Want help with deep learning for time series? Take the FREE Mini-Course

Search...

How to Develop LSTM Models for Time Series Forecasting

by **Jason Brownlee** on November 14, 2018 in **Deep Learning for Time Series**

Tweet Share Share

Long Short-Term Memory networks, or LSTMs for short, can be applied to time series forecasting.

There are many types of LSTM models that can be used for each specific type of time series forecasting problem.

In this tutorial, you will discover how to develop a suite of LSTM models for a range of standard time series forecasting problems.

The objective of this tutorial is to provide standalone examples of each model on each type of time series problem as a template that you can copy and adapt for your specific time series forecasting problem.

After completing this tutorial, you will know:

- How to develop LSTM models for univariate time series forecasting.
- How to develop LSTM models for multivariate time series forecasting.
- How to develop LSTM models for multi-step time series forecasting.

This is a large and important post; you may want to bookmark it for future reference.

Discover how to build models for multivariate and multi-step time series forecasting with LSTMs and more in my new book, with 25 step-by-step tutorials and full source code.

Let’s get started.

How to Develop LSTM Models for Time Series Forecasting
Photo by N i c o l a, some rights reserved.

Tutorial Overview

In this tutorial, we will explore how to develop a suite of different types of LSTM models for time series forecasting.

The models are demonstrated on small contrived time series problems intended to give the flavor of the type of time series problem being addressed. The chosen configuration of the models is arbitrary and not optimized for each problem; that was not the goal.

This tutorial is divided into four parts; they are:

1. Univariate LSTM Models
2. Multivariate LSTM Models
3. Multi-Step LSTM Models
4. Multivariate Multi-Step LSTM Models

Univariate LSTM Models

LSTMs can be used to model univariate time series forecasting problems.

These are problems comprised of a single series of observations and a model is required to learn from the series of past observations to predict the next value in the sequence.

We will demonstrate a number of variations of the LSTM model for univariate time series forecasting.

This section is divided into six parts; they are:

1. Data Preparation
2. Vanilla LSTM
3. Stacked LSTM
4. Bidirectional LSTM
5. CNN LSTM
6. ConvLSTM

Each of these models are demonstrated for one-step univariate time series forecasting, but can easily be adapted and used as the input part of a model for other types of time series forecasting problems.

Data Preparation

Before a univariate series can be modeled, it must be prepared.

The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. As such, the sequence of observations must be transformed into multiple examples from which the LSTM can learn.

Consider a given univariate sequence:

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

We can divide the sequence into multiple input/output patterns called samples, where three time steps are used as input and one time step is used as output for the one-step prediction that is being learned.

```
X,              y
10, 20, 30      40
20, 30, 40      50
30, 40, 50      60
...
```

The *split_sequence()* function below implements this behavior and will split a given univariate sequence into multiple samples where each sample has a specified number of time steps and the output is a single time step.

```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

We can demonstrate this function on our small contrived dataset above.

The complete example is listed below.

```python
# univariate data preparation
from numpy import array

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Running the example splits the univariate series into six samples where each sample has three input time steps and one output time step.

```
[10 20 30] 40
[20 30 40] 50
[30 40 50] 60
[40 50 60] 70
[50 60 70] 80
[60 70 80] 90
```

Now that we know how to prepare a univariate series for modeling, let’s look at developing LSTM models that can learn the mapping of inputs to outputs, starting with a Vanilla LSTM.

Vanilla LSTM

A Vanilla LSTM is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.

We can define a Vanilla LSTM for univariate time series forecasting as follows.

```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

Key in the definition is the shape of the input; that is what the model expects as input for each sample in terms of the number of time steps and the number of features.

We are working with a univariate series, so the number of features is one, for one variable.

The number of time steps as input is the number we chose when preparing our dataset as an argument to the *split_sequence()* function.

The shape of the input for each sample is specified in the *input_shape* argument on the definition of first hidden layer.

We almost always have multiple samples, therefore, the model will expect the input component of training data to have the dimensions or shape:

```
[samples, timesteps, features]
```

Our *split_sequence()* function in the previous section outputs the X with the shape [*samples, timesteps*], so we easily reshape it to have an additional dimension for the one feature.

```
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

In this case, we define a model with 50 LSTM units in the hidden layer and an output layer that predicts a single numerical value.

The model is fit using the efficient Adam version of stochastic gradient descent and optimized using the mean squared error, or '*mse*' loss function.

Once the model is defined, we can fit it on the training dataset.

```
# fit model
model.fit(X, y, epochs=200, verbose=0)
```

After the model is fit, we can use it to make a prediction.

We can predict the next value in the sequence by providing the input:

```
[70, 80, 90]
```

And expecting the model to predict something like:

```
[100]
```

The model expects the input shape to be three-dimensional with [*samples, timesteps, features*], therefore, we must reshape the single input sample before making the prediction.

```
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
```

We can tie all of this together and demonstrate how to develop a Vanilla LSTM for univariate time series forecasting and make a single prediction.

```
# univariate lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example prepares the data, fits the model, and makes a prediction.

Your results may vary given the stochastic nature of the algorithm; try running the example a few times.

We can see that the model predicts the next value in the sequence.

```
[[102.09213]]
```

Stacked LSTM

Multiple hidden LSTM layers can be stacked one on top of another in what is referred to as a Stacked LSTM model.

An LSTM layer requires a three-dimensional input and LSTMs by default will produce a two-dimensional output as an interpretation from the end of the sequence.

We can address this by having the LSTM output a value for each time step in the input data by setting the *return_sequences=True* argument on the layer. This allows us to have 3D output from hidden LSTM layer as input to the next.

We can therefore define a Stacked LSTM as follows.

```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

We can tie this together; the complete code example is listed below.

```
# univariate stacked lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# split a univariate sequence
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', return_sequences=True, input_shape=(n_steps, n_features))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example predicts the next value in the sequence, which we expect would be 100.

```
[[102.47341]]
```

Bidirectional LSTM

On some sequence prediction problems, it can be beneficial to allow the LSTM model to learn the input sequence both forward and backwards and concatenate both interpretations.

This is called a Bidirectional LSTM.

We can implement a Bidirectional LSTM for univariate time series forecasting by wrapping the first hidden layer in a wrapper layer called Bidirectional.

An example of defining a Bidirectional LSTM to read input both forward and backward is as follows.

```
# define model
model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

The complete example of the Bidirectional LSTM for univariate time series forecasting is listed below.

```
# univariate bidirectional lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Bidirectional

# split a univariate sequence
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
```

```
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# define model
model = Sequential()
model.add(Bidirectional(LSTM(50, activation='relu'), input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example predicts the next value in the sequence, which we expect would be 100.

```
[[101.48093]]
```

CNN LSTM

A convolutional neural network, or CNN for short, is a type of neural network developed for working with two-dimensional image data.

The CNN can be very effective at automatically extracting and learning features from one-dimensional sequence data such as univariate time series data.

A CNN model can be used in a hybrid model with an LSTM backend where the CNN is used to interpret subsequences of input that together are provided as a sequence to an LSTM model to interpret. This hybrid model is called a CNN-LSTM.

The first step is to split the input sequences into subsequences that can be processed by the CNN model. For example, we can first split our univariate time series data into input/output samples with four steps as input and one as output. Each sample can then be split into two sub-samples, each with two time steps. The CNN can interpret each subsequence of two time steps and provide a time series of interpretations of the subsequences to the LSTM model to process as input.

We can parameterize this and define the number of subsequences as *n_seq* and the number of time steps per subsequence as *n_steps*. The input data can then be reshaped to have the required structure:

```
[samples, subsequences, timesteps, features]
```

For example:

```
# choose a number of time steps
n_steps = 4
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
n_features = 1
n_seq = 2
```

```
n_steps = 2
X = X.reshape((X.shape[0], n_seq, n_steps, n_features))
```

We want to reuse the same CNN model when reading in each sub-sequence of data separately.

This can be achieved by wrapping the entire CNN model in a TimeDistributed wrapper that will apply the entire model once per input, in this case, once per input subsequence.

The CNN model first has a convolutional layer for reading across the subsequence that requires a number of filters and a kernel size to be specified. The number of filters is the number of reads or interpretations of the input sequence. The kernel size is the number of time steps included of each 'read' operation of the input sequence.

The convolution layer is followed by a max pooling layer that distills the filter maps down to 1/2 of their size that includes the most salient features. These structures are then flattened down to a single one-dimensional vector to be used as a single input time step to the LSTM layer.

```
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(Nor
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
```

Next, we can define the LSTM part of the model that interprets the CNN model's read of the input sequence and makes a prediction.

```
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
```

We can tie all of this together; the complete example of a CNN-LSTM model for univariate time series forecasting is listed below.

```
# univariate cnn lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import TimeDistributed
from keras.layers.convolutional import Conv1D
from keras.layers.convolutional import MaxPooling1D

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

```
# choose a number of time steps
n_steps = 4
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, subsequences, timesteps, features]
n_features = 1
n_seq = 2
n_steps = 2
X = X.reshape((X.shape[0], n_seq, n_steps, n_features))
# define model
model = Sequential()
model.add(TimeDistributed(Conv1D(filters=64, kernel_size=1, activation='relu'), input_shape=(No
model.add(TimeDistributed(MaxPooling1D(pool_size=2)))
model.add(TimeDistributed(Flatten()))
model.add(LSTM(50, activation='relu'))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=500, verbose=0)
# demonstrate prediction
x_input = array([60, 70, 80, 90])
x_input = x_input.reshape((1, n_seq, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example predicts the next value in the sequence, which we expect would be 100.

```
[[101.69263]]
```

ConvLSTM

A type of LSTM related to the CNN-LSTM is the ConvLSTM, where the convolutional reading of input is built directly into each LSTM unit.

The ConvLSTM was developed for reading two-dimensional spatial-temporal data, but can be adapted for use with univariate time series forecasting.

The layer expects input as a sequence of two-dimensional images, therefore the shape of input data must be:

```
[samples, timesteps, rows, columns, features]
```

For our purposes, we can split each sample into subsequences where timesteps will become the number of subsequences, or *n_seq*, and columns will be the number of time steps for each subsequence, or *n_steps*. The number of rows is fixed at 1 as we are working with one-dimensional data.

We can now reshape the prepared samples into the required structure.

```
# choose a number of time steps
n_steps = 4
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, rows, columns, features]
n_features = 1
n_seq = 2
n_steps = 2
X = X.reshape((X.shape[0], n_seq, 1, n_steps, n_features))
```

We can define the ConvLSTM as a single layer in terms of the number of filters and a two-dimensional kernel size in terms of (rows, columns). As we are working with a one-dimensional series, the number of rows is always fixed to 1 in the kernel.

The output of the model must then be flattened before it can be interpreted and a prediction made.

```
model.add(ConvLSTM2D(filters=64, kernel_size=(1,2), activation='relu', input_shape=(n_seq, 1, n_
model.add(Flatten())
```

The complete example of a ConvLSTM for one-step univariate time series forecasting is listed below.

```
# univariate convlstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import ConvLSTM2D

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 4
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, rows, columns, features]
n_features = 1
n_seq = 2
n_steps = 2
X = X.reshape((X.shape[0], n_seq, 1, n_steps, n_features))
# define model
model = Sequential()
model.add(ConvLSTM2D(filters=64, kernel_size=(1,2), activation='relu', input_shape=(n_seq, 1, r
model.add(Flatten())
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=500, verbose=0)
# demonstrate prediction
x_input = array([60, 70, 80, 90])
x_input = x_input.reshape((1, n_seq, 1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example predicts the next value in the sequence, which we expect would be 100.

```
[[103.68166]]
```

Now that we have looked at LSTM models for univariate data, let’s turn our attention to multivariate data.

Multivariate LSTM Models

Multivariate time series data means data where there is more than one observation for each time step.

There are two main models that we may require with multivariate time series data; they are:

1. Multiple Input Series.
2. Multiple Parallel Series.

Let’s take a look at each in turn.

Multiple Input Series

A problem may have two or more parallel input time series and an output time series that is dependent on the input time series.

The input time series are parallel because each series has an observation at the same time steps.

We can demonstrate this with a simple example of two parallel input time series where the output series is the simple addition of the input series.

```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
```

We can reshape these three arrays of data as a single dataset where each row is a time step, and each column is a separate time series. This is a standard way of storing parallel time series in a CSV file.

```
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
```

The complete example is listed below.

```
# multivariate data preparation
from numpy import array
from numpy import hstack
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
```

```
print(dataset)
```

Running the example prints the dataset with one row per time step and one column for each of the two input and one output parallel time series.

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```

As with the univariate time series, we must structure these data into samples with input and output elements.

An LSTM model needs sufficient context to learn a mapping from an input sequence to an output value. LSTMs can support parallel input time series as separate variables or features. Therefore, we need to split the data into samples maintaining the order of observations across the two input sequences.

If we chose three input time steps, then the first sample would look as follows:

Input:

```
10, 15
20, 25
30, 35
```

Output:

```
65
```

That is, the first three time steps of each parallel series are provided as input to the model and the model associates this with the value in the output series at the third time step, in this case, 65.

We can see that, in transforming the time series into input/output samples to train the model, that we will have to discard some values from the output time series where we do not have values in the input time series at prior time steps. In turn, the choice of the size of the number of input time steps will have an important effect on how much of the training data is used.

We can define a function named *split_sequences()* that will take a dataset as we have defined it with rows for time steps and columns for parallel series and return input/output samples.

```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

We can test this function on our dataset using three time steps for each input time series as input.

The complete example is listed below.

```
# multivariate data preparation
from numpy import array
from numpy import hstack

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
print(X.shape, y.shape)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Running the example first prints the shape of the X and y components.

We can see that the X component has a three-dimensional structure.

The first dimension is the number of samples, in this case 7. The second dimension is the number of time steps per sample, in this case 3, the value specified to the function. Finally, the last dimension specifies the number of parallel time series or the number of variables, in this case 2 for the two parallel series.

This is the exact three-dimensional structure expected by an LSTM as input. The data is ready to use without further reshaping.

We can then see that the input and output for each sample is printed, showing the three time steps for each of the two input series and the associated output for each sample.

```
(7, 3, 2) (7,)

[[10 15]
 [20 25]
 [30 35]] 65
[[20 25]
 [30 35]
 [40 45]] 85
[[30 35]
 [40 45]
 [50 55]] 105
[[40 45]
 [50 55]
 [60 65]] 125
[[50 55]
 [60 65]
 [70 75]] 145
[[60 65]
 [70 75]
 [80 85]] 165
[[70 75]
 [80 85]
 [90 95]] 185
```

We are now ready to fit an LSTM model on this data.

Any of the varieties of LSTMs in the previous section can be used, such as a Vanilla, Stacked, Bidirectional, CNN, or ConvLSTM model.

We will use a Vanilla LSTM where the number of time steps and parallel series (features) are specified for the input layer via the *input_shape* argument.

```
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

When making a prediction, the model expects three time steps for two input time series.

We can predict the next value in the output series providing the input values of:

```
80,  85
90,  95
100, 105
```

The shape of the one sample with three time steps and two variables must be [1, 3, 2].

We would expect the next value in the sequence to be 100 + 105, or 205.

```
# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
```

The complete example is listed below.

```
# multivariate lstm example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example prepares the data, fits the model, and makes a prediction.

```
[[208.13531]]
```

Multiple Parallel Series

An alternate time series problem is the case where there are multiple parallel time series and a value must be predicted for each.

For example, given the data from the previous section:

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```

We may want to predict the value for each of the three time series for the next time step.

This might be referred to as multivariate forecasting.

Again, the data must be split into input/output samples in order to train a model.

The first sample of this dataset would be:

Input:

```
10, 15, 25
20, 25, 45
30, 35, 65
```

Output:

```
40, 45, 85
```

The *split_sequences()* function below will split multiple parallel time series with rows for time steps and one series per column into the required input/output shape.

```python
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

We can demonstrate this on the contrived problem; the complete example is listed below.

```python
# multivariate output data prep
from numpy import array
from numpy import hstack

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
print(X.shape, y.shape)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Running the example first prints the shape of the prepared X and y components.

The shape of X is three-dimensional, including the number of samples (6), the number of time steps chosen per sample (3), and the number of parallel time series or features (3).

The shape of y is two-dimensional as we might expect for the number of samples (6) and the number of time variables per sample to be predicted (3).

The data is ready to use in an LSTM model that expects three-dimensional input and two-dimensional output shapes for the X and y components of each sample.

Then, each of the samples is printed showing the input and output components of each sample.

```
(6, 3, 3) (6, 3)

[[10 15 25]
 [20 25 45]
 [30 35 65]] [40 45 85]
[[20 25 45]
 [30 35 65]
 [40 45 85]] [ 50  55 105]
[[ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]] [ 60  65 125]
[[ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]] [ 70  75 145]
[[ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]] [ 80  85 165]
[[ 60  65 125]
```

```
 [ 70  75 145]
 [ 80  85 165]] [ 90  95 185]
```

We are now ready to fit an LSTM model on this data.

Any of the varieties of LSTMs in the previous section can be used, such as a Vanilla, Stacked, Bidirectional, CNN, or ConvLSTM model.

We will use a Stacked LSTM where the number of time steps and parallel series (features) are specified for the input layer via the *input_shape* argument. The number of parallel series is also used in the specification of the number of values to predict by the model in the output layer; again, this is three.

```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(n_steps, n_features))
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_features))
model.compile(optimizer='adam', loss='mse')
```

We can predict the next value in each of the three parallel series by providing an input of three time steps for each series.

```
70, 75, 145
80, 85, 165
90, 95, 185
```

The shape of the input for making a single prediction must be 1 sample, 3 time steps, and 3 features, or [1, 3, 3]

```
# demonstrate prediction
x_input = array([[70,75,145], [80,85,165], [90,95,185]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
```

We would expect the vector output to be:

```
[100, 105, 205]
```

We can tie all of this together and demonstrate a Stacked LSTM for multivariate output time series forecasting below.

```
# multivariate output stacked lstm example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(n_steps, n_features)
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_features))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=400, verbose=0)
# demonstrate prediction
x_input = array([[70,75,145], [80,85,165], [90,95,185]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example prepares the data, fits the model, and makes a prediction.

```
[[101.76599 108.730484 206.63577 ]]
```

Multi-Step LSTM Models

A time series forecasting problem that requires a prediction of multiple time steps into the future can be referred to as multi-step time series forecasting.

Specifically, these are problems where the forecast horizon or interval is more than one time step.

There are two main types of LSTM models that can be used for multi-step forecasting; they are:

1. Vector Output Model
2. Encoder-Decoder Model

Before we look at these models, let's first look at the preparation of data for multi-step forecasting.

Data Preparation

As with one-step forecasting, a time series used for multi-step time series forecasting must be split into samples with input and output components.

Both the input and output components will be comprised of multiple time steps and may or may not have the same number of steps.

For example, given the univariate time series:

```
[10, 20, 30, 40, 50, 60, 70, 80, 90]
```

We could use the last three time steps as input and forecast the next two time steps.

The first sample would look as follows:

Input:

```
[10, 20, 30]
```

Output:

```
[40, 50]
```

The *split_sequence()* function below implements this behavior and will split a given univariate time series into samples with a specified number of input and output time steps.

```
# split a univariate sequence into samples
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

We can demonstrate this function on the small contrived dataset.

The complete example is listed below.

```
# multi-step data preparation
from numpy import array

# split a univariate sequence into samples
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# split into samples
X, y = split_sequence(raw_seq, n_steps_in, n_steps_out)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Running the example splits the univariate series into input and output time steps and prints the input and output components of each.

```
[10 20 30] [40 50]
[20 30 40] [50 60]
[30 40 50] [60 70]
[40 50 60] [70 80]
[50 60 70] [80 90]
```

Now that we know how to prepare data for multi-step forecasting, let's look at some LSTM models that can learn this mapping.

Vector Output Model

Like other types of neural network models, the LSTM can output a vector directly that can be interpreted as a multi-step forecast.

This approach was seen in the previous section were one time step of each output time series was forecasted as a vector.

As with the LSTMs for univariate data in a prior section, the prepared samples must first be reshaped. The LSTM expects data to have a three-dimensional structure of [*samples, timesteps, features*], and in this case, we only have one feature so the reshape is straightforward.

```
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

With the number of input and output steps specified in the *n_steps_in* and *n_steps_out* variables, we can define a multi-step time-series forecasting model.

Any of the presented LSTM model types could be used, such as Vanilla, Stacked, Bidirectional, CNN-LSTM, or ConvLSTM. Below defines a Stacked LSTM for multi-step forecasting.

```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(n_steps_in, n_feature
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse')
```

The model can make a prediction for a single sample. We can predict the next two steps beyond the end of the dataset by providing the input:

```
[70, 80, 90]
```

We would expect the predicted output to be:

```
[100, 110]
```

As expected by the model, the shape of the single sample of input data when making the prediction must be [1, 3, 1] for the 1 sample, 3 time steps of the input, and the single feature.

```
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
```

Tying all of this together, the Stacked LSTM for multi-step forecasting with a univariate time series is listed below.

```
# univariate multi-step vector-output stacked lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# split a univariate sequence into samples
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# split into samples
X, y = split_sequence(raw_seq, n_steps_in, n_steps_out)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(n_steps_in, n_featur
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=50, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example forecasts and prints the next two time steps in the sequence.

```
[[100.98096 113.28924]]
```

Encoder-Decoder Model

A model specifically developed for forecasting variable length output sequences is called the Encoder-Decoder LSTM.

The model was designed for prediction problems where there are both input and output sequences, so-called sequence-to-sequence, or seq2seq problems, such as translating text from one language to another.

This model can be used for multi-step time series forecasting.

As its name suggests, the model is comprised of two sub-models: the encoder and the decoder.

The encoder is a model responsible for reading and interpreting the input sequence. The output of the encoder is a fixed length vector that represents the model's interpretation of the sequence. The encoder is traditionally a Vanilla LSTM model, although other encoder models can be used such as Stacked, Bidirectional, and CNN models.

```
model.add(LSTM(100, activation='relu', input_shape=(n_steps_in, n_features)))
```

The decoder uses the output of the encoder as an input.

First, the fixed-length output of the encoder is repeated, once for each required time step in the output sequence.

```
model.add(RepeatVector(n_steps_out))
```

This sequence is then provided to an LSTM decoder model. The model must output a value for each value in the output time step, which can be interpreted by a single output model.

```
model.add(LSTM(100, activation='relu', return_sequences=True))
```

We can use the same output layer or layers to make each one-step prediction in the output sequence. This can be achieved by wrapping the output part of the model in a TimeDistributed wrapper.

```
model.add(TimeDistributed(Dense(1)))
```

The full definition for an Encoder-Decoder model for multi-step time series forecasting is listed below.

```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(RepeatVector(n_steps_out))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
```

As with other LSTM models, the input data must be reshaped into the expected three-dimensional shape of [*samples, timesteps, features*].

```
X = X.reshape((X.shape[0], X.shape[1], n_features))
```

In the case of the Encoder-Decoder model, the output, or y part, of the training dataset must also have this shape. This is because the model will predict a given number of time steps with a given number of features for each input sample.

```
y = y.reshape((y.shape[0], y.shape[1], n_features))
```

The complete example of an Encoder-Decoder LSTM for multi-step time series forecasting is listed below.

```
# univariate multi-step encoder-decoder lstm example
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed

# split a univariate sequence into samples
def split_sequence(sequence, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the sequence
        if out_end_ix > len(sequence):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix:out_end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# split into samples
X, y = split_sequence(raw_seq, n_steps_in, n_steps_out)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
y = y.reshape((y.shape[0], y.shape[1], n_features))
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(RepeatVector(n_steps_out))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=100, verbose=0)
# demonstrate prediction
x_input = array([70, 80, 90])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example forecasts and prints the next two time steps in the sequence.

```
[[[101.9736
  [116.213615]]]
```

Multivariate Multi-Step LSTM Models

In the previous sections, we have looked at univariate, multivariate, and multi-step time series forecasting.

It is possible to mix and match the different types of LSTM models presented so far for the different problems. This too applies to time series forecasting problems that involve multivariate and multi-step forecasting, but it may be a little more challenging.

In this section, we will provide short examples of data preparation and modeling for multivariate multi-step time series forecasting as a template to ease this challenge, specifically:

1. Multiple Input Multi-Step Output.
2. Multiple Parallel Input and Multi-Step Output.

Perhaps the biggest stumbling block is in the preparation of data, so this is where we will focus our attention.

Multiple Input Multi-Step Output

There are those multivariate time series forecasting problems where the output series is separate but dependent upon the input time series, and multiple time steps are required for the output series.

For example, consider our multivariate time series from a prior section:

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```

We may use three prior time steps of each of the two input time series to predict two time steps of the output time series.

Input:

```
10, 15
20, 25
30, 35
```

Output:

```
65
85
```

The *split_sequences()* function below implements this behavior.

```python
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out-1
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1:out_end_ix, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

We can demonstrate this on our contrived dataset.

The complete example is listed below.

```python
# multivariate multi-step data preparation
from numpy import array
from numpy import hstack

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out-1
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1:out_end_ix, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# covert into input/output
X, y = split_sequences(dataset, n_steps_in, n_steps_out)
print(X.shape, y.shape)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Running the example first prints the shape of the prepared training data.

We can see that the shape of the input portion of the samples is three-dimensional, comprised of six samples, with three time steps, and two variables for the 2 input time series.

The output portion of the samples is two-dimensional for the six samples and the two time steps for each sample to be predicted.

The prepared samples are then printed to confirm that the data was prepared as we specified.

```
(6, 3, 2) (6, 2)

[[10 15]
 [20 25]
 [30 35]] [65 85]
[[20 25]
 [30 35]
 [40 45]] [ 85 105]
[[30 35]
 [40 45]
 [50 55]] [105 125]
[[40 45]
 [50 55]
 [60 65]] [125 145]
[[50 55]
 [60 65]
 [70 75]] [145 165]
[[60 65]
 [70 75]
 [80 85]] [165 185]
```

We can now develop an LSTM model for multi-step predictions.

A vector output or an encoder-decoder model could be used. In this case, we will demonstrate a vector output with a Stacked LSTM.

The complete example is listed below.

```
# multivariate multi-step stacked lstm example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out-1
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1:out_end_ix, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

```

```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# covert into input/output
X, y = split_sequences(dataset, n_steps_in, n_steps_out)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True, input_shape=(n_steps_in, n_featur
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
x_input = array([[70, 75], [80, 85], [90, 95]])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example fits the model and predicts the next two time steps of the output sequence beyond the dataset.

We would expect the next two steps to be: [185, 205]

It is a challenging framing of the problem with very little data, and the arbitrarily configured version of the model gets close.

```
[[188.70619 210.16513]]
```

Multiple Parallel Input and Multi-Step Output

A problem with parallel time series may require the prediction of multiple time steps of each time series.

For example, consider our multivariate time series from a prior section:

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```

We may use the last three time steps from each of the three time series as input to the model and predict the next time steps of each of the three time series as output.

The first sample in the training dataset would be the following.

Input:

```
10, 15, 25
20, 25, 45
30, 35, 65
```

Output:

```
40, 45, 85
50, 55, 105
```

The *split_sequences()* function below implements this behavior.

```python
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)
```

We can demonstrate this function on the small contrived dataset.

The complete example is listed below.

```python
# multivariate multi-step data preparation
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
```

```
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# covert into input/output
X, y = split_sequences(dataset, n_steps_in, n_steps_out)
print(X.shape, y.shape)
# summarize the data
for i in range(len(X)):
    print(X[i], y[i])
```

Running the example first prints the shape of the prepared training dataset.

We can see that both the input (X) and output (Y) elements of the dataset are three dimensional for the number of samples, time steps, and variables or parallel time series respectively.

The input and output elements of each series are then printed side by side so that we can confirm that the data was prepared as we expected.

```
(5, 3, 3) (5, 2, 3)

[[10 15 25]
 [20 25 45]
 [30 35 65]] [[ 40  45  85]
 [ 50  55 105]]
[[20 25 45]
 [30 35 65]
 [40 45 85]] [[ 50  55 105]
 [ 60  65 125]]
[[ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]] [[ 60  65 125]
 [ 70  75 145]]
[[ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]] [[ 70  75 145]
 [ 80  85 165]]
[[ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]] [[ 80  85 165]
 [ 90  95 185]]
```

We can use either the Vector Output or Encoder-Decoder LSTM to model this problem. In this case, we will use the Encoder-Decoder model.

The complete example is listed below.

```
# multivariate multi-step encoder-decoder lstm example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
```

```python
from keras.layers import RepeatVector
from keras.layers import TimeDistributed

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps_in, n_steps_out):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps_in
        out_end_ix = end_ix + n_steps_out
        # check if we are beyond the dataset
        if out_end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix:out_end_ix, :]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps_in, n_steps_out = 3, 2
# covert into input/output
X, y = split_sequences(dataset, n_steps_in, n_steps_out)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_steps_in, n_features)))
model.add(RepeatVector(n_steps_out))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(n_features)))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=300, verbose=0)
# demonstrate prediction
x_input = array([[60, 65, 125], [70, 75, 145], [80, 85, 165]])
x_input = x_input.reshape((1, n_steps_in, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example fits the model and predicts the values for each of the three time steps for the next two time steps beyond the end of the dataset.

We would expect the values for these series and time steps to be as follows:

```
90, 95, 185
100, 105, 205
```

We can see that the model forecast gets reasonably close to the expected values.

```
[[[ 91.86044   97.77231  189.66768 ]
  [103.299355 109.18123  212.6863  ]]]
```

Summary

In this tutorial, you discovered how to develop a suite of LSTM models for a range of standard time series forecasting problems.

Specifically, you learned:

- How to develop LSTM models for univariate time series forecasting.
- How to develop LSTM models for multivariate time series forecasting.
- How to develop LSTM models for multi-step time series forecasting.

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

About Jason Brownlee

Jason Brownlee, PhD is a machine learning specialist who teaches developers how to get results with modern machine learning methods via hands-on tutorials.
View all posts by Jason Brownlee →

< How to Develop Convolutional Neural Network Models for Time Series Forecasting

How to Grid Search Deep Learning Models for Time Series Forecasting >

263 Responses to *How to Develop LSTM Models for Time Series Forecasting*

Jenna Ma November 16, 2018 at 12:09 am # REPLY

This tutorial is so helpful to me. Thank you very much!
It will be more helpful in the real projects if the dataset is split into batches. Hope you will mention this in the future.

Jason Brownlee November 16, 2018 at 6:16 am # REPLY

Keras will split the dataset into batches.

Jenna Ma November 16, 2018 at 7:27 pm # REPLY

I think this blog (https://machinelearningmastery.com/use-different-batch-sizes-training-predicting-python-keras/) may answer my question. I will do more research. Thanks a lot.

Jason Brownlee November 17, 2018 at 5:45 am # REPLY

Great!

Amy November 16, 2018 at 7:17 am # REPLY

Thanks Jason for this good tutorial. I have a question. When we have two different time series, 1 and 2. Time series 1 will influence time series 2 and our goal is to predict the future value of time series 2. How can we use LSTM for this case?

Jason Brownlee November 16, 2018 at 1:55 pm # REPLY

I call this a dependent time series problem. I given an example of how to model it on this post:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

Amy November 16, 2018 at 2:21 pm #

REPLY

The link is the link of the current page, Do you mean that?

Jason Brownlee November 17, 2018 at 5:41 am #

REPLY

Yes, I give an example above.

Kwan November 22, 2018 at 8:03 pm #

REPLY

Thanks Jason for this good tutorial, I have read your tutorial for a long time , I have a question. How to use LSTM model forecasting Multi-Site Multivariate Time Series, such as EMC Data Science Global Hackathon dataset, thank you very much!

Jason Brownlee November 23, 2018 at 7:48 am #

REPLY

I have advice for multi-site forecasting here:
https://machinelearningmastery.com/faq/single-faq/how-to-develop-forecast-models-for-multiple-sites

Caiyuan November 29, 2018 at 1:33 pm #

REPLY

Thank you for sharing. I found that the results of time series prediction using LSTM are similar to the results of one step behind the original sequence. What do you think?

Jason Brownlee November 29, 2018 at 2:40 pm #

REPLY

Sounds like the model has learned a persistance model and may not be skillful.

Sudrit Saisa-ing August 9, 2019 at 8:48 pm #

REPLY

I have some question?
If I have model from LSTM,I want to know percent of accurate of new prediction.

How to know percent accurate for new forcast?

Thank you

Jason Brownlee August 10, 2019 at 7:16 am # REPLY

If your model is predicting a class label, you can specify the accuracy measure as a metric in the call to compile() then use the evaluate() model to calculate the accuracy.

You can learn how for an MLP in this post which will be the same for an LSTM:
http://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

WLF December 5, 2018 at 6:04 pm # REPLY

Thanks a lot! I have read your websites for a long time!
I have a question, in "Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras" you said that:
"LSTMs are sensitive to the scale of the input data, specifically when the sigmoid (default) or tanh activation functions are used. It can be a good practice to rescale the data to the range of 0-to-1, also called normalizing. "
So why don't you normalize input here?
Because you used relu? Because the data is increasing (so we can't normalize the future input)? Or because you just give us an example?
Do you suggest normalizing here?

Jason Brownlee December 6, 2018 at 5:51 am # REPLY

It would be a good idea to prepare the data with normalization or similar here.

I chose not to because it seems to confuse more readers than it helps. Also, choice of relu does make the model a lot more robust to unscaled data.

rkk621 December 6, 2018 at 2:27 am # REPLY

Thanks for a great article. Minor typo or confusion:

For the Multiple input case in Multivariate series, if we use three time steps and

10,15
20,25
30,35

as our inputs, shouldn't the output (predicted val used for training) be

85

instead of 65?

Jason Brownlee December 6, 2018 at 5:57 am #

REPLY

In the chosen framing of the problem, we want to predict the output at t not t+1, given inputs up to and including t.

You can choose to frame the problem differently if you like. It is arbitrary.

WLF December 6, 2018 at 11:02 pm #

REPLY

You can also reference 'Multiple Parallel …'

So you can find the differences in function 'split_sequences'

if you want to predict 85, you can change the code to:

if end_ix > len(sequences)-1:
break
seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix, -1]

Notice 'len(sequences)-1', and 'sequences[end_ix, -1]'

Ida December 10, 2018 at 7:55 pm #

REPLY

Thanks sooooo much Jason.
It helped me a lot.

Jason Brownlee December 11, 2018 at 7:42 am #

REPLY

I'm happy to hear that.

John December 12, 2018 at 9:21 am #

REPLY

Hi Jason,

Thanks for this nice blog! I am new to LSTM in time-series, and I need your help.

Most info on internet is for a single time series and for next-step forecasting. I want to produce 6 months ahead forecast using previous 15 months for 100 different time series, each of length 54 months.

So, there is 34 windows for each time-series if we use sliding windows. So, my initial X_train has a shape of (3400,15). Then. I am reshaping my X_train [samples, timesteps, features] as follows: (3400, 15, 1). Is this reshaping correct? In genera, how can we choose “timesteps” and “features” arguments in this multi-input multi-step forecast?

Also, how can I choose “batch_size” and “units”? Since I want 6 months ahead forecast, my output should be a matrix with dimensions (100,6). I chose units=6, and batch_size=1. Are these numbers correct?

Thanks for your help!

Jason Brownlee December 12, 2018 at 2:14 pm # REPLY

Looks good.

Time steps is really problem specific – e.g. how much history do you need to make a prediction. Perhaps test with your data.

Batch size and units – again, depends on your problem. Test. 6 units is too few. Start with 100, try 500, 1000, etc. Batch size of 1 seems small, perhaps also try 32, 64, etc.

Let me know how you go.

John December 13, 2018 at 2:06 am # REPLY

Hi Jason,

Thanks for your response.

I don’t understand “6 units is too few”. In documentation of lstm functions in R, units is defined as “dimensionality of the output space”. Since I need an output with 6 columns (6 months forecast), I define units=6. Any other number does not produce the output I want. Is there anything wrong in my interpretation?

Jason Brownlee December 13, 2018 at 7:55 am # REPLY

I recommend using a Dense layer as the output rather than the outputting from the LSTM directly.

Then dramatically increase the capacity of the model by increasing the number of LSTM units.

Shaifali December 16, 2018 at 1:21 am # REPLY

Bidirectional LSTM works better than LSTM. Can you please explain the working of bidirectional LSTM. Since we do not know future values. How do we do prediction?

Jason Brownlee December 16, 2018 at 5:23 am #

REPLY

It has two LSTM layers, one that processes the sequences forwards, and one that processes it backwards.

You can learn more here:
https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

Jenna Ma December 16, 2018 at 4:24 pm #

REPLY

In the last encoder-decoder model, if I have different features of input and output, is it correct that I change the code like this?
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(n_steps_in, n_features_in)))
model.add(RepeatVector(n_steps_out))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(n_features_out)))
model.compile(optimizer='adam', loss='mse')

Jason Brownlee December 17, 2018 at 6:19 am #

REPLY

I'm sure I understand, what do you mean exactly?

Jenna Ma December 18, 2018 at 1:50 pm #

REPLY

I am sorry for not expressing my question clearly.
In the last part of your tutorial, you gave an example like this:
[[10 15 25]
[20 25 45]
[30 35 65]]
[[40 45 85]
[50 55 105]]
Then, you introduced the Encoder-Decoder LSTM to model this problem.
If I want to use the last three time steps from each of the three time series as input to the model and predict the next two time steps of the third time series as output. Namely, my input and output elements are like the following. The shapes of input and output are (5, 3, 3) and (5, 2, 1) respectively.
[[10 15 25]
[20 25 45]

[30 35 65]]
[[85]
[105]]
When I define the Encoder-Decoder LSTM model, the code will be like this:
model = Sequential()
model.add(LSTM(200, activation='relu', input_shape=(3,3)))
model.add(RepeatVector(2))
model.add(LSTM(200, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
Is it correct?
Thank you very much!

Jason Brownlee December 18, 2018 at 2:36 pm # REPLY

It looks correct, but I don't have the capacity to test the code to be sure.

Jenna Ma December 18, 2018 at 6:05 pm #

Thank you!
I test the code, and I want to show you what I got.
I assume the input sequence:
in_seq1 = np.arange(10,1000,10)
in_seq2 = np.arange(15,1005,10)
Define the prediction input:
x_input = np.array([[960, 965, 1925], [970, 975, 1945], [980, 985, 1965]])
I expect the output values would be as follows:
[[1985] [2005]]
And the model forecasts: [[1997.1425] [2026.6136]]
I think this means that the model can work.

Jason Brownlee December 19, 2018 at 6:31 am #

Nice work! Now you can start tuning the model to lift skill.

dani December 19, 2018 at 12:55 pm # REPLY

how we can test these examples if have big excel data set?and its time series data, kindly refer to a link?

Jason Brownlee December 19, 2018 at 2:30 pm #

REPLY

Save as a CSV file then use code in this post to prepare it for modeling:
https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

mk December 20, 2018 at 7:13 pm #

REPLY

Can Multivariate time series apply to cnn-lstm model?

Jason Brownlee December 21, 2018 at 5:27 am #

REPLY

Yes, I have a good beginner example here:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

Lionel December 21, 2018 at 6:16 pm #

REPLY

I want to predict visibility on one airport for the next 120 hours.
I already build a LSTM to predict the visibility for the next hour, solely based on visibility observation.
(Basically, the network learned that persistance is a good algorithm.)

My next step is to include a weather model forecast of say humidity as input.

I have then as input:
visibility observation on the airport (past and present)
prediction of humidity for the next 120 hours.

I have trouble to combine these two information.
Do you have suggestions?

Jason Brownlee December 22, 2018 at 6:03 am #

REPLY

What trouble are you having exactly?

Lionel December 22, 2018 at 7:21 pm #

REPLY

let's say:
Input : last 120 h of measured visibility

weather forcast for the next 120 h

Output: visibility prediction for the next 120 h

Implementation:
make visibility prediction every hour for the next 120 h

I have trouble to see how the LSTM will update its state every hour, since it will only get as new information a measured visibility for the last hour, and not about the full 120 h prediction.

I must say that I'm a newbie in ML.

Jason Brownlee December 23, 2018 at 6:04 am # REPLY

The model is only aware of the data that you provide it.

Potofski December 22, 2018 at 3:48 am # REPLY

Thanks a lot for your post. Your work is a great resource on forecasts with lstm!

Assume, I have dependent time series (heating costs and temperature) and I want to predict the dependent (heating costs), how could I implement temperature predictions (from other weather forecasts) into my model for heating cost predictions?

Do you know of any common approaches to this? Or any papers on how to handle external forecasts for independent variables?

Jason Brownlee December 22, 2018 at 6:07 am # REPLY

I recommend this process generally:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

Jenna Ma January 4, 2019 at 9:35 pm # REPLY

Hi Jason,
I think I saw you mentioning the activation function 'relu' usually works better than 'tanh' in LSTM model.
But, I forget I saw this in which post. I don't find any post from your blog that focuses on how to choose the activation function. So, I submit this question under this post and hope you don't mind.
Is it true that 'relu' often works better than 'tanh' in your experience? If you have any post talking about activation function, please give me the title or URL.
Thank you very much!

Jason Brownlee January 5, 2019 at 6:55 am # REPLY

It really depends on the dataset, I have found LSTMs with relu more robust on some problems.

Jenna Ma January 7, 2019 at 12:50 am # REPLY

Thank you! So, the way I can make sure which activation function is the best for my dataset is to enumerate and see the results?

Jason Brownlee January 7, 2019 at 6:37 am # REPLY

Yes. It will almost certainly be relu or tanh.

Matt January 9, 2019 at 3:21 am # REPLY

This is awesome for someone starting out with LSTM.

All the content on your site is amazing, I really appreciate it. Thank you.

Jason Brownlee January 9, 2019 at 8:47 am # REPLY

Thanks!

Andrew Jabbitt January 10, 2019 at 4:23 am # REPLY

Hi Jason,

Still lovin' your work!

1 question: can you please explain the purpose of the out_seq series in the Multiple Parallel Series example?

Many thanks,
Andrew

Jason Brownlee January 10, 2019 at 7:57 am # REPLY

It is the output sequence, dependent upon the input sequences.

sophia January 22, 2019 at 8:29 am #

REPLY

another great article, Jason! I'm trying to get started on a project that is similar to the LSTM model described in this article: https://medium.com/bcggamma/using-deep-learning-to-predict-not-just-what-but-when-fae6515acb1b

I'd greatly appreciate your input on how to develop an LSTM model that can predict 'what' a consumer may buy and 'when' they will buy it;

Based on your article, it looks like the right model to choose would be Multiple Parallel Input and Multi-Step Output. Would you agree or do you think i should choose a different model? Any pointers or links to relevant articles would help!

Thanks,

Jason Brownlee January 22, 2019 at 11:42 am #

REPLY

I'd encourage you to prototype and explore a suite of different framings of the problem in order to discover what works best for your specific dataset.

Raman January 22, 2019 at 3:17 pm #

REPLY

```
def split_sequence(sequence, n_steps):
    X, y = list(), list()
    for i in range(len(sequence)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the sequence
        if end_ix > len(sequence)-1:
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

# define input sequence
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]
# choose a number of time steps
n_steps = 3
# split into samples
X, y = split_sequence(raw_seq, n_steps)
# reshape from [samples, timesteps] into [samples, timesteps, features]
n_features = 1
X = X.reshape((X.shape[0], X.shape[1], n_features))
# define model
model = Sequential()
```

```
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
```

I have used your code to get started, at the last step I am getting a below error-
NameError: name ‘to_list’ is not defined

Could you please help, I am not sure what am i missing here.

Thanks for your help

Jason Brownlee January 23, 2019 at 8:43 am #

REPLY

Ensure you have copied all of the code from the tutorial, I have more suggestions here:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Raman January 23, 2019 at 4:09 pm #

REPLY

Hi Jason,

Thanks for taking time out, I have copied your code line by line and checked couple of times as well.
Example is from Vanila LSTM.

Checks done-
I was getting some error, then I followed stack overflow and downgraded my keras to Version: 2.1.5
I searched stack overflow and related questions and even posted my questions there.

Your help is appreciated.

Jason Brownlee January 24, 2019 at 6:38 am #

REPLY

I recommend using the latest version of Keras and TensorFlow.

Sarra January 30, 2019 at 3:02 am #

REPLY

Please, have you an example of LSTM encoder-decoder with the train / test-evaluation partitions.

I tried but it does not work like this:

# split into samples

trainX, trainy = split_sequence(train, n_steps_in, n_steps_out)
testX, testy = split_sequence(test, n_steps_in, n_steps_out)

# reshape

```
trainX = trainX.reshape((trainX.shape[0], trainX.shape[1], n_features))
testX = testX.reshape((testX.shape[0], testX.shape[1], n_features))
….

# fit model
model.fit(trainX, trainy, epochs=5, verbose=2)

# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
```

thank you very much

Jason Brownlee January 30, 2019 at 8:14 am #

REPLY

I may, you can use the search box to look at all tutorials that use the encoder-decoder pattern.

Gunay February 6, 2019 at 2:43 am #

REPLY

Hi Jason,

Thanks for this tutorial. I am quite new to the time series forecasting with LSTM. I have a question about the part "Multiple Parallel Input and Multi-Step Output". The output data shape is (5,2,3). I mean the each instance on the output is not just a sequence, It is a sequence of sequence. And you have show the example there with Encoder and Decoder. I just want to implement one of the methods of Stacked or Bidirectional LSTM. But I am not sure which number I should put the Dense layer. For example, in the previous examples, the output shape is like (6,2) and It is obvious we should put 2 for the Dense layer. But I can not figure out the right thing for the Stacked LSTM. Do you have any example tutorial for this?

Kind Regards,
Gunay

Jason Brownlee February 6, 2019 at 7:51 am #

REPLY

With multi-step output, the number of nodes in the output layer must match the number of output time steps.

With multivariate multi-step, a vanilla or bidirectional LSTM is not suited. You could force it, but you will need n x m nodes in the output for n time steps for m time series. The time steps of each series would be flattened in this structure. You must interpret each of the outputs as a specific time step for a specific series consistently during training and prediction.

I don't have an example, it is not an ideal approach.

Gunay February 6, 2019 at 7:27 pm # REPLY

Thank you!

Gunay February 6, 2019 at 7:29 pm # REPLY

Is there any alternative structure for this kind of problems except Encoder-Decoder?

Jason Brownlee February 7, 2019 at 6:37 am # REPLY

Yes, the one I described. There may be others, it is good to brainstorm and prototype approaches.

Tian February 10, 2019 at 5:29 pm # REPLY

Thanks for your great tutorial. I just wonder should we avoid using bidirectional LSTM for time series data? Does it mean we use future data to train the past model parameters?

Jason Brownlee February 11, 2019 at 7:56 am # REPLY

No, it means the model will process the input sequence forwards and backwards at the same time.

Gunay February 15, 2019 at 8:56 am # REPLY

Hi Jason,

I faced one problem and just interesting maybe you did it before. I have the forecasting problem as like Multiple Input Multi-Step Output but a little bit different. Let's just assume, my input(which are features dataset) and output (target we want to forecast) datasets have historic data. And I should forecast one week ahead for the target. But I have also the one week ahead forecasted input dataset(which is forecasted by another system). I should use both the historic input and one week ahead forecasted input to forecast one week ahead output. But I do not know how I should use that one week ahead forecasted input data during the learning process. Can you give me any hint?

Jason Brownlee February 15, 2019 at 2:20 pm #

REPLY

Perhaps use a model with two heads, one for the historical data and one for the other forecast?

The functional API will help you to design a model of this type:
https://machinelearningmastery.com/keras-functional-api-deep-learning/

Anirban February 15, 2019 at 4:08 pm #

REPLY

What if we want to predict anything for the next 20 upcoming days! Here sequentially we have to predict for 20 days. How can we apply LSTM here?

Jason Brownlee February 16, 2019 at 6:13 am #

REPLY

Yes, although the further you predict into the future, the more error the model will make.

This is called multi-step forecasting, there are many examples, perhaps start here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Aaron March 6, 2019 at 2:47 pm #

REPLY

HI Jason, thanks for all the tutorials. They are really helpful. I am looking to try and implement an LSTM that returns a sequence, and had read this tutorial – https://machinelearningmastery.com/multi-step-time-series-forecasting-long-short-term-memory-networks-python/

One thing I am having trouble understanding is how to really shape the input data and get a sequence output using Tensorflow / Keras. I am looking to predict the sequence T – T+12 hours using T-1 – T-48 hours. So predicting the next 12 hours from the last 48 hours in 1 hour increments. Each hour of data has a dozen or so features for that time step. From what I have read of yours so far it seems as if each of the 48 previous time steps should be considered features of the time step T to predict a sequence for the next 12 hours. And so basically, from what I gather, I would end up with the input for Timestep T having 576 columns (48 time steps, each with 12 features) – I mean does that seem right? I am also a bit unsure of what particular model I should use… is it going to be a multi-step, multi-input network… just a bit confused on the jargon as well and maybe thats why I'm having trouble figuring out what I need to do.

Looking at some of your books too, but not sure what might be the right one to help guide me through a problem like this.

Thanks,
Aaron

Jason Brownlee March 6, 2019 at 2:49 pm # REPLY

Perhaps this will help:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

Aaron March 6, 2019 at 11:59 pm # REPLY

Thanks! That definitely makes sense now from the input shape standpoint. If I have 20 samples with 48 timesteps and 12 features the input shape would be [20, 48, 12]

For the output however, looking through the Keras docs https://keras.io/layers/recurrent/, I am trying to get a return sequence. Would I be using a 3D tensor? (batch_size, timesteps, units) where it would look like (20, 12, 1)? Since I am trying to find 1 value at each of the 12 time steps for the sample size of 20

Thanks again!
Aaron

Jason Brownlee March 7, 2019 at 6:52 am # REPLY

I don't recommend returning a sequence from the LSTM itself, instead use an encoder-decoder model:
https://machinelearningmastery.com/start-here/#lstm

Aaron March 7, 2019 at 10:05 am #

Why don't you recommend returning a sequence from the LSTM? If I was using the below encoder-decoder model from another one of your posts, what would the output of the first LSTM be?

```
model = Sequential()
model.add(LSTM(…, input_shape=(…)))
model.add(RepeatVector(…))
model.add(LSTM(…, return_sequences=True))
model.add(TimeDistributed(Dense(…)))
```

Jason Brownlee March 7, 2019 at 2:32 pm #

Generally the output sequence from an LSTM is the activation of the nodes from each step in the input sequence. It is unlikely to capture anything meaningful.

It is better to interpret these activations or the final activations with more LSTM or Dense layers, and the output a sequence of the same or different lengths using a separate model.

Gideon May 2, 2019 at 6:10 am #

Hi there,
I love this tutorial, all of your tutorials actually but this one I have found the most helpful. Questions about the MIMO LSTM output shape has come up a few times, and I am also having trouble with it.

I am trying to use a Dense layer as my final layer as you suggest, passing it n_steps_out as an argument. I am predicting 3 variables and n_steps_out is 10.

Keras complains that it is expecting the dense layer to have 2 dimensions, but I am passing it an array with shape (n_samples,n_steps_out,n_features)

Can you help me make sense of this?

Thank you

Jason Brownlee May 2, 2019 at 8:09 am #

I would recommend a model with a time distributed wrapper or decoder for multivariate multi-step output, so you can output one vector for each time step.

Abderrahim March 15, 2019 at 5:20 pm #

REPLY

Hi Jason,
I have a question: are LSTM suitable for predicting based on a test set with the same nature of inputs as of train set ? Like in other cases of prediction where you will be having input signals in train set, that the model will work on. plus the memory based on the fact that entries are ordered.
I trained an LSTM on a CNN model acting on ordered images, to predict a timeserie. on test set I have the following ordered set of images by time. I guess there is no concept of horizon here, how should I improve my model, and what starting point in predicting test set in this case?

Many thanks.

Jason Brownlee March 16, 2019 at 7:48 am # REPLY

I would recommend modeling the raw time series directly, instead of images of the time series.

Tayson March 26, 2019 at 12:06 am # REPLY

Hello Jason,

Many thanks for the helpful article..
I have tried to copy the code "Multiple Parallel Input and Multi-Step Output" and run it exactly the same without any changing but I got a different results than the one you got.

[[
[147.56306 167.8626 312.92883]
[185.38152 205.36024 385.96536]]]

Is there any reason for that?

Best regards,
Tayson

Jason Brownlee March 26, 2019 at 8:08 am # REPLY

Yes, this is due to the stochastic nature of the learning algorithm, more here:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

Chris March 26, 2019 at 1:27 am # REPLY

Hi Jason,
How would you handle building the LSTM model for time series data with irregular time intervals (e.g. Jan 1, Jan 2, Jan 4, Jan 7, Jan 13, Jan 14, etc…)?

It appears this model presupposes a regular time-interval spacing.

You could fill the "missing" days with zeros or impute them with, say, the mean of the last 3 values, but I would like to know how to make the LSTM model without filling/imputing the time series data. How would you handle this?

Thanks, and great lesson.

Jason Brownlee March 26, 2019 at 8:10 am # REPLY

Yes, I would try many approaches and compare results, such as:

– model as is
– normalize interval with padding
– upsample/downsample to new intervals
– etc.

Ron March 27, 2019 at 1:06 am # REPLY

If we are forecasting in monthly buckets and using 5 years of data, how do we know how many months of data to have on each row?

Jason Brownlee March 27, 2019 at 9:05 am # REPLY

Perhaps perform a sensitivity analysis of the model to see how history impacts model performance.

There will be a sweet spot for a given dataset.

Ron March 27, 2019 at 1:16 pm # REPLY

Thanks Jason! If the history has distinct patterns for each quarter, should we have 3 months in each row? How would the results differ when we keep 12 months on each row versus 3 months on each row versus 1 month on each row?

Jason Brownlee March 27, 2019 at 2:07 pm # REPLY

Depends on the dataset, I recommend testing to discover the specific answers with your data and model.

Peter March 28, 2019 at 5:57 am # REPLY

Hi Jason,

I am trying to predict high and low value of a time series in next X days, my output layer in RNN is :

model.add(Dense(2, activation='linear'))

so basically output vector is [y_high, y_low], the model works pretty well however it sometimes outputs y_low > y_high, which of course doesn't make any sense, is there a way to enforce model so that condition y_high >= y_low is always met.

Jason Brownlee March 28, 2019 at 8:24 am #

REPLY

Interesting, perhaps you simplify the problem and predict a value in a discrete ordinal interval, e.g. each category is a blocks of values?

Peter March 29, 2019 at 3:00 am #

REPLY

I was trying to modify loss function but I am unable to access y_pred individual members, I don't even know whether it's ultimately possible.

Jason Brownlee March 29, 2019 at 8:42 am #

REPLY

I don't follow, why not? What is the error?

Joe March 29, 2019 at 2:43 am #

REPLY

Hi Jason, a colleague and I are thinking of trying an LSTM model for time series forecasting. We are faced with over a thousand potential predictors, and would like to select only a smaller number for the final model. In particular, I have recently become fascinated by SHAP values; e.g., see this informal blog post by Scott Lundberg himself, in the context of XGBoost.
https://towardsdatascience.com/interpretable-machine-learning-with-xgboost-9ec80d148d27

Tantalizingly, Scott L. demonstrates SHAP values in the context of an LSTM model here:
https://slundberg.github.io/shap/notebooks/deep_explainer/Keras%20LSTM%20for%20IMDB%20Sentiment%20Classification.html
But that is using text input (sentiment classification in the IMDB data set), which involves an Embedding layer just before the LSTM layer. For a non-text problem like time series forecasting, we would exclude the Embedding layer. But doing so breaks the code.

Do you have any suggestions how SHAP values might be used in the context of LSTMs for time series forecasting (not text processing)? If not, do you have any suggestions for feature selection in that context?

Thanks!

Jason Brownlee March 29, 2019 at 8:42 am #

REPLY

I don't know what SHAP is, sorry.

Hsin March 29, 2019 at 5:26 pm #

REPLY

Hi Jason,
Thanks for this useful tutorial.
I am confused to inverse scaling of my data after splitting it into the form:
x(data_length, n_step, feature)
Because the scaler only can be used in 2D condition.

What I want to do is evaluate rmse between prediction and true values, so I have to inverse transform data. Could you please tell me how to deal with this problem?

Jason Brownlee March 30, 2019 at 6:23 am #

REPLY

Yes, I show how here:
https://machinelearningmastery.com/machine-learning-data-transforms-for-time-series-forecasting/

Pratik March 30, 2019 at 12:12 am #

REPLY

Hi Jason,
Firstly, I must say you have a fabulous chunk of articles on ML/DL. Thanks for helping out the community at large.

Coming to LSTMs, I am stuck in one problem from last few days. Here is how it goes –
I have 3 columns namely customer id and basket_index and timestamp. For every customer, each row represents one time stamp. Lets say there are 3 customers with variable time stamps. First one is having 30 time stamps, 2nd is having 25 and 3rd is having 50. So, the total number of rows are 105. Now for the column basket index, each row signifies a list of product keys bought by any customer on a particular timestamp. Here is the snapshot of the dataset –

CustomerID basket_index timestamp predicted_basket
111 [1,2,3] 1 [4,5]
111 [4,5] 2 [9,7]
111 [9,7] 3 [3,5,6,1]
.
.
222 [6,2,3] 1 [1,0,2,5]
222 [1,0,2,5] 2 [7,5]
.
.
333
.

. and so on..
Now, since every customer has a different time series,

1) How to pass everything into one network?
2) Do I have to build multiple LSTM models (one for each customer) in this case?

3) Also, I am creating an embedding layer for both customer and product keys (taking mean for every basket). How to specify how many steps back does every time series look in such cases?
4) How should I specify batch size in this case?

Your help will be really appreciated. Thanks!

Jason Brownlee March 30, 2019 at 6:29 am # REPLY

Great question, I have some suggestions here that might help:
https://machinelearningmastery.com/faq/single-faq/how-to-develop-forecast-models-for-multiple-sites

Generally, I would encourage you to try to learn across customers.

Huiping March 30, 2019 at 1:13 am # REPLY

Thanks Jason for nice post.

One question hopes to get your guide: For a LSTM work, we can't stop on say the model is good but most important is how to use the good model outcome.

For example flu or not for patients. Now I want to predict the flu for future half year (Jun-2019 to Dec-2019) but what I have is history data (I have past 4 years those people's flu data and target on that model is half year from 6-1-2018 to 12-31-2018).

How can I apply history LSTM outcome to predict future?

Can I get a list of important features from the history model with some value(like a weight) and apply this to my future data?

Or can i get the list of important feature from a good fit LSTM model and those features are important than other features?

Appreciate your guide!

Jason Brownlee March 30, 2019 at 6:30 am # REPLY

This is a common question that I answer here:
https://machinelearningmastery.com/make-predictions-long-short-term-memory-models-keras/

Jeyson Hernández April 3, 2019 at 11:40 am #

REPLY

Hi Jason,

Amazing work! Thanks sharing us your knowledge, this tutorial was so helpfull.

I'm new in ML/DL, i'm trying to predict sales in a company for future six months using LSTM. But i have an issue, i'm not sure about how to get more than 1 next step from your code using just one x vector by input. I'm using a monthly time step

Could you help me to understand a little bit better how to get it?

Jason Brownlee April 3, 2019 at 4:12 pm #

REPLY

There are many examples of multi-step forecasting that you can use, including in the above tutorial.

There are also more examples here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Md. Abul Kalam Azad April 4, 2019 at 3:05 pm #

REPLY

Dear Sir,

Thanks for your sharing example. I have collected traffic information like (Road property, weather, datetime,adjacent road speed, target road speed and more) for predicting road speed. Currently, I have prepared my code using Vanilla LSTM model for one step as well as multi-step-ahead prediction. Can you suggest me for which below model will be best for road speed prediction with higher accuracy?

Models are:
Data Preparation
Vanilla LSTM
Stacked LSTM
Bidirectional LSTM
CNN LSTM
ConvLSTM

I am waiting for your response.

Thanks,
Azad

Jason Brownlee April 5, 2019 at 6:11 am #

REPLY

I recommend testing a suite of model types and model configurations in order to discover what works best for your specific dataset, you can learn more here:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

Fazano April 8, 2019 at 8:58 pm #

REPLY

hi Jason, im using vanilla LSTM for forecasting,and i want to forecast 10 days ahead using this code

```
# Forecat real future

# Number of desired forecast in the future
L=10
#creat inputs and output empty matrices for future forecasting
Future_input=np.zeros((L,3))
Future=np.zeros((L,1))

#add last 3 forecast as input for forecasting next day (tommorow)
Future_input[0,:]=[predict[-3],predict[-2],predict[-1]]
#create 3 dimension input for LSTM inputs
Future_input= np.reshape(Future_input,(Future_input.shape[0],1,Future_input.shape[1]))
#predict tommorrow value
Future[0,0]=model.predict(np.expand_dims(Future_input[0],axis=0))

#Loop to predict next 9 days values
for i in range (0,9):
Future_input[i+1,0,:]=np.roll(Future_input[i,0,:], -1, axis=0)
Future_input[i+1,0,2]=Future[i,0]
Future[i+1,0]=model.predict(np.expand_dims(Future_input[i],axis=0))

#print 10 day ahead values
print(Future)
```

can it be like that?

Jason Brownlee April 9, 2019 at 6:23 am #

REPLY

Sorry, I cannot debug your code.

If you need more help with multi-step forecasting, see this:
https://machinelearningmastery.com/faq/single-faq/how-do-you-use-lstms-for-multi-step-time-series-forecasting

Sher April 13, 2019 at 2:41 am #

REPLY

Hi, do you have any tips for implementing univariate ConvLSTM for two-dimensional spatial-temporal data? I'm trying to input 10 time steps of 55 x 55 images for single-step time series forecasting.

The following error code appears:
"ValueError: Error when checking target: expected dense_10 to have 2 dimensions, but got array with shape (10, 55, 55)"

Jason Brownlee April 13, 2019 at 6:38 am #

REPLY

Sorry, I don't have a tutorial on the topic.

Randy April 13, 2019 at 6:21 am #

REPLY

Dear Sir,
i have sequence 1247 data and i want to forecast 30 next, so the data would be 1277.
i follow this tutorial, but it just can 1 or 2 forecast. and i follow this tutorial

https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/

but i get little confusion. so you have any advise to me?
its stock price data actually.

Jason Brownlee April 13, 2019 at 6:44 am #

REPLY

Stock prices are not predictable:
https://machinelearningmastery.com/faq/single-faq/can-you-help-me-with-machine-learning-for-finance-or-the-stock-market

Lloyd April 19, 2019 at 6:23 am #

REPLY

Amazing Tutorial, thank you.

I have a question, is there a model where the outputs can influence each other?

I.e. you have multiple sequences all which move independently but can influence the others?

Thank you

Jason Brownlee April 19, 2019 at 3:03 pm #

REPLY

Thanks.

Yes, an encoder-decoder model that outputs a time step for each series in concert might be such an approach.

Jules April 19, 2019 at 10:07 pm #

REPLY

Awesome. Great Explanation as always. I have always got rather frustrated and confused over the shape of data going into Keras models. So I relied upon your tutorials to make it clear.

Anyway using your examples I have been able demonstrate use of LSTM in predicting simple 2-D ballistics prediction calculations. I have used your code to help me here.

https://github.com/JulesVerny/BallisticsRNNPredictions

Pygame is required to animate the simulations

Jason Brownlee April 20, 2019 at 7:37 am #

REPLY

Well done, your project is very cool!

Kishore April 19, 2019 at 11:48 pm #

REPLY

Dear Prof,

Imagine I have raw text containing only words 'N1,N2,N3,…………,N1000' in a shuffled format , i.e, 1 million words, each of which can belong to any of these 1000 words.

I want to select the number of time steps =5, and predict the next word.
Eg: An input of [N1,N6,N5,N88,N32] would be followed by 'N73′.

Now, assume that I have tokenized all the 1000 possible words into numbers.

This is a scenario with 1000 possible output classes.
So should I replace model.add(Dense(1)) with model.add(Dense(1000,activation='softmax')) ?
If not, what is the main change I need to make, as compared to your univariate stacked LSTM code ?

Jason Brownlee April 20, 2019 at 7:39 am #

REPLY

If the words are shuffled, then there would be no structure for a model to learn.

HK April 23, 2019 at 8:06 pm #

REPLY

Dear Jason!

I’m trying to use stacked lstm for this problem – Multiple Parallel Input and Multi-Step Output. However I’m not sure how the final Dense layer should look like. Could you give me some hints, please?

Jason Brownlee April 24, 2019 at 7:57 am # REPLY

Perhaps start with the example in the above post and then add an additional LSTM layer?

HK April 29, 2019 at 6:11 am # REPLY

Which example do you mean? I can’t find any example for Multiple parallel input and multi step output LSTM, which uses stacked LSTM layers instead of encoder decoder.

Jason Brownlee April 29, 2019 at 8:28 am # REPLY

Yes, under the section “Multivariate Multi-Step LSTM Models”

Specifically the subsection “Multiple Parallel Input and Multi-Step Output”

The examples can be adapted to use any models you wish.

Raman Singh April 25, 2019 at 8:27 am # REPLY

Thanks Jason for detailed explanation.

Could you please tell how can we add hyperparameters for tuning “Forget Gate”, Input Gate” and “Output Gate” in LSTM compile or fit methods or is it done internally and we can’t control these gates?

Jason Brownlee April 25, 2019 at 8:28 am # REPLY

They are not tuned.

will April 25, 2019 at 1:36 pm # REPLY

How to predict multiple such inputs， x_input = array([[70,75,145], [80,85,165], [90,95,185],…, [200,205,405]]),Expect the next output， [210,215,425]，
See this input in the article，x_input = array([[70,75,145], [80,85,165], [90,95,185]])，Predict such results，

[[101.76599 108.730484 206.63577]]，But it doesn’t seem to matter why you need to enter such a sequence.in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])，in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
thanks

Jason Brownlee April 25, 2019 at 2:45 pm # REPLY

I believe there is a few multi-time step models listed above that will provide a good starting point.

John April 25, 2019 at 4:33 pm # REPLY

Hi Jason,

Thanks for the article.
I was working with your code and planning to implement in my work, but I have noticed a different behavior. If I compile and run the code different times, it gives different result each time although I didn’t change anything in your code. I have tried with your example data and run several times and each time I got different results. I tried with my own dataset and the result is the same.

Now I am confused to implement LSTM in my work.
Could you please clarify this behavior?

Jason Brownlee April 26, 2019 at 8:29 am # REPLY

Yes, this is to be expected. You can learn more here:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

Shiva April 26, 2019 at 10:59 pm # REPLY

Hi jason,

Say we have 3 variates(X).. and 1 dependent (Y)
The relation of 2 variate in X is like for 3 lags and 1 variate is 30 lag.

What is your advice when we have to model in such case?

Jason Brownlee April 27, 2019 at 6:32 am # REPLY

I have many examples, including a few above.

I also have a tutorial here that might help as a starting point:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/

Raghu April 28, 2019 at 3:45 pm #

REPLY

Hi Jason,

Thanks for the very informative tutorial. Can you please throw more light on how to come up with confidence intervals for the predicted value

Jason Brownlee April 29, 2019 at 8:16 am #

REPLY

Do you mean prediction intervals instead of confidence intervals?

Perhaps start here:
https://machinelearningmastery.com/prediction-intervals-for-machine-learning/

parsa April 28, 2019 at 10:42 pm #

REPLY

Hi Jason
Thanks for your helpful tutorial
Could you please tell how can we predict the futures that we don't have its data available
for example, I finalized my LSTM model, how can I predict the values on 2050

Jason Brownlee April 29, 2019 at 8:22 am #

REPLY

Yes, I show how in this post:
https://machinelearningmastery.com/make-predictions-long-short-term-memory-models-keras/

will April 28, 2019 at 11:41 pm #

REPLY

Thanks for the article.However, I have a problem that every prediction results are different, such as Multiple Parallel Series， The first time is [[101.25582 106.49429 207.8928]]， The second time it became [[101.82945 107.527626 209.8016]]， Why is this?
thanks

Jason Brownlee April 29, 2019 at 8:22 am #

REPLY

This is a common question that I answer here:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

I recommend that you try to run the example a few times.

shiva April 29, 2019 at 4:39 am #

REPLY

I want to restate my question…
Suppose we are trying to model a water bucket that was 1 open inlet at the top and 2 outlets at the side one near the top and one near the bottom.
this will mean that the outlet at the top can release when the water is really good..
the outlet near the bottom has release which is exponential function of water above it.

now say such systems are in paralell(one above another, say2) and series(say 2, the final outlet from each parallel series join at the final output.) (Total 4 buckets).

can this be modeled by LSTM?
I have done this analytically…results are ok ..
tyring to use lstm for this ,,,

Jason Brownlee April 29, 2019 at 8:27 am #

REPLY

Perhaps.

You can use this framework to explore different framings of your problem:
http://machinelearningmastery.com/how-to-define-your-machine-learning-problem/

Ali April 29, 2019 at 7:53 pm #

REPLY

Hello Jason,

to the step:

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])

I want to ask how I can load a fully column out of a dataset.
I don´t want to insert each value because I have more than 22 million rows. After that I want to split into sequences of 200-400 time steps.

To the step:

out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])

I don´t have a right mathematical equation. I want to predict the output without any knowledge about the relationship between the input signals.

I hope you can help me.

Kind regards

Ali

Jason Brownlee April 30, 2019 at 6:53 am # REPLY

I have many examples, perhaps start here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Sree April 29, 2019 at 8:59 pm # REPLY

Hi Jason,

Thanks for these explanations and sample codes!

I was interested in the example you have provided for multi-variate version of LSTM. You have provided an example of a simple addition case. How can this be extended to instances where there are multiple inputs, but an exact relation between the inputs are not known even though it is known that the inputs are correlated? Thanks much for your guidance!

Jason Brownlee April 30, 2019 at 6:54 am # REPLY

The model will learn the relationship, addition was just for demonstration.

Sree May 1, 2019 at 9:34 am # REPLY

Thanks Jason! That's perfect.

In that case, what should the statement "out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])" be replaced by, since we don't know the exact relation between the variables? Thanks again!

Jason Brownlee May 1, 2019 at 2:09 pm # REPLY

Observations across multiple input time series are aligned to time steps based on their time of observation.

Perhaps this will make things clearer:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

Sree May 2, 2019 at 8:17 pm # REPLY

Thanks Jason. I shall read the content on that link.

Cheers,
Sree.

Gideon May 3, 2019 at 5:37 am # REPLY

Hello, thank you again. I think my previous question could be made more clear.
I would like to use the vector output approach for a mimo lstm, making multi step predictions into the future similar to your encoder/decoder example.

I have tried using the split_sequences method from the encoder/decoder example with the vector output example and the dimensions dont work out. I end up with a value error

ValueError: Error when checking target: expected dense_2 to have 2 dimensions, but got array with shape (5, 2, 3)

I greatly appreciate your help, I have been struggling with this for a while. I would imagine the output should be a matrix (number of features X prediction horizon) so I think there is something conceptually I am not understanding.

Thank you, and thank you for all of the wonderful tutorials

Gideon

Jason Brownlee May 3, 2019 at 6:25 am # REPLY

Perhaps start wit the code example you want to use and slowly change it for your needs.

If the data size does not match the models expectations, you will need to change the data shape or change the model's expectations.

Gideon May 3, 2019 at 7:30 am # REPLY

I will toil away some more, but I just want to be sure it is possible to use a dense layer/vector output approach for Multiple Parallel Input and Multi-Step Output LSTM in Keras.
Thanks again for your time.

Gideon

Jason Brownlee May 3, 2019 at 2:40 pm #

REPLY

It is possible to use a Dense for multi-step multivariate output without a decoder or timedistributed wrapper layer, it is just ugly.

E.g. the output would be a vector with n x m nodes, where n is number of variates and m is the number of steps.

Gideon Prior May 4, 2019 at 8:20 am #

Ive figured it out, and its not too ugly and exactly what I needed. I was unaware of the Reshape layer in Keras.

from keras.layers import Reshape

…

model.add(Dense(n_steps_out*n_features))
model.add(Reshape((n_steps_out,n_features)))

Thank you again for your help. I am buying your book right now.

Cheers

Gideon

Jason Brownlee May 5, 2019 at 6:18 am #

Nice work.

George July 17, 2019 at 11:48 pm #

REPLY

Hi Gideon,

I was struggling around something similar and applying your solution, solved all the matters! Do you have any more documentation on this?

Alberto May 3, 2019 at 11:07 pm # REPLY

Hello Jason,

Great article, very useful. I want to use LSTM to predict sun irradiance 12 hours ahead using 8 features (including sun irradiance) of the last 24 hours as inputs. Thus, it would be a multivariate multi-step LSTM where the output is a sequence of 12 timesteps. I have 8 years of data and I want to use first 6 for training and last 2 for testing. I have some questions:

1) Should I overlap the input sequences?

2) Should I use a vector output model or an encoder-decoder model?

Jason Brownlee May 4, 2019 at 7:08 am # REPLY

I recommend testing both approaches and use data to make the decision, e.g. choose the model that gives the best result.

aravind May 5, 2019 at 3:54 am # REPLY

hai jason,
the article was very much helpful.
can you just tell me which approach should I take if I have two columns in my dataset .
one is time in ddmmyyyy format and the other is stock price.
I have the data for last 12 months.
I want to predict the stock price for 4 upcoming months.
how can I do the same.
one more doubt is that if the column for time is not actually having a same interval in between them, then is there anything more that I should do to or consider for predicting the 4 upcoming months stock price

Jason Brownlee May 5, 2019 at 6:35 am # REPLY

You can drop the date if all observations have a consistent interval.

Stock prices are not predictable:
https://machinelearningmastery.com/faq/single-faq/can-you-help-me-with-machine-learning-for-finance-or-the-stock-market

Regardless, I would recommend this process:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

aravind May 5, 2019 at 3:44 pm # REPLY

So after dropping the dates column I guess I would have to go for a univariate multistep lstm model. right?
And what should be done if seasonality comes into picture?
and one more doubt is that when I am predicting the four upcoming months in a multistep , will the model consider the predicted value of 3rd month while predicting the fourth month or will the model consider the predicted value of 2rd month while predicting the third month and likewise ?

Jason Brownlee May 6, 2019 at 6:46 am #

REPLY

Try a suite of models, and compare to a linear model or naive model to confirm they have skill.

If you have seasonality, try modeling with and without the seasonality and compare performance.

Try multiple approaches for multi-step prediction, e.g. direct, recursive, etc:
https://machinelearningmastery.com/multi-step-time-series-forecasting/

aravind May 7, 2019 at 2:50 am #

If you have seasonality, try modeling with and without the seasonality and compare performance.
what does this mean? I didn't get you
my training data will anyway contain the seasonality if my original dataset has seasonality right?
how will I be able to make a model without seasonality?
is there a additional parameter or feature in LSTM for seasonality?

Jason Brownlee May 7, 2019 at 6:18 am #

You can remove seasonality from a dataset via seasonal differencing:
https://machinelearningmastery.com/remove-trends-seasonality-difference-transform-python/

shiva May 7, 2019 at 7:33 am #

REPLY

In Multiple Input Series,
(7, 3, 2) (7,)

[[10 15]
[20 25]

[30 35]] 65
[[20 25]
[30 35]
[40 45]] 85
[[30 35]
[40 45]
[50 55]] 105
[[40 45]
[50 55]
[60 65]] 125
[[50 55]
[60 65]
[70 75]] 145
[[60 65]
[70 75]
[80 85]] 165
[[70 75]
[80 85]
[90 95]] 185

1. How many lstm block will be here in this example(x=7)
if batch size = 3,is the number of lstm block equal to the number of x in the batches?
or the number of timesteps?

2.are timesteps, neurons and batchsize all hyperparameter? how do we optimize them

Jason Brownlee May 7, 2019 at 2:26 pm # REPLY

No, the number of blocks in the first hidden layer is unrelated to the length of the input sequences.

See this:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

shiva May 8, 2019 at 4:56 am # REPLY

thanks..
Then what is the total number of LSTM blocks?
for every epoch, are the weights reinitialized and states are reset?

Jason Brownlee May 8, 2019 at 6:46 am # REPLY

The number of LSTM units is specified in each hidden LSTM layer.

LSTM states are reset at the end of every batch.

shiva May 8, 2019 at 8:15 am # REPLY

sorry but i dont get this?

In model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
input_shape here is equal to an input to each LSTM node right?

and here 50 means,, h(hidden layer) is a vector of 50*1 right?

my question is the number of individual LSTM nodes(block) equal to number of samples in the a batch?

Jason Brownlee May 8, 2019 at 2:08 pm # REPLY

Yes, the shape defines the shape of each input sample (time steps and features).

Yes, 50 refers to units in the first hidden layer.

The number of units and sample shape are both unrelated to the batch size. Unless you are working with a stateful LSTM, in which case the input shape must also specify the batch size.

Does that help?

shiva May 8, 2019 at 11:08 pm # REPLY

yeah.. one followup question
[10 15]
[20 25]
[30 35]] 65
here is it like many to one ?

this feeds as xt (single input) right?
in this case what is the size of weight ?

Jason Brownlee May 9, 2019 at 6:43 am # REPLY

Yes, multivariate multistep input to one output.

shiva May 9, 2019 at 11:05 am # REPLY

how does this input concatenate with hidden layer … i cannot visualize this..
i was thinking the input were a vector[n*1]

Jason Brownlee May 9, 2019 at 2:05 pm #

Each node in the hidden layer gets the complete put sequence.

shiva May 12, 2019 at 9:33 am #

Thank you so much..

[10 15]
[20 25]
[30 35]] 65

so in this case ,,, what is the size of xt and weight matrix?

Jason Brownlee May 13, 2019 at 6:42 am #

You can calculate it based on the number of nodes in your network.

shiva May 15, 2019 at 10:32 pm #

Thank you jason.. you have so kind and helpful..

shiva May 8, 2019 at 11:07 am #

REPLY

The number of cells is equal to the number of fixed time steps.
The blogs says so. I am very confused with number of cells and what controls it.

https://stackoverflow.com/questions/37901047/what-is-num-units-in-tensorflow-basiclstmcell#39440218

Sorry for trouble

Jason Brownlee May 8, 2019 at 2:11 pm #

REPLY

Not in the Keras implementation.

Philipp May 13, 2019 at 2:26 am #

REPLY

Dear Jason,

Thank you for writing all these awesome tutorials!

My question:
As I understood it, a LSTM network learns the information in a time-series by backpropagation through a specific length (in time) at which the LSTM cells are unrolled during training.
So, while training it is necessary to define the number of timesteps provided in the training data. But shouldn't it be possible to use the (trained) network with ANY number of input timesteps to make a prediction (because of the recurrent nature in which the LSTM cells work)?
Am I getting something wrong here from the beginning?

Thank you for hints on this
Philipp

sumitra May 13, 2019 at 3:41 pm #

REPLY

Dear Jason,

I am currently working on a disease outbreak prediction model. I have 4 years of data with over 100 input variables and each year has got 365 data points. I would like to create a LSTM model that will be able to predict the future outbreak (whether thr will be an outbreak-1 or no outbreak-0) based on the given input variables. For example, given 7 days of data points, i would like to predict the occurance of outbreak (whether 0 or 1) on the 8th day.

However, i am not sure on which LSTM model will best fit my case. Will 'multiple input multi-step output) be the best approach? Your guidance will be much appreciated.

Thank you

Jason Brownlee May 14, 2019 at 7:38 am #

REPLY

Perhaps you can model it as a time series classification problem.

The tutorials here will help you to get started:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Nitin May 26, 2019 at 1:54 pm #

REPLY

Hi Jason,

Can you please provide some pointers that will help us in minimizing the step-loss during model fitting….

Thanks

Jason Brownlee May 27, 2019 at 6:43 am # REPLY

Yes, here are some suggestions:
https://machinelearningmastery.com/start-here/#better

ICHaLiL May 29, 2019 at 12:26 am # REPLY

Dear Jason,

Thank you for your tutorials. They are really useful for us.

I've one question about LSTM. I have different time series more than one (for example 100). I need to train network with 100 different time series. and test 10 different time series. Which method should I use?

Thanks for your helps.

Jason Brownlee May 29, 2019 at 8:44 am # REPLY

I recommend this process:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

QuantCub May 30, 2019 at 2:00 pm # REPLY

Hi Jason,

Thank you for sharing. I wonder if there is a way to set timestep > 1 without doing subsequence sampling as you did in data preparation, e.g. convert a 9-by-1 time series to a 6-by-3 data set. After the conversion, the 3-feature dataset is no more time dependent. You are able to use any kind of ML models (say OLS) to predict y. So why LSTM? Should LSTM be able to select (forget) previous information without this conversion?

Jason Brownlee May 30, 2019 at 2:55 pm # REPLY

LSTM does have the benefit that it can remember across samples.

This may or may not be useful, and is often not useful for simple autoregressions.

QuantCub May 31, 2019 at 1:19 am # REPLY

Thank you for your quick reply. In your example, if I do a subsequence sampling and convert
[10, 20, 30, 40, 50, 60, 70, 80, 90] to
[[10, 20, 30],
[40, 50, 60],
[70, 80, 90]] (no replacement between each subsequence)
and run LSTM(input_shape=(3,1)), is that the same as I run LSTM(batch_input_shape=(3,1,1), stateful=True) on the origin time series (9-by-1)?

Jason Brownlee May 31, 2019 at 7:51 am # REPLY

No, as each "sample" will cause an output from the model, e.g. 1 sample with 3 time steps vs 3 samples with 1 time step.

More on samples vs timesteps here:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

QuantCub May 31, 2019 at 12:49 pm #

Thank you, Jason!

Jason Brownlee May 31, 2019 at 2:47 pm #

No problem.

Neel June 11, 2019 at 9:08 pm # REPLY

For a classification LSTM, using a Seed I get the same classification matrix each time I run it. However, when I vary the batch size in model.predict, I get the following:

Prediction Batch Sizes:

32 = Different Classification Matrix on each repeat

Batch size in predictions is merely for ram managment. Correct? If yes, what do you think Dr. Jason would cause these irregularities ?

Jason Brownlee June 12, 2019 at 8:01 am # REPLY

No batch size impacts the learning algorithm:
https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

Neel June 12, 2019 at 4:06 pm # REPLY

Hi Jason,

Sorry I didn't explain my concern well. I was referring to the Batch Size parameter that we mention in "model.predict i.e. predicting" and not while training. I agree that batch size during training will have an impact. During prediction, the default size is 32 as defined by keras but when I change that to anything but 32 I get a different classification matrix even though I use a seed. When I leave the batch size as default, my seed is able to produce the same results.

Jason Brownlee June 13, 2019 at 6:11 am # REPLY

Recall that with the LSTM, the state is reset at the end of each batch. This explains why you are getting different results for the same model with different inference batch sizes.

Diego June 24, 2019 at 5:23 am # REPLY

Hi Jason,

Thanks for the tutorial.
I'd like to apply this example to a real case.

I have to forecast how much money will be withdrawn every day from a group of ATMs.
Currently I am using a time series for every ATM. (100 ATMs = 100 time series).

Wich method do you think could be better from this tutorial ?
I need to use historical information and external information such as holidays, day of week, etc.
Thanks in advance.

Jason Brownlee June 24, 2019 at 6:37 am # REPLY

I recommend following this framework:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

Liang Zhao June 25, 2019 at 5:58 am #

REPLY

Hi Jason, I want to use some kind of machine learning method to demonstrate that there is a relationship between the score gap of two basketball teams and the demand for a taxi outside the stadium.

I have time series of pick-ups near a stadium. I have the score gap time series between two basketball teams.

What I want to achieve is that training a machine learning model that could tell me, based on the taxi pick-ups at time t, what is the taxi pick-ups at time t+1.
I also want to see if I also have the score gap at time t, can I improve my prediction accuracy of pick-ups at time t+1.

Which machine learning model should I use?

thank you so much!

Jason Brownlee June 25, 2019 at 6:30 am #

REPLY

Why not just measure statistical correlation between the observations?
https://machinelearningmastery.com/how-to-use-correlation-to-understand-the-relationship-between-variables/

---

liang zhao June 26, 2019 at 7:26 pm #

REPLY

Thank you very much for your reply!
Yes, this could work for finding a relationship.
but what if I want to forecast the number of pick-ups at t+1. Can LSTM or ARIMA do this job?

Jason Brownlee June 27, 2019 at 7:48 am #

REPLY

Yes, but perhaps test a suite of methods and discover what works best for your specific dataset.

James July 1, 2019 at 1:11 am #

REPLY

Hi Jason,

Thanks for the tutorial.

Suppose I have several time series showing cumulative bookings for different trains last year. I don't want to forecast but just classify those time series to see if some of them have similar patterns. Can I include all those series into one LSTM model? Is there any risks when doing so?

Thanks in advance.

Jason Brownlee July 1, 2019 at 6:35 am # REPLY

Sure, it means you are learning/modeling across books. Sounds reasonable.

James July 1, 2019 at 11:44 pm # REPLY

Thanks Jason!

So is it the same as multivariate LSTM? Sorry I'm new to modelling so still find things confusing

Jason Brownlee July 2, 2019 at 7:32 am # REPLY

Probably not, each example is a separate sample or input-output pair for the model to learn from.

Irini July 1, 2019 at 9:35 am # REPLY

Hi Jason,

thanks for the nice tutorial!

I have a dataset with 3000 univariate timeseries (i.e. 3000 samples) and each sample has 4000 timesteps.
When i use [samples, time steps, features]=[3000, 4000, 1] the code is extremely slow and with bad performance.
On the other hand, if instead [3000, 4000, 1] i write [3000, 1, 4000] the code is very fast and with great performance.
But is the reshape [3000, 1, 4000] correct? I mean according to the rule [samples, time steps, features] and given the fact that each of my samples have 4000 timesteps and for each time step there is one feature the correct should be [3000, 4000, 1].

So is [3000, 1, 4000] correct? And if it is not (logically it is not) why it works much better than [3000, 4000, 1] ?

Thanks in advance

Jason Brownlee July 1, 2019 at 11:35 am # REPLY

I would recommend not using more than 200 to 400 time steps per sample. Perhaps you can truncate your data?

Irini July 1, 2019 at 8:15 pm # REPLY

I did also an experiment and i truncated my data and used as input [samples, time steps, features]=[3000, 400, 1]. It was quicker but i got a mean accuracy 42% (in 10 random splits).
As i told you in my previous post when i exchange timesteps with features namely when i use [3000, 1, 4000] i get an accuracy 90%.
But giving 1 timestep means that i don't exploit the memory, whis is the characteristic of lstm?

I am confused as to whether i should use [3000, 1, 4000], which is very quick and gives very good results but maybe it is not very correct? Or it is correct as if i used [3000, 400, 1](if i truncated my data to 400)

Jason Brownlee July 2, 2019 at 7:30 am # REPLY

The state of the LSTM is reset at the end of each batch by default, so you can get some across-sample memory.

I recommend testing a suite of different configurations to see what works well or best for your specific dataset. I cannot know what will work well, you must discover the answer.

Manish July 2, 2019 at 2:39 am # REPLY

Hello Jason,

I am quite new to ML and LSTMs. I have a scenario where I intend to train a model using my hourly sensor values. For eg

12-1-2019 12:00:00 12
12-1-2019 13:00:00 16
…
12-5-2019 12:00:00 14

Once I am done with my training I intend to predict values every hour and compare the values with live sensor values….I am planning to use LSTM and which approach do you recommend me ?

Jason Brownlee July 2, 2019 at 7:35 am # REPLY

I recommend this framework:
https://machinelearningmastery.com/how-to-develop-a-skilful-time-series-forecasting-model/

Harish July 2, 2019 at 7:18 pm # REPLY

Jason, this is very useful. Im try to to do some prediction around IT incidents. based on historic data i want to predict what type incident i can expect next month/week/day. do you have anything similar done if so request to share pls

Jason Brownlee July 3, 2019 at 8:32 am # REPLY

Perhaps you can model it as a time series classification, e.g. what event is predicted in this interval.

The tutorials here might help as a starting point:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Lopa July 3, 2019 at 12:35 am # REPLY

Hi Jason,

Thanks for answering my question in your other tutorials. I have a minor doubt suppose my data has a continuous time series(non stationary) & other categorical variables (which are already encoded). Under that circumstance what is the best way to difference the data ? Because categorical data are not differenced but they have to be used while training the model.

The function written above differences all the variables irrespective of whether they are continuous or categorical. It would be great if you can help.

Jason Brownlee July 3, 2019 at 8:36 am # REPLY

Difference the real-values only, and only if they are non-stationary.

myro July 3, 2019 at 5:16 pm # REPLY

Hi Jason,
I copied and pasted your first example from Multi-Step LSTM Models, the one with the vector output of two values and the input being one.

You report as an output the values:

input [[70 80 90]]
output [[100.98096 113.28924]]

but with those parameters I cannot get any closer than

input [[70 80 90]]
output [[122.678955 139.9465]]

This you use the parameters you report? Is this so dependant on architecture?

Jason Brownlee July 4, 2019 at 7:40 am # REPLY

Results are dependent upon the model, the model configuration and the data, the performance is also stochastic, subject to random variance.

---

myro July 19, 2019 at 7:49 pm # REPLY

Hi, thanks for your reply.
I understand that, that’s why I am asking,
I have same model, same model config. same data, and the stochasticity should be symmetrically distributed (?). Then I assume that the results you report are not from the parameters you have in the code examples.

Lopa July 3, 2019 at 7:10 pm # REPLY

My data is non stationary & there are seasonality every 7 days (as evident from the ADF tests & ETS plots) & a first order differencing makes it stationary.

I totally get that I have to difference only the real values & that is what I have been aiming to do . But the reason I asked this question because the moment I difference the real values its get shifted by one place so if the original data has 100 observations the differenced data will have 99 observations (with a first order differencing). But the categorical data which cannot be differenced remains to be the same 100. How do I deal with this ?

Jason Brownlee July 4, 2019 at 7:44 am # REPLY

You discard the first observation and the difference value corresponds to the categorical value at the same time step.

Lopa July 3, 2019 at 8:06 pm # REPLY

I think I have been able to solve the issue thanks Jason for addressing my query

Jason Brownlee July 4, 2019 at 7:44 am # REPLY

I'm happy to hear that.

Leon July 5, 2019 at 5:58 am # REPLY

in the Vector Output Model section,
I copied your code and tried, the actual answer is not correct as of the expected [100, 110], they are actually [110, 120].

Jason Brownlee July 5, 2019 at 8:11 am # REPLY

Perhaps try running the example a few times? It can very given the stochastic nature of the learning algorithm.

Leon July 11, 2019 at 1:30 am # REPLY

never get any chance to around [100, 110]. I ran many times, the output is always around [110, 120] with some variations.

no kidding 🙂 you can try that part of codes. The output looks ridiculous.

Jason Brownlee July 11, 2019 at 9:50 am # REPLY

Intersting.

Is Keras/TensorFlow/Python up to date?

Matthew July 5, 2019 at 5:56 pm # REPLY

Hi Jason, I am doing an electrical demand forecast and am trying to build a model which predicts the demand for the following 24 hours given the last 90 hours. I have implemented two types: a 24 step prediction and a recursively defined prediction, which predicts the next hour and then uses the previous 89 true values and the new predicted value to predict the next value, and so on. I am wondering which method you believe to be the best(if either) and any tips for improving my model as depending on the time of year the forecast can vary massively with accuracy. I currently have an LSTM(50) connected to a Dense(20)

connected to an output Dense(1) for both cases.
Any help would be greatly appreciated. Thank you. Matthew

Jason Brownlee July 6, 2019 at 8:29 am # REPLY

Well done, very cool!

I recommend testing each method and use the one with the lowest error.

Also, get creative and test a suite of other configurations. Ensure your test harness is robust and reliable so that you can trust the decisions you make.

skyrim4ever July 8, 2019 at 8:13 pm # REPLY

Hello, this example was nice to follow and seemed little more simpler than other LSTM examples because of no pre-processinhg transformations (normalization, standardization, making data into stationary, etc.). However, should I perform these pre-processing transformations in general for time series prediction? Should I do such thing for this kind of examples too even though the dataset is simple?

Jason Brownlee July 9, 2019 at 8:09 am # REPLY

Yes, test to see if the data preparation improves model performance.

I keep it out of examples for brevity.

Nutakki July 11, 2019 at 8:59 pm # REPLY

#In Multiple Parallel Series
I have defined the input like this
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
x_input = array([[70,75,1,4], [80,85,165,5], [90,95,185,6]])
n_steps=4
n_features=X.shape[2]

how the input is looping to obtain output as follows: [[72.74373 106.51455 251.78499]]?
Can you give a clear idea what does n_steps=4, n_features=X.shape[2] really means and how does it function?

Jason Brownlee July 12, 2019 at 8:40 am # REPLY

Yes, perhaps this will help you to understand the input shape:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

Amelie July 12, 2019 at 2:10 am # REPLY

Please, is there a method to find the correct parameter of an ANN model: LSTM, MLP (hidden layer number, activation function, loss function ..)

what does it mean when my train and validation loss curves are parallel while the Train Score and Test Score are small?

Is there a method to optimize all these results?

Jason Brownlee July 12, 2019 at 8:45 am # REPLY

Yes, see this post:
https://machinelearningmastery.com/faq/single-faq/how-many-layers-and-nodes-do-i-need-in-my-neural-network

Samil July 17, 2019 at 8:16 am # REPLY

Thanks for the tutorial. I have applied the multistep, multivariate logic to my own dataset. Namely, I have 12 look-back, 12 look-ahead and 41 features (all having exact look-back as the main variable of interest). Trying the TimeDistributed code snippet gave me progressively increasing RMSE. Is this due to the nature of my time series or is it a sign of mistake done during construction of the model? It is hard to tell for you but maybe you can share your take on this issue. Thanks

Jason Brownlee July 17, 2019 at 8:33 am # REPLY

It could be either.

Perhaps try fewer features and evaluate impact?
Perhaps try different models and evaluate impact?

Samil July 19, 2019 at 9:42 am # REPLY

Thanks II tried encoder-decoder and stack LSTM. Both gives me increasing RMSE for further look-aheads.It is understandable for encoder-decoder as it uses the output as an input (so associated error also comes with the prediction and builds up over time) but not sure why I see the same thing with the stack lstm. Anyways, thanks again for the response and the post!

Samil July 19, 2019 at 9:48 am # REPLY

Also, one quick related question. You use "-1" in multi step future multivariate split_sequence models (such as n_steps_out-1 etc.). This reduces the number of resulting features by one when compared to other split_sequence snippets. I tested it with the other multistep split_sequence code you shared above. Not sure but are'nt we supposed to have the same number of features? Thanks

Jason Brownlee July 19, 2019 at 2:20 pm #

Thanks.

Jason Brownlee July 19, 2019 at 2:20 pm # REPLY

Well done on the improvement!

Aziz Ahmad July 22, 2019 at 4:56 am # REPLY

Sir Plz! Suggest me good learning sources about my project (carbon emission forcasting using LSTM).

Jason Brownlee July 22, 2019 at 8:28 am # REPLY

Start here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

Mans Oshanov July 22, 2019 at 6:20 pm # REPLY

Thank you for the great tutorial. Is it possible to get the probability of prediction(in percentage) or second best prediction out of these models? Thank you)

Jason Brownlee July 23, 2019 at 7:58 am # REPLY

Yes, model.predict() will return a probability on classification tasks.

More details here:
https://machinelearningmastery.com/how-to-make-classification-and-regression-predictions-for-deep-learning-models-in-keras/

Kennard July 26, 2019 at 11:57 am # REPLY

Hi, Jason

Your tutorial helps me a lot, thank you very much!

And I have a question that how to adjust the learning rate of the LSTM network in the CNN-LSTM code you've mentioned above.

I'm looking forward to your reply, thank you!

(The reply I left in https://machinelearningmastery.com/how-to-develop-lstm-models-for-multi-step-time-series-forecasting-of-household-power-consumption/?unapproved=494293&moderation-hash=2b6d045a4e1ff047d0720753b2b1e418#comment-494293 is in wrong place, sorry about that)

Jason Brownlee July 26, 2019 at 2:18 pm # REPLY

You can learn more about how to tune the learning rate (generally) here:
https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Luis July 27, 2019 at 3:10 am # REPLY

This is amazing. I love the blog

Jason Brownlee July 27, 2019 at 6:11 am # REPLY

Thanks Luis.

Armande Kertanio July 28, 2019 at 8:31 am # REPLY

Thank for this nice explanation.

I have a problem when reshaping the data for multiple output architecture.

the architecture is:

outputs=[]

main_input = Input(shape= (seq_length,feature_cnt), name='main_input')
lstm = LSTM(32,return_sequences=True)(main_input)
for _ in range((5)):
prediction = LSTM(8,return_sequences=False)(lstm)
out = Dense(1)(prediction)
outputs.append(out)

model = Model(inputs=main_input, outputs=outputs)
model.compile(optimizer='rmsprop',loss='mse')

and when reshaping the y using:

y=y.reshape((len(y),5,1))

I got a reshaping error:

ValueError: Error when checking model target: the list of Numpy arrays that you are passing to your model is not the size the model expected. Expected to see 5 array(s), but instead got the following list of 1 arrays: [array([[0.35128802, 0.01439778, 0.60109704, 0.52722118, 0.25493708],

would you please help?

Jason Brownlee July 29, 2019 at 5:58 am #

REPLY

Perhaps define what you want the output shape to be, e.g. n samples with m time steps, then confirm your data has that shape, or if not set that shape?

Florian July 29, 2019 at 2:00 am #

REPLY

You use "model.add(TimeDistributed(MaxPooling1D(pool_size=2)))" and write "max pooling layer that distills the filter maps down to 1/4 of their size". A typo or is there a different reason explaining the use of 2 vs. 4 here?

Jason Brownlee July 29, 2019 at 6:16 am #

REPLY

Sorry for the confusion.

If the map is 8×8 and we apply a 2×2 pooling layer, then we get a 4×4 out, e.g. 1/4 the area (64 down to 16).

For time series, if we have 1×8 and apply a 1×2 pooling, we get 1×4, you're right. 1/2 the size, not 1/4 as in image data.

Fixed. Thnaks!

Nic July 29, 2019 at 5:53 pm #

REPLY

Hi Jason,

first of all, thanks for that awesome introduction into LSTM-Models.
There is just one thing i don't get.

In the section "Multiple Input Series" you used the following example:
[[10 15 25]
[20 25 45]
[30 35 65]
[40 45 85]
[50 55 105]
[60 65 125]
[70 75 145]
[80 85 165]
[90 95 185]]

As you mentioned the first two entries in the arrays refer to the two time series and the last one to the corresponding target variable. To train the LSTM you split the data into input and output samples like:
[[10 15]
[20 25]
[30 35]] 65

Why do I drop the first two target entries (25 and 45). Isn't that information my network loses for training? Why don't we use each (single) sample like x = [10 15] y[25] to train the time series. Isn't it easier to lern the series if i have the target for each step?

Jason Brownlee July 30, 2019 at 6:04 am #

REPLY

Good question.

We must create samples of inputs and outputs.

Some of the input at the beginning of the dataset don't have enough prior data to recreate an input, therefore must be removed.

Joel August 1, 2019 at 12:04 am #

REPLY

Good work, However, you should provide the library imports, to make it easier for beginners.

Jason Brownlee August 1, 2019 at 6:53 am #

REPLY

All library inputs are provided in the “complete example” listed in the post.

Sorry for the confusion.

will August 11, 2019 at 12:27 am #

REPLY

Hi, Jason,I need to predict a hundred thousand sequences like this[10, 20, 30, 40, 50, 60, 70, 80, 90], how do I do it, do I do it in cycles, one by one, I do it in cycles, it feels like it’s going to take longer

Jason Brownlee August 11, 2019 at 5:59 am #

REPLY

If the model is read only and you are not dependent upon state across samples, you can run the model in parallel on different machines and prepare batches of samples for each model to make predictions.

PRADEEP CHAKRAVARTHI NUTAKKI August 11, 2019 at 3:42 am #

REPLY

Hi, I am very happy to have this LSTM example to have a practice.

I have a problem as follows:

I have 300 excel workbooks of which each excel sheet has 3 values…..

the 3 values will be in this format [1.02,2.20,1.0]; [2.9,3.5,3.3];…….like this 300 sets.

Now i want to train and test my model with the data from 300 excel workbooks as input and the model has to predict the 301th set for example: [5,3.3,2.4] depending on the sequence of previous values.
Note: the output shouldn’t be the probability set from the 300 sets, the output should be a new set.

Can you suggest me any solution to this problem?

Jason Brownlee August 11, 2019 at 6:04 am #

REPLY

Perhaps you can use some custom code to extract all of the data from the excel files into a csv file ready for modeling?

y jing August 12, 2019 at 4:45 pm # REPLY

How to construct parallel three lstms, and then add a DNN in series.

Jason Brownlee August 13, 2019 at 6:06 am # REPLY

You can use one LSTM with 3 variables or 3 LSTMs and concat the outputs together.

See the functional API:
https://machinelearningmastery.com/keras-functional-api-deep-learning/

Doron August 13, 2019 at 4:42 pm # REPLY

Hi Jason,

Thanks for this wonderful post. I have been trying to digest LSTM's (metaphorically) and one particular aspect was not clear to me. I know the general structure of LSTM's but I'm having hard time to understand:

model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))

When ReLU is set as an activation function, but not in the output layer, what exactly happens behind the scenes? To make myself clear, I am aware of the gates and their respective activation functions: sigmoid and tanh. But if we set ReLU like above, does that mean that each unit/LSTM cell outputs a hidden state –> pass it to a ReLu –> pass it to the next unit/LSTM cell?

Thanks!

Jason Brownlee August 14, 2019 at 6:33 am # REPLY

Yes, that is correct. It controls the output gate, not the internal gates which are governed by a sigmoid.

Dawjidda August 17, 2019 at 1:46 am # REPLY

hello Mr Jason Brownlee please my dataset is in matrics form, i want convert it to fit into GRU or LSTM sequential model,

Jason Brownlee August 17, 2019 at 5:55 am # REPLY

If your matrix represents a sequence, you can reshape it for your model. This will help:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

If not, an RNN would not be appropriate.

Tommy August 18, 2019 at 9:22 pm #

REPLY

Hi Jason,

A problem is involved in my mind, If it is possible, I want to know your opinion.

What will happen if we use both lstm and gru layers simultaneously in the model? Does this make sense? For example this architecture:

```
model=Sequential()
model.add(GRU(256 , input_shape = (x.shape[1], x.shape[2]) , return_sequences=True))
model.add(LSTM(256))
model.add(Dense(64))
model.add(Dense(1))
```

Because I used this model and I got good results compared to using each one separately.

Jason Brownlee August 19, 2019 at 6:06 am #

REPLY

You can, but why?

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery!

Hi, I'm **Jason Brownlee**, PhD.
I write tutorials to help developers (*like you*) get results with machine learning.

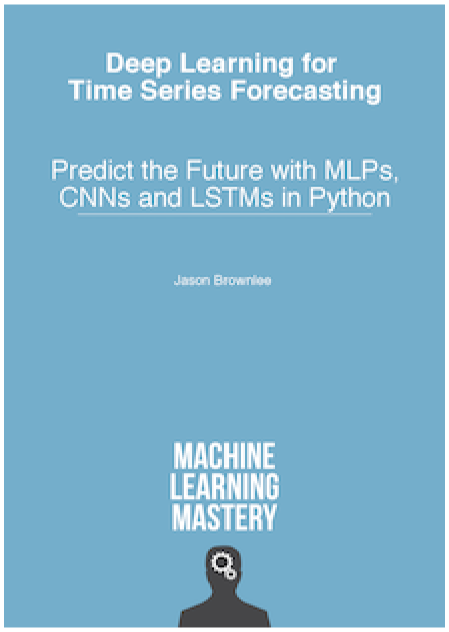
Read More

More Tutorials?

Start Here
(over 780 tutorials!)

Deep Learning for Time Series
Forecast the future with MLPs, CNNs, and LSTMs.

Click to Get Started Now!



Join Newsletter

Subscribe Now

(join over 150,000 readers!)