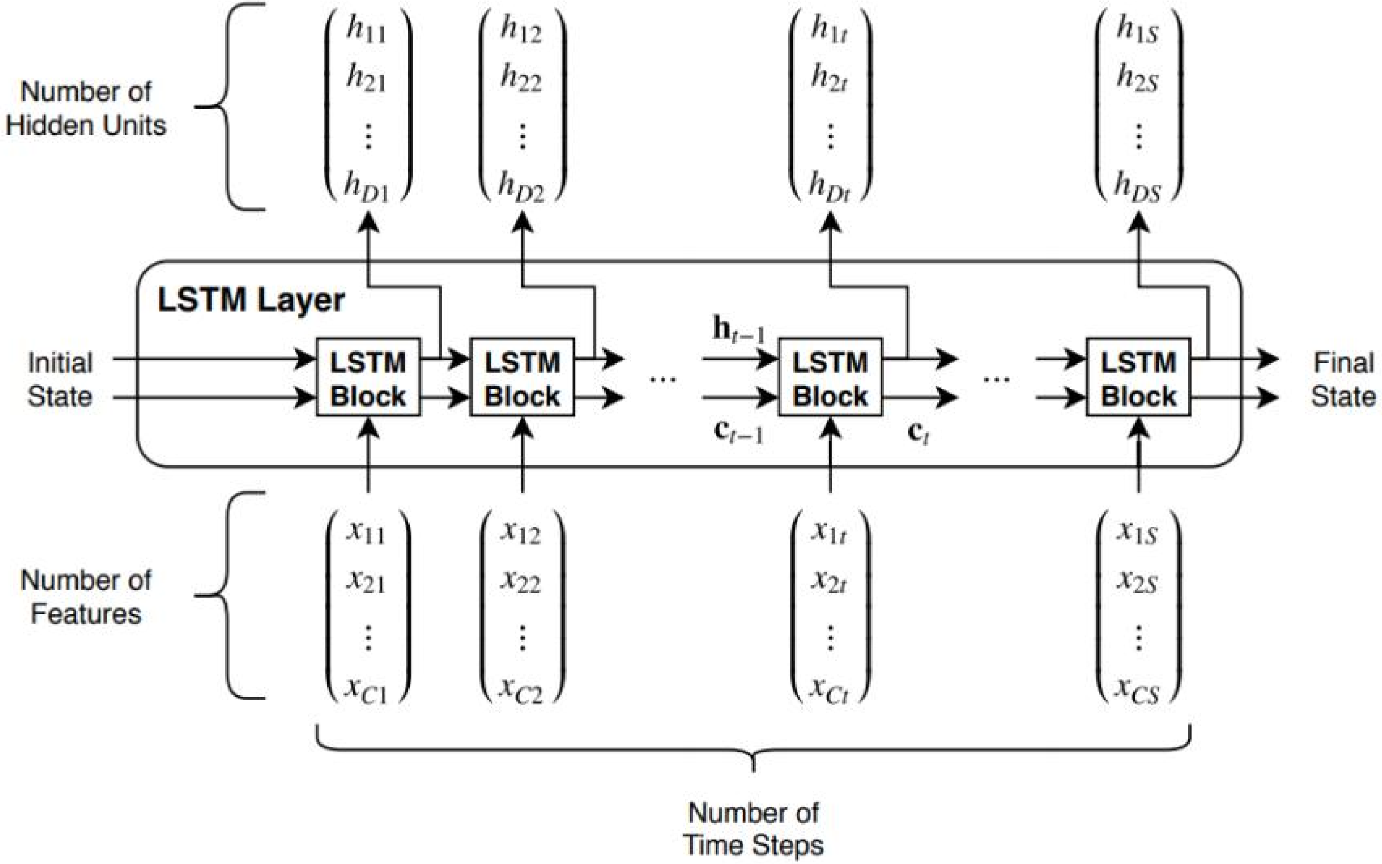


Specify Padding Direction

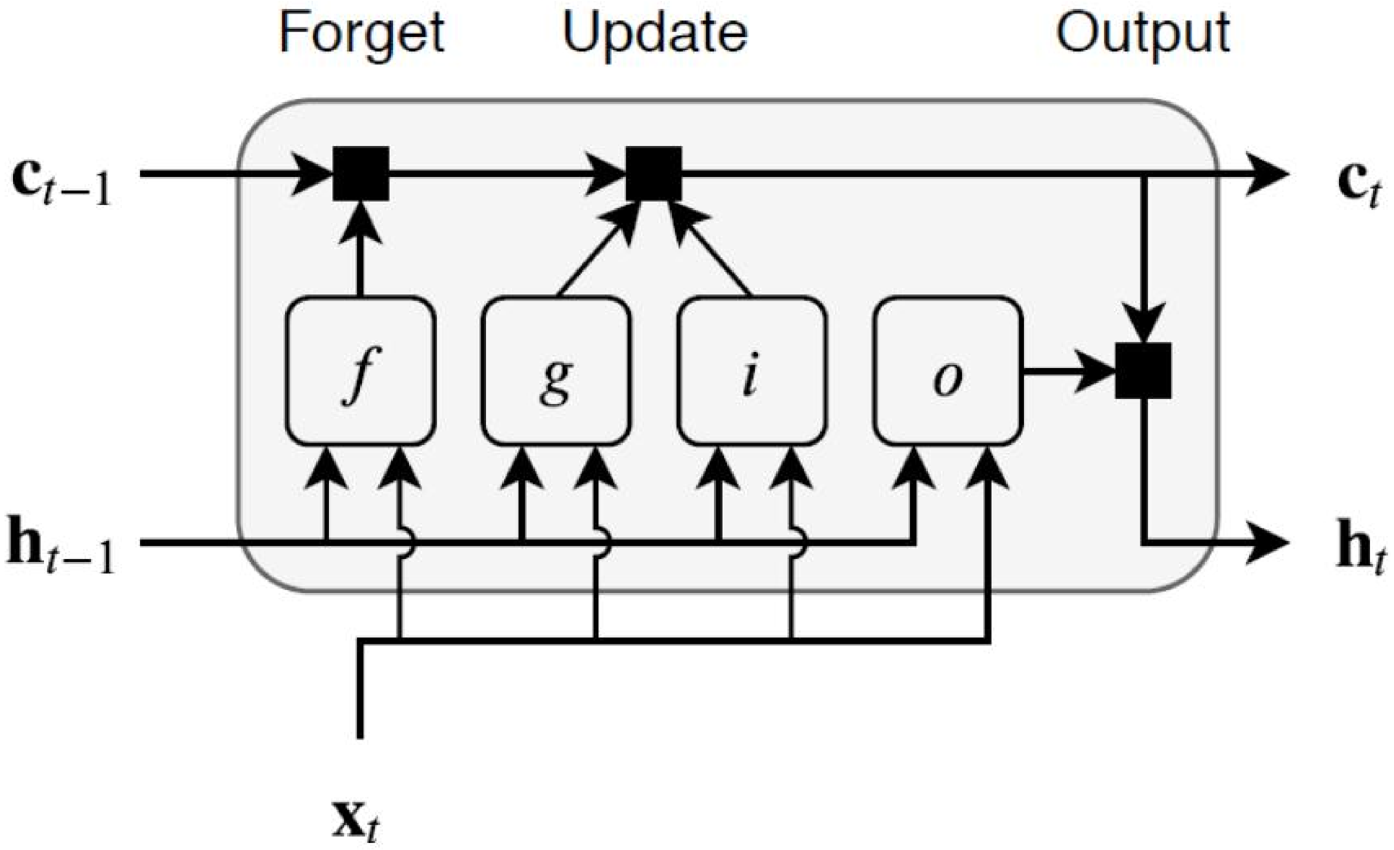
- The following figures illustrate truncating sequence data on the left and on the right.



LSTM Layer Architecture



LSTM Layer Architecture



LSTM Layer Architecture

- The learnable weights of an LSTM layer are the input weights W (InputWeights), the recurrent weights R (RecurrentWeights), and the bias b (Bias). The matrices W, R, and b are concatenations of the input weights, the recurrent weights, and the bias of each component, respectively.

Component	Formula
Input gate	$i_t = \sigma_g(W_i \mathbf{x}_t + \mathrm{R}_i \mathbf{h}_{t-1} + b_i)$
Forget gate	$f_t = \sigma_g(W_f \mathbf{x}_t + \mathrm{R}_f \mathbf{h}_{t-1} + b_f)$
Cell candidate	$g_t = \sigma_c(W_g \mathbf{x}_t + \mathrm{R}_g \mathbf{h}_{t-1} + b_g)$
Output gate	$o_t = \sigma_g(W_o \mathbf{x}_t + \mathrm{R}_o \mathbf{h}_{t-1} + b_o)$

$$\mathbf{c}_t = f_t \odot \mathbf{c}_{t-1} + i_t \odot g_t, \qquad \mathbf{h}_t = o_t \odot \sigma_c(\mathbf{c}_t),$$

Classify Text Data Using LSTM

- This example shows how to classify text data using a deep learning long short-term memory (LSTM) network.
- Text data is naturally sequential. A piece of text is a sequence of words, which might have dependencies between them. To learn and use long-term dependencies to classify sequence data, use an LSTM neural network. An LSTM network is a type of recurrent neural network (RNN) that can learn long-term dependencies between time steps of sequence data.

Classify Text Data Using LSTM

- To input text to an LSTM network, first convert the text data into numeric sequences. You can achieve this using a word encoding which maps documents to sequences of numeric indices. For better results, also include a word embedding layer in the network. Word embeddings map words in a vocabulary to numeric vectors rather than scalar indices. These embeddings capture semantic details of the words, so that words with similar meanings have similar vectors. They also model relationships between words through vector arithmetic.

Classify Text Data Using LSTM

There are four steps in training and using the LSTM network in this example:

- Import and preprocess the data.
- Convert the words to numeric sequences using a word encoding.
- Create and train an LSTM network with a word embedding layer.
- Classify new text data using the trained LSTM network.

Import Data

- Import the factory reports data. This data contains labeled textual descriptions of factory events. To import the text data as strings, specify the text type to be 'string'.

```
filename = "factoryReports.csv";
data = readtable(filename,'TextType','string');
head(data)
```

ans=8×5 table

Description Category Urgency Resolution Cost

"Items are occasionally getting stuck in the scanner spools." "Mechanical Failure" "Medium" "Readjust Machine" 45

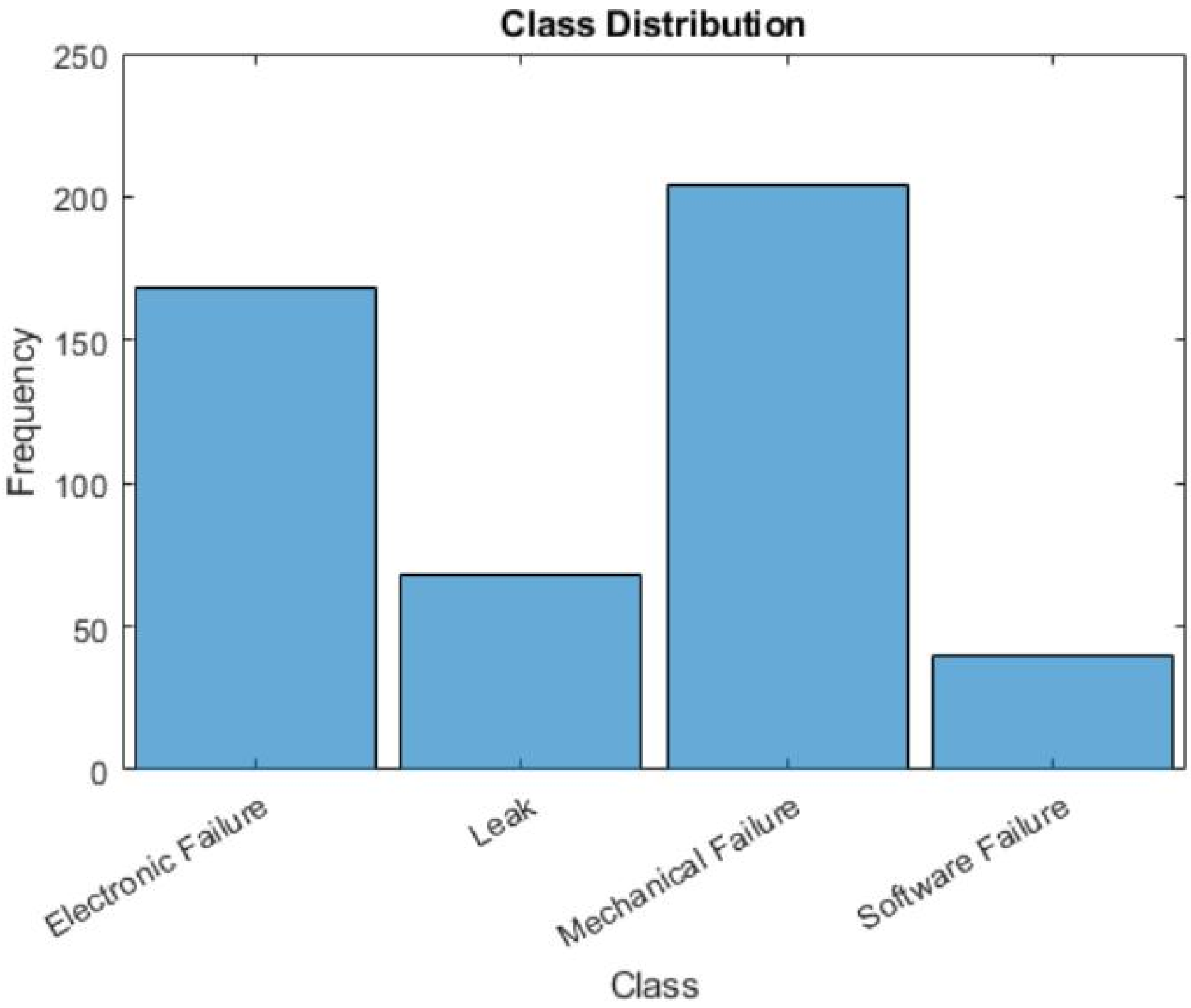
Import Data

- The goal of this example is to classify events by the label in the Category column. To divide the data into classes, convert these labels to categorical.

```
data.Category = categorical(data.Category);
```

- View the distribution of the classes in the data using a histogram.

```
figure
histogram(data.Category);
xlabel("Class")
ylabel("Frequency")
title("Class Distribution")
```



Class Distribution
250
200
150
100
50
0
Frequency
Electronic Failure
Leak
Mechanical Failure
Software Failure
Class

Import Data

- The next step is to partition it into sets for training and validation. Partition the data into a training partition and a held-out partition for validation and testing. Specify the holdout percentage to be 20%.

```
cvp = cvpartition(data.Category,'Holdout',0.2);
dataTrain = data(training(cvp),:);
dataValidation = data(test(cvp),:);
```

Import Data

- Extract the text data and labels from the partitioned tables.

```
textDataTrain = dataTrain.Description;
textDataValidation = dataValidation.Description;
YTrain = dataTrain.Category;
YValidation = dataValidation.Category;
```

Import Data

- To check that you have imported the data correctly, visualize the training text data using a word cloud.

```
figure
wordcloud(textDataTrain);
title("Training Data")
```

Preprocess Text Data

- Create a function that tokenizes and preprocesses the text data. The function preprocessText, listed at the end of the example, performs these steps:

1 Tokenize the text using tokenizedDocument.

2 Convert the text to lowercase using lower.

3 Erase the punctuation using erasePunctuation.

- Preprocess the training data and the validation data using the preprocessText function.

documentsTrain = preprocessText(textDataTrain);

documentsValidation = preprocessText(textDataValidation);

Convert Document to Sequences

- To input the documents into an LSTM network, use a word encoding to convert the documents into sequences of numeric indices. To create a word encoding, use the wordEncoding function.

```
enc = wordEncoding(documentsTrain);
```

Convert Document to Sequences

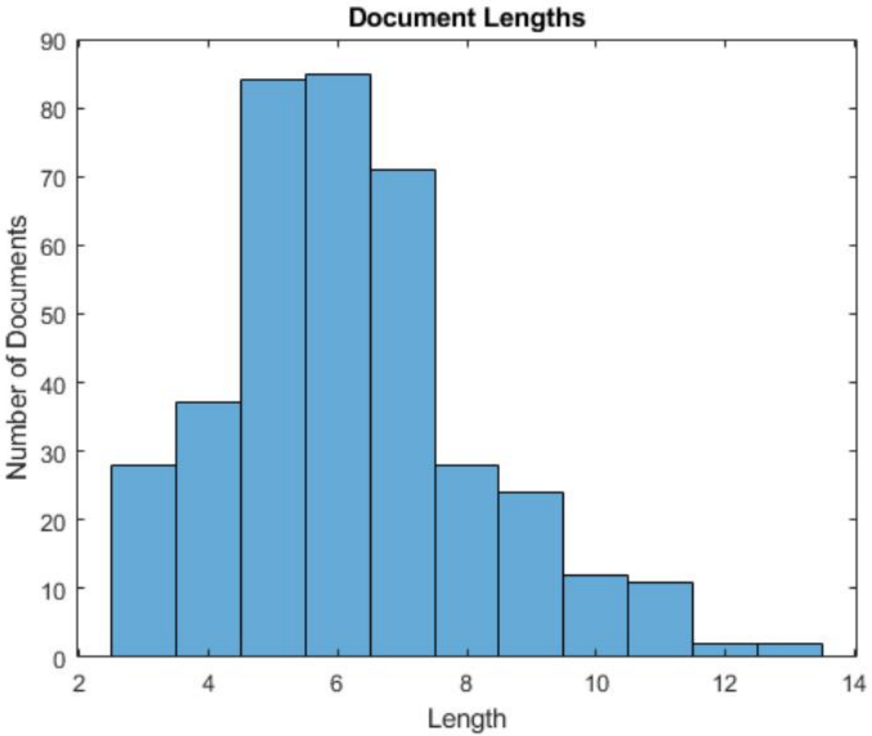
- The next conversion step is to pad and truncate documents so they are all the same length. The **trainingOptions** function provides options to pad and truncate input sequences automatically. However, these options are not well suited for sequences of word vectors. Instead, pad and truncate the sequences manually. If you left-pad and truncate the sequences of word vectors, then the training might improve.

Convert Document to Sequences

- To pad and truncate the documents, first choose a target length, and then truncate documents that are longer than it and left-pad documents that are shorter than it. For best results, the target length should be short without discarding large amounts of data. To find a suitable target length, view a histogram of the training document lengths.

Convert Document to Sequences

```
documentLengths = doclength(documentsTrain);
figure
histogram(documentLengths)
title("Document Lengths")
xlabel("Length")
ylabel("Number of Documents")
```



Convert Document to Sequences

- Most of the training documents have fewer than 10 tokens. Use this as your target length for truncation and padding. Convert the documents to sequences of numeric indices using doc2sequence. To truncate or left-pad the sequences to have length 10, set the 'Length' option to 10.

```
sequenceLength = 10;
XTrain = doc2sequence(enc,documentsTrain,'Length',sequenceLength);
```

- Convert the validation documents to sequences too.

```
XValidation = doc2sequence(enc,documentsValidation,'Length',sequenceLength);
```

Create and Train LSTM Network

- Define the LSTM network architecture. To input sequence data into the network, include a sequence input layer and set the input size to 1. Next, include a word embedding layer of dimension 50 and the same number of words as the word encoding. Next, include an LSTM layer and set the number of hidden units to 80. To use the LSTM layer for a sequence-to-label classification problem, set the output mode to 'last'. Finally, add a fully connected layer with the same size as the number of classes, a softmax layer, and a classification layer.

```
inputSize = 1;
embeddingDimension = 50;
numHiddenUnits = 80;
numWords = enc.NumWords;
numClasses = numel(categories(YTrain));
layers = [ ...
 sequenceInputLayer(inputSize)
 wordEmbeddingLayer(embeddingDimension,numWords)
 lstmLayer(numHiddenUnits,'OutputMode','last')
 fullyConnectedLayer(numClasses)
 softmaxLayer
 classificationLayer]
```

Specify Training Options

Specify the training options:

- Train using the Adam solver.
- Specify a mini-batch size of 16.
- Shuffle the data every epoch.
- Monitor the training progress by setting the 'Plots' option to'training-progress'.
- Specify the validation data using the 'ValidationData' option.
- Suppress verbose output by setting the 'Verbose' option to false.

Specify Training Options

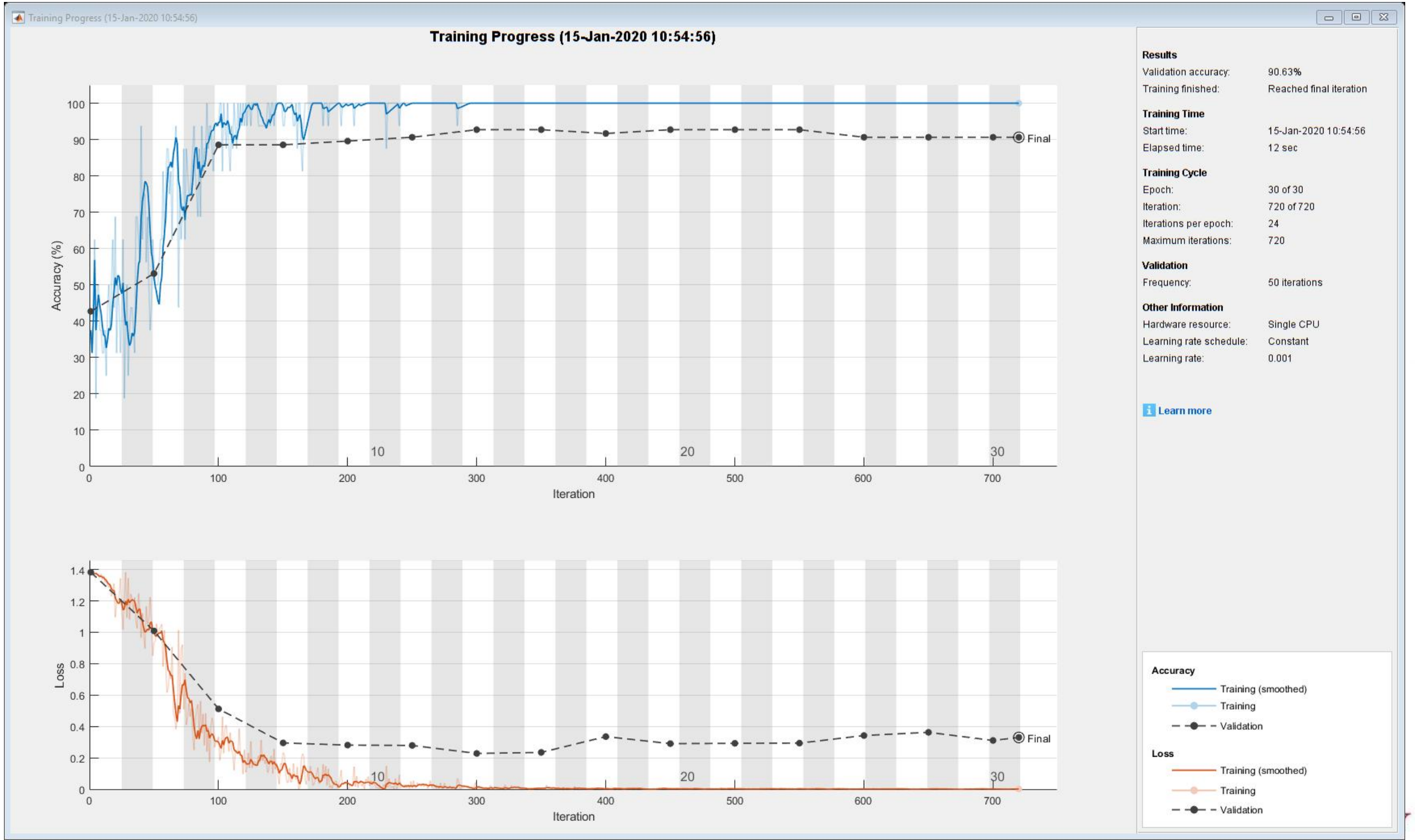
- By default, trainNetwork uses a GPU if one is available. Otherwise, it uses the CPU. To specify the execution environment manually, use the 'ExecutionEnvironment' name-value pair argument of trainingOptions. Training on a CPU can take significantly longer than training on a GPU.

Specify Training Options

```
options = trainingOptions('adam', ...
 'MiniBatchSize',16, ...
 'GradientThreshold',2, ...
 'Shuffle','every-epoch', ...
 'ValidationData',{XValidation,YValidation}, ...
 'Plots','training-progress', ...
 'Verbose',false);
```

- Train the LSTM network using the trainNetwork function.

```
net = trainNetwork(XTrain,YTrain,layers,options);
```



OF CHINA

Predict Using New Data

- Classify the event type of three new reports. Create a string array containing the new reports.

```
reportsNew = [ ...
 "Coolant is pooling underneath sorter."
 "Sorter blows fuses at start up."
 "There are some very loud rattling sounds coming from the assembler."];
```

- Preprocess the text data using the preprocessing steps as the training documents.

```
documentsNew = preprocessText(reportsNew);
```

Predict Using New Data

- Convert the text data to sequences using doc2sequence with the same options as when creating the training sequences.

```
XNew = doc2sequence(enc,documentsNew,'Length',sequenceLength);
```

- Classify the new sequences using the trained LSTM network.

```
labelsNew = classify(net,XNew)
labelsNew = 3×1 categorical
 Leak
 Electronic Failure
 Mechanical Failure
```