# How to Use Timesteps in LSTM Networks for Time Series Forecasting

by **Jason Brownlee** on April 17, 2017 in **Long Short-Term Memory Networks**

The Long Short-Term Memory (LSTM) network in Keras supports time steps.

This raises the question as to whether lag observations for a univariate time series can be used as time steps for an LSTM and whether or not this improves forecast performance.

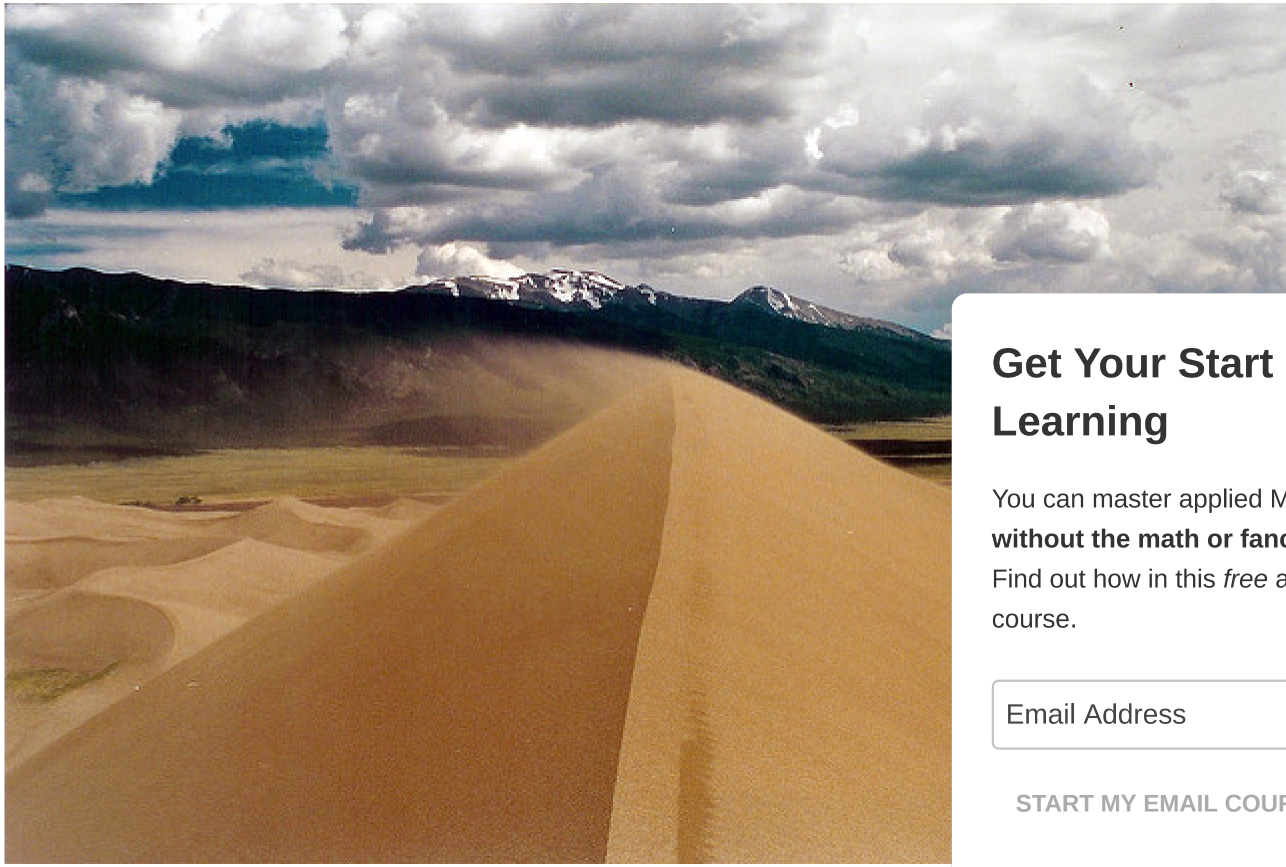
In this tutorial, we will investigate the use of lag observations as time steps in LSTMs models in Python.

After completing this tutorial, you will know:

- How to develop a test harness to systematically evaluate LSTM time steps for time series foreca

- The impact of using a varied number of lagged observations as input time steps for LSTM models.
- The impact of using a varied number of lagged observations and matching numbers of neurons for LSTM models.

Let’s get started.





How to Use Timesteps in LSTM Networks for Time Series Forecasting
Photo by YoTuT, some rights reserved.

# Tutorial Overview

This tutorial is divided into 4 parts. They are:

1. Shampoo Sales Dataset

2. Experimental Test Harness
3. Experiments with Time Steps
4. Experiments with Time Steps and Neurons

## Environment

This tutorial assumes you have a Python SciPy environment installed. You can use either Python 2 or 3 with this example.

This tutorial assumes you have Keras v2.0 or higher installed with either the TensorFlow or Theano backend.

This tutorial also assumes you have scikit-learn, Pandas, NumPy, and Matplotlib installed.

If you need help setting up your Python environment, see this post:

- How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda



# Shampoo Sales Dataset

This dataset describes the monthly number of sales of shampoo over a 3-year period.

The units are a sales count and there are 36 observations. The original dataset is credited to Makrid

You can download and learn more about the dataset here.

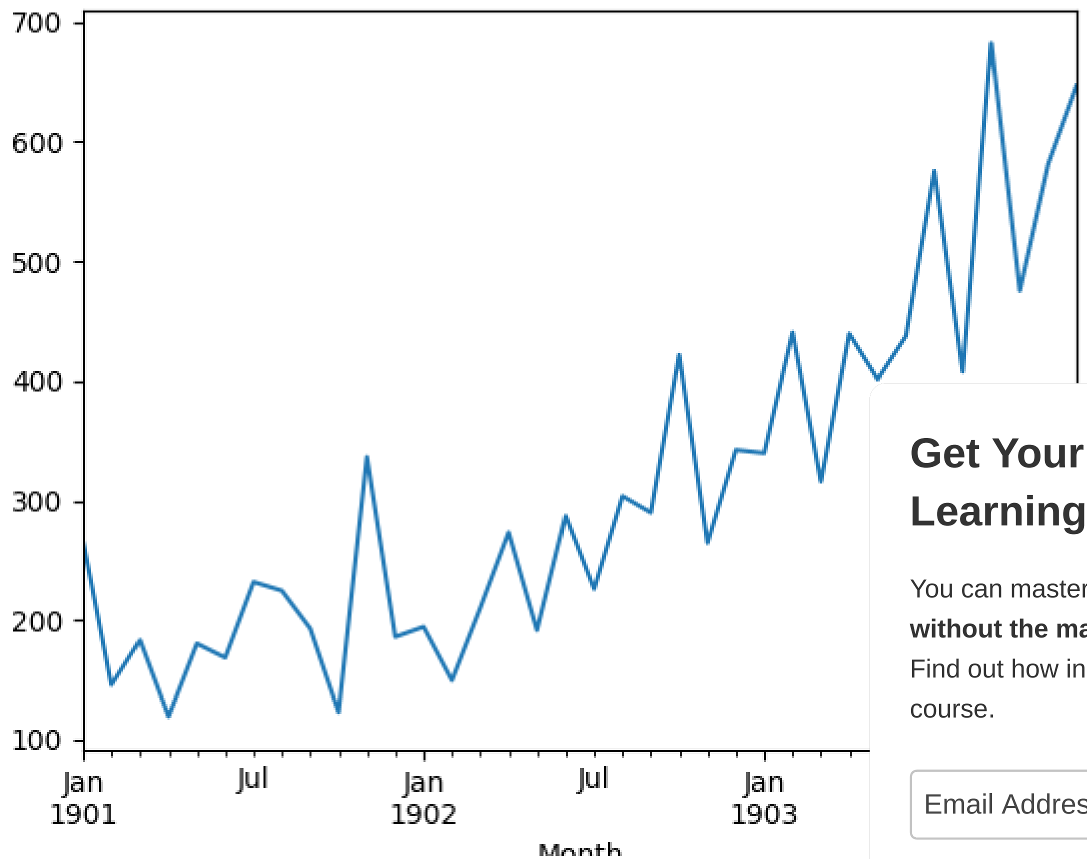
The example below loads and creates a plot of the loaded dataset.

```
# load and plot dataset
from pandas import read_csv
from pandas import datetime
from matplotlib import pyplot
# load dataset
def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')
series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, squeeze=True, date_parser=parser)
# summarize first few rows
print(series.head())
# line plot
series.plot()
pyplot.show()
```

Running the example loads the dataset as a Pandas Series and prints the first 5 rows.

```
Month
1901-01-01 266.0
1901-02-01 145.9
1901-03-01 183.1
1901-04-01 119.3
1901-05-01 180.3
Name: Sales, dtype: float64
```

A line plot of the series is then created showing a clear increasing trend.




Line Plot of Shampoo Sales Dataset



Next, we will take a look at the LSTM configuration and test harness used in the experiment.

## Experimental Test Harness

This section describes the test harness used in this tutorial.

### Data Split

[redacted]

We will split the Shampoo Sales dataset into two parts: a training and a test set.

The first two years of data will be taken for the training dataset and the remaining one year of data will be used for the test set.

Models will be developed using the training dataset and will make predictions on the test dataset.

The persistence forecast (naive forecast) on the test dataset achieves an error of 136.761 monthly shampoo sales. This provides a lower acceptable bound of performance on the test set.

## Model Evaluation

A rolling-forecast scenario will be used, also called walk-forward model validation.

Each time step of the test dataset will be walked one at a time. A model will be used to make a forec ue from the test set will be taken and made available to the model for the forecast on the next time step

This mimics a real-world scenario where new Shampoo Sales observations would be available each g month.

This will be simulated by the structure of the train and test datasets.

All forecasts on the test dataset will be collected and an error score calculated to summarize the skil E) will be used as it punishes large errors and results in a score that is in the same units as the forecas



## Data Preparation

Before we can fit an LSTM model to the dataset, we must transform the data.

The following three data transforms are performed on the dataset prior to fitting a model and making a forecast.

1. **Transform the time series data so that it is stationary**. Specifically, a lag=1 differencing to remove the increasing trend in the data.
2. **Transform the time series into a supervised learning problem**. Specifically, the organization of data into input and output patterns where the observation at the previous time step is used as an input to forecast the observation at the current time timestep

3. **Transform the observations to have a specific scale**. Specifically, to rescale the data to valu[redacted] tangent activation function of the LSTM model.

These transforms are inverted on forecasts to return them into their original scale before calculating and error score.

### LSTM Model

We will use a base stateful LSTM model with 1 neuron fit for 500 epochs.

A batch size of 1 is required as we will be using walk-forward validation and making one-step forecasts for each of the final 12 months of test data.

A batch size of 1 means that the model will be fit using online training (as opposed to batch training or mini-batch training). As a result, it is expected that the model fit will have some variance.

Ideally, more training epochs would be used (such as 1000 or 1500), but this was truncated to 500 to

The model will be fit using the efficient ADAM optimization algorithm and the mean squared error los

### Experimental Runs

Each experimental scenario will be run 10 times.

The reason for this is that the random initial conditions for an LSTM network can result in very differe ed.

Let’s dive into the experiments.

## Experiments with Time Steps

We will perform 5 experiments, each will use a different number of lag observations as time steps from 1 to 5.

A representation with 1 time step would be the default representation when using a stateful LSTM. Using 2 to 5 timesteps is contrived. The hope would be that the additional context from the lagged observations may improve the performance of the predictive model.

The univariate time series is converted to a supervised learning problem before training the model. T number of input variables (*X*) used to predict the next time step (*y*). As such, for each time step used removed from the beginning of the dataset. This is because there are no prior observations to use as [redacted]

The complete code listing for testing 1 time step is listed below.

The time steps parameter in the *run()* function is varied from 1 to 5 for each of the 5 experiments. In addition, the results are saved to file at the end of the experiment and this filename must also be changed for each different experimental run; e.g.: *experiment_timesteps_1.csv*, *experiment_timesteps_2.csv*, etc.

```
from pandas import DataFrame
from pandas import Series
from pandas import concat
from pandas import read_csv
from pandas import datetime
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from math import sqrt
import matplotlib
import numpy
from numpy import concatenate

# date-time parsing function for loading the dataset
def parser(x):
    return datetime.strptime('190'+x, '%Y-%m')

# frame a sequence as a supervised learning problem
def timeseries_to_supervised(data, lag=1):
    df = DataFrame(data)
    columns = [df.shift(i) for i in range(1, lag+1)]
    columns.append(df)
    df = concat(columns, axis=1)
    return df

# create a differenced series
def difference(dataset, interval=1):
    diff = list()
    for i in range(interval, len(dataset)):
        value = dataset[i] - dataset[i - interval]
        diff.append(value)
    return Series(diff)

# invert differenced value
```

```
def inverse_difference(history, yhat, interval=1):
    return yhat + history[-interval]

# scale train and test data to [-1, 1]
def scale(train, test):
    # fit scaler
    scaler = MinMaxScaler(feature_range=(-1, 1))
    scaler = scaler.fit(train)
    # transform train
    train = train.reshape(train.shape[0], train.shape[1])
    train_scaled = scaler.transform(train)
    # transform test
    test = test.reshape(test.shape[0], test.shape[1])
    test_scaled = scaler.transform(test)
    return scaler, train_scaled, test_scaled

# inverse scaling for a forecasted value
def invert_scale(scaler, X, yhat):
    new_row = [x for x in X] + [yhat]
    array = numpy.array(new_row)
    array = array.reshape(1, len(array))
    inverted = scaler.inverse_transform(array)
    return inverted[0, -1]

# fit an LSTM network to training data
def fit_lstm(train, batch_size, nb_epoch, neurons, timesteps):
    X, y = train[:, 0:-1], train[:, -1]
    X = X.reshape(X.shape[0], timesteps, 1)
    model = Sequential()
    model.add(LSTM(neurons, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stc
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    for i in range(nb_epoch):
        model.fit(X, y, epochs=1, batch_size=batch_size, verbose=0, shuffle=False)
        model.reset_states()
    return model

# make a one-step forecast
def forecast_lstm(model, batch_size, X):
    X = X.reshape(1, len(X), 1)
    yhat = model.predict(X, batch_size=batch_size)
    return yhat[0,0]

# run a repeated experiment
def experiment(repeats, series, timesteps):
    # transform data to be stationary
    raw_values = series.values
```

```
    diff_values = difference(raw_values, 1)
    # transform data to be supervised learning
    supervised = timeseries_to_supervised(diff_values, timesteps)
    supervised_values = supervised.values[timesteps:,:]
    # split data into train and test-sets
    train, test = supervised_values[0:-12, :], supervised_values[-12:, :]
    # transform the scale of the data
    scaler, train_scaled, test_scaled = scale(train, test)
    # run experiment
    error_scores = list()
    for r in range(repeats):
        # fit the base model
        lstm_model = fit_lstm(train_scaled, 1, 500, 1, timesteps)
        # forecast test dataset
        predictions = list()
        for i in range(len(test_scaled)):
            # predict
            X, y = test_scaled[i, 0:-1], test_scaled[i, -1]
            yhat = forecast_lstm(lstm_model, 1, X)
            # invert scaling
            yhat = invert_scale(scaler, X, yhat)
            # invert differencing
            yhat = inverse_difference(raw_values, yhat, len(test_scaled)+1-i)
            # store forecast
            predictions.append(yhat)
        # report performance
        rmse = sqrt(mean_squared_error(raw_values[-12:], predictions))
        print('%d) Test RMSE: %.3f' % (r+1, rmse))
        error_scores.append(rmse)
    return error_scores

# execute the experiment
def run():
    # load dataset
    series = read_csv('shampoo-sales.csv', header=0, parse_dates=[0], index_col=0, sque
    # experiment
    repeats = 10
    results = DataFrame()
    # run experiment
    timesteps = 1
    results['results'] = experiment(repeats, series, timesteps)
    # summarize results
    print(results.describe())
    # save results
    results.to_csv('experiment_timesteps_1.csv', index=False)

# entry point
```

```
run()
```

Run the 5 different experiments for the 5 different numbers of time steps.

You can run them in parallel if you have sufficient memory and CPU resources. GPU resources are not required for these experiments and experiments should be complete in minutes to tens of minutes.

After running the experiments, you should have 5 files containing the results, as follows:

```
experiment_timesteps_1.csv
experiment_timesteps_2.csv
experiment_timesteps_3.csv
experiment_timesteps_4.csv
experiment_timesteps_5.csv
```

We can write some code to load and summarize these results.

Specifically, it is useful to review both descriptive statistics from each run and compare the results fo

Code to summarize the results is listed below.

```
from pandas import DataFrame
from pandas import read_csv
from matplotlib import pyplot
# load results into a dataframe
filenames = ['experiment_timesteps_1.csv', 'experiment_timesteps_2.csv',
    'experiment_timesteps_3.csv','experiment_timesteps_4.csv','experiment_timesteps_5.cs
results = DataFrame()
for name in filenames:
    results[name[11:-4]] = read_csv(name, header=0)
# describe all results
print(results.describe())
# box and whisker plot
results.boxplot()
pyplot.show()
```



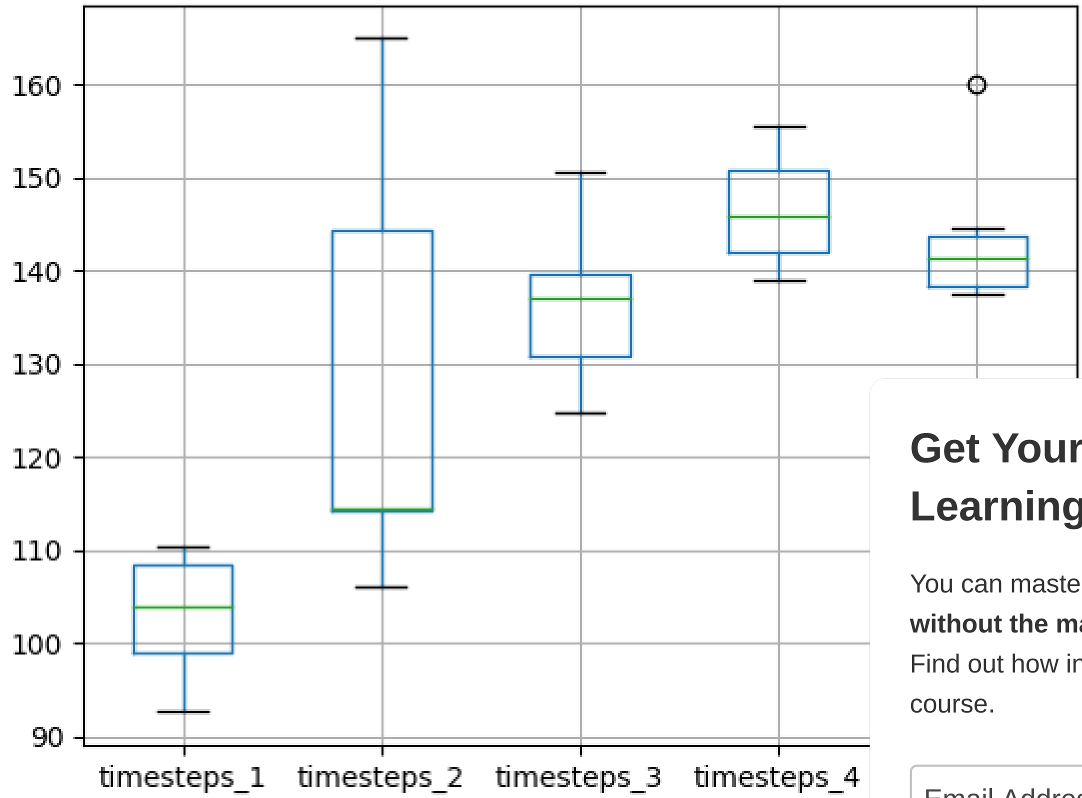
Running the code first prints descriptive statistics for each set of results.

We can see from the average performance alone that the default of using a single time step resulted in the best performance. This is also shown when reviewing the median test RMSE (50th percentile).

```
       timesteps_1  timesteps_2  timesteps_3  timesteps_4  timesteps_5
count    10.000000    10.000000    10.000000    10.000000    10.000000
mean    102.785197   127.308725   136.182907   146.277122   142.631684
std       6.299329    22.171668     7.760161     5.609412     6.611638
min      92.603903   106.124901   124.724903   138.845314   137.359503
25%      98.979692   114.100891   130.719154   141.906083   138.354265
50%     103.904185   114.519986   137.055840   145.865171   141.409855
75%     108.434727   144.328534   139.615541   150.729938   143.604275
max     110.270559   164.880226   150.497130   155.603461   159.948033
```

A box and whisker plot comparing the distributions of results is also created.

The plot tells the same story as the descriptive statistics. There is a general trend of increasing test RMSE as the number of time steps is increased.




Box and Whisker Plot of Timesteps vs RMSE



The expectation of increased performance with the increase of time steps was not observed, at least with the dataset and LSTM configuration used.

This raises the question as to whether the capacity of the network is a limiting factor. We will look at this in the next section.

## Experiments with Time Steps and Neurons

The number of neurons (also called blocks) in the LSTM network defines its learning capacity.

It is possible that in the previous experiments the use of one neuron limited the learning capacity of [redacted] effective use of the lagged observations as time steps.

We can repeat the above experiments and increase the number of neurons in the LSTM with the increase in time steps and see if it results in an increase in performance.

This can be achieved by changing the line in the experiment function from:

```
lstm_model = fit_lstm(train_scaled, 1, 500, 1, timesteps)
```

to

```
lstm_model = fit_lstm(train_scaled, 1, 500, timesteps, timesteps)
```

In addition, we can keep the results written to file separate from the results created in the first experi es, for example, changing:

```
results.to_csv('experiment_timesteps_1.csv', index=False)
```

to

```
results.to_csv('experiment_timesteps_1_neurons.csv', index=False)
```

Repeat the same 5 experiments with these changes.

After running these experiments, you should have 5 result files.

```
experiment_timesteps_1_neurons.csv
experiment_timesteps_2_neurons.csv
experiment_timesteps_3_neurons.csv
experiment_timesteps_4_neurons.csv
experiment_timesteps_5_neurons.csv
```

As in the previous experiment, we can load the results, calculate descriptive statistics, and create a box and whisker plot. The complete code listing is below.

```
from pandas import DataFrame
from pandas import read_csv
from matplotlib import pyplot
# load results into a dataframe
```

```
filenames = ['experiment_timesteps_1_neurons.csv', 'experiment_timesteps_2_neurons.csv',[redacted]
    'experiment_timesteps_3_neurons.csv','experiment_timesteps_4_neurons.csv','experiment_timesteps_5_neurons.csv']
results = DataFrame()
for name in filenames:
    results[name[11:-12]] = read_csv(name, header=0)
# describe all results
print(results.describe())
# box and whisker plot
results.boxplot()
pyplot.show()
```

Running the code first prints descriptive statistics from each of the 5 experiments.

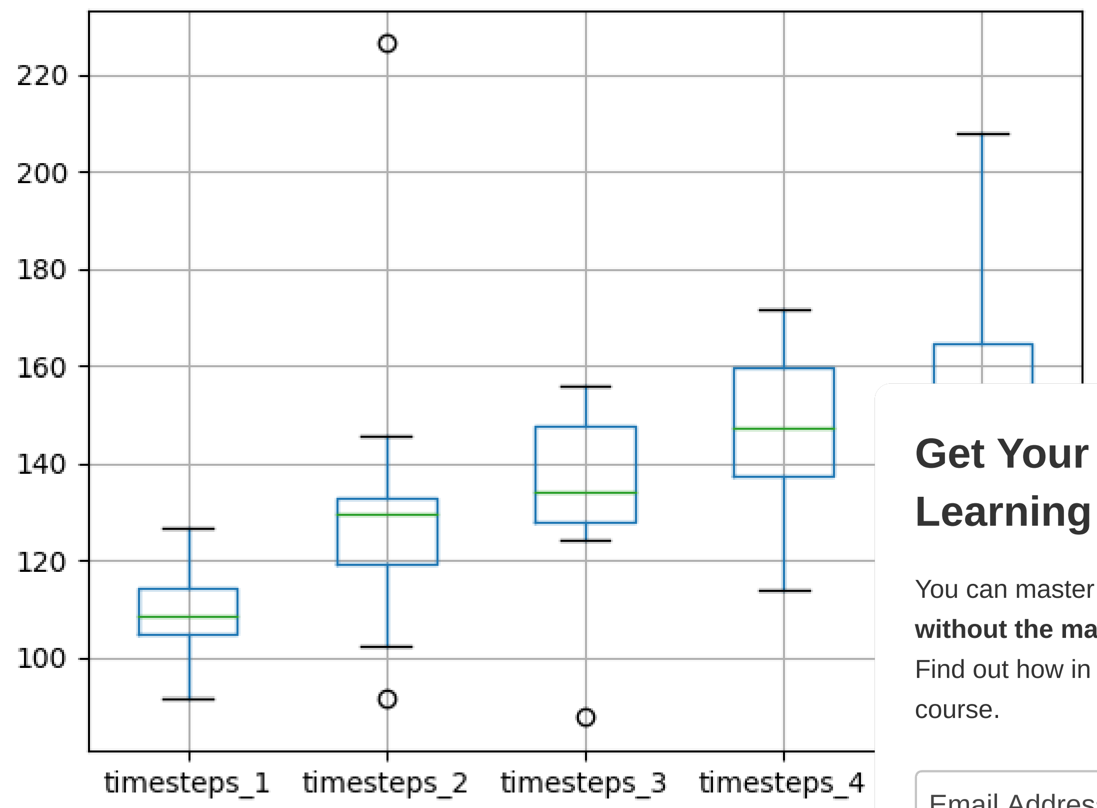
The results tell a similar story to the first set of experiments with a one neuron LSTM. The average test RMSE appears lowest when the number of neurons and the number of time steps is set to one.

```
       timesteps_1  timesteps_2  timesteps_3  timesteps_4  timesteps_5
count    10.000000    10.000000    10.000000    10.000000    10.000000
mean    109.484374   133.195856   133.432933   145.843701   149.854229
std       9.663732    36.328757    19.347675    19.389278    30.194324
min      91.803241    91.791014    87.739484   113.808683   103.612424
25%     104.757265   119.269854   127.937277   137.417983   131.278548
50%     108.464050   129.775765   134.076721   147.222168   151.999097
75%     114.265381   132.796259   147.557091   159.518828   164.741625
max     126.581011   226.396127   156.019616   171.570206   208.030615
```

A box and whisker plot is created to compare the distributions.

The trend in spread and median performance almost shows a linear increase in test RMSE as the nu

The linear trend may suggest that the increase in network capacity is not given sufficient time to fit th epochs would be required as well.




Box and Whisker Plot of Timesteps and Neurons vs RMSE



## Extensions

This section lists some areas for further investigation that you may consider exploring.

- **Lags as Features**. The use of lagged observations as time steps also raises the question as to whether lagged observations can be used as input features. It is not clear whether time steps and features are treated the same way internally by the Keras LSTM implementation.
- **Diagnostic Run Plots**. It may be helpful to review plots of train and test RMSE over epochs for multiple runs for a given experiment. This might help tease out whether overfitting or underfitting is taking place, and in turn, methods to address it.

- **Increase Training Epochs**. An increase in neurons in the LSTM in the second set of experimen[redacted] training epochs. This could be explored with some follow-up experiments.
- **Increase Repeats**. Using 10 repeats results in a relatively small population of test RMSE results. It is possible that increasing repeats to 30 or 100 (or even higher) may result in a more stable outcome.

Did you explore any of these extensions?
Share your findings in the comments below; I'd love to hear what you found.

## Summary

In this tutorial, you discovered how to investigate using lagged observations as input time steps in an LSTM network.

Specifically, you learned:

- How to develop a robust test harness for experimenting with input representation with LSTMs.
- How to use lagged observations as input time steps for time series forecasting with LSTMs.
- How to increase the learning capacity of the network with the increase of time steps.

You discovered that the expectation that the use of lagged observations as input time steps did not c nd LSTM configuration.

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



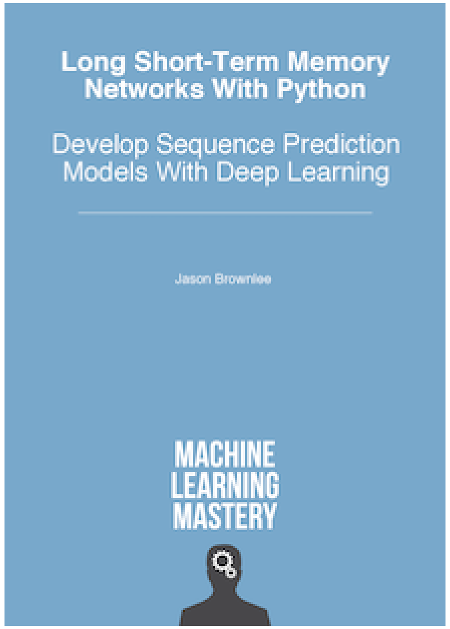
---

## Develop LSTMs for Sequence Prediction Today!

### Develop Your Own LSTM models in Minutes

…with just a few lines of python code

Discover how in my new Ebook:
Long Short-Term Memory Networks with Python




It provides **self-study tutorials** [redacted]
*CNN LSTMs, Encoder-Decoder LSTMs, generative models, data preparation, making predictions* and much more…

**Finally Bring LSTM Recurrent Neural Networks to Your Sequence Predictions Projects**

Skip the Academics. Just Results.

Click to learn more.



### About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional devel
to helping developers get started and get good at applied machine learning. Learn more.
View all posts by Jason Brownlee →

< How to Update LSTM Networks During Training for Time Series Forecasting

How to Use Features in LSTM Networks for Time Series Forecasting >

Get Your Start in Machine Learning

## 28 Responses to *How to Use Timesteps in LSTM Networks for Time Series Forecasting*

**Hasan** April 17, 2017 at 12:29 pm # REPLY

The problem with using lagged values as predictors is that the model misses out the subtle time dependencies which are usually captured by the time series models.

**Jason Brownlee** April 18, 2017 at 8:29 am #

Agreed. The promise of LSTMS is to learn the temporal dependence.

**Hasan** April 19, 2017 at 7:52 pm #

So LSTM will work for all kinds of time series?

**Jason Brownlee** April 20, 2017 at 9:24 am #

Yes, but test other methods and double down on what works best on your problem.



**Kunpeng Zhang** April 18, 2017 at 1:02 pm # REPLY

Hi Jason,
Your posts are always helpful.
Now, I get two similar data sets. I'd like to train this data using multitask model in keras. To be percise, I

separately in one train model.
Is it possible in keras? I get some content. https://keras.io/getting-started/functional-api-guide/
But I still do not figure it out how. Could you give me some advice?

**Jason Brownlee** April 19, 2017 at 7:49 am # REPLY

Almost all neural nets can have multiple output values.

Just frame your dataset and set the number of outputs you require in the output layer of the network.

**Kunpeng Zhang** April 18, 2017 at 1:06 pm #

Another question. Compared with tensorflow, a fine-tuned keras model will get a better result or

**Jason Brownlee** April 19, 2017 at 7:50 am #

Keras is built on top of TensorFlow. Comparing results from the two does not make sense (

**Kunpeng Zhang** April 20, 2017 at 10:25 am #

Thank you for your reply.
Have a good day.

**Jack Brown** April 18, 2017 at 9:06 pm # REPLY

Hi Jason,
could you elaborate this line

train = train.reshape(train.shape[0], train.shape[1])

isn't this the same?

---

**Jason Brownlee** April 19, 2017 at 7:52 am #

REPLY

It does look that way, I may have been too excited with all the resizing. Try removing it and see if all is well.

**Jay Reynolds** May 26, 2017 at 11:26 am #

"Lags as Features. The use of lagged observations as time steps also raises the question as to
features. It is not clear whether time steps and features are treated the same way internally by the Keras

Any further thoughts on this?
I'm a little confused on how to use timesteps when some input features are lagged and some are not. (re
exists at all, given that it would seem any lagged input should just be treated as features). There's surpri
timesteps on the internet… I don't recall ever coming across the concept of timesteps in any of Schmidh
attention!)

Thanks for the great resource you've put together and continue to share, btw.



---

**Jason Brownlee** June 2, 2017 at 11:52 am #

REPLY

Yes, I was wrong.

Features are weighted inputs. Timesteps are discrete inputs of features over time. (does that make sense, it reads poorly…)

The key to understanding timesteps is the BPTT algorithm. I have a post on this scheduled.

**John Jaro** July 2, 2017 at 1:27 am #

REPLY

"I'm a little confused on how to use timesteps when some input features are lagged and some are not. (really, I'm fundamentally confused as to why timesteps exists at all, given that it would seem any lagged input should just be treated as features). There's surprisingly little clear information on the matter of LSTM timesteps on the internet…"

This is 100% my question, I've done so much Googling (and read multiple of Jason's posts) and I still don't understand this at all. Cannot figure out how to prep lagged time steps + features for LSTM.

**Jason Brownlee** July 2, 2017 at 6:33 am #

Lagged obs are time steps in LSTMs.

LSTM input is 3d: [samples, time steps, features]. If your series is univariate, you one many time
day of data, you have one sample, 25 hours of time steps and one feature.

Does that help?



**lawrance** May 27, 2017 at 6:50 pm #

In your previous blog(http://machinelearningmastery.com/time-series-prediction-lstm-recurrent-n
you use "trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))"(1),
and now you use "X = X.reshape(X.shape[0], timesteps, 1)" (2).
If the second parameter means timestamp. then in (1), you may use "look_back" in that article instead of 1
If the third parameter means one var, then in (1), you may use 1 instead of trainX.shape[1], because trainX.shape[1] means look_back or timesteps in this article.

**Jason Brownlee** June 2, 2017 at 12:01 pm #

I would recommend using past observations as timesteps when inputting to the model.

**Birkey** June 2, 2017 at 2:22 pm #

REPLY

Could it be overfitting with more neurons? since more neurons means more degrees of freedom, so the model can (over) fit the training data well, while generalize poorly.

If that's the case, more epochs won't help though, we need more training data.

**Jason Brownlee** June 3, 2017 at 7:19 am #

Yes, more neurons can result in overfitting.

**Roger** July 9, 2017 at 1:16 am #

Hi Jason – thank you for the great content. Really enjoyed your ML Python recipes. I am having t data for the LSTM, since everywhere I look seems to suggest something different.

I understand that the input X has the shape (samples, timesteps, features). My use case is I have about features to forecast another time series 5 steps ahead at a time (updating as new information in the rollir What will the structure of X look like in my case? I currently have something like this:

X Y
[[t0, t1, t2], [[t3, t4]
[t1, t2, t3], [t4, t5]
… …

for each feature, which I've then stacked together into a 3D shape using np.stack( ). But it seems like this is incorrect, since the timesteps should be 2, not 3? Am I coming at this the right way? The timestep/feature/lag confusion seems to be prevalent on the Internet. Also each feature might have greater predictive power at different lag/leads, will this LSTM setup potentially bottleneck my accuracy, and is there a bette

**Jason Brownlee** July 9, 2017 at 10:55 am # REPLY

If you have 5 series then that would be 5 features.

I would recommend loading the data as a 2d matrix then using reshape, perhaps with 1 sample.

Does that help?

**Nihit** August 8, 2017 at 8:35 pm # REPLY

Hi Jason, great post.
I have been trying to implement Keras LSTM using R. How can I reshape my univariate data frame to th



**Jason Brownlee** August 9, 2017 at 6:28 am #

Sorry, I don't have material on using Keras in R.

**Nihit** August 9, 2017 at 3:48 pm #

Ohh that's unfortunate. Although I did find reshape layer in keras, but I am not sure if it
Also when i used it to train a model, it converted the train set into 3D array but now i cannot evaluate the model since I am stuck on trying to convert test set in to 3D array. Thanks.

**Jason Brownlee** August 10, 2017 at 6:51 am # REPLY

What error are you getting?

**Nihit** August 10, 2017 at 5:05 pm # REPLY

I was able to fix the problem by reshaping the train set to 3D array with timesteps = 1 and including lagged values as input.
But I cannot set timesteps more than 1.
E.g. I have a dataset with time interval of every 15 mins. If I set timestep to 96(1 Day) and built a LSTM model then I cannot forecast on test(1 Month) set since I get only (2880/96 = ) 30 values and not 2880 values.

**Pablios** September 22, 2017 at 6:38 pm # REPLY

hey thank you very much for this posts !!
they are really useful

**Jason Brownlee** September 23, 2017 at 5:37 am #

I'm glad to hear that.

## Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

### Welcome to Machine Learning Mastery

Hi, I'm Dr. Jason Brownlee.
My goal is to make practitioners like YOU awesome at applied machine learning.

Read More

### Deep Learning for Sequence Prediction

Cut through the math and research papers.
Discover 4 Models, 6 Architectures, and 14 Tutorials.



Get Your Start in Machine Learning

POPULAR


**Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras**
JULY 21, 2016


**Your First Machine Learning Project in Python Step-By-Step**
JUNE 10, 2016


**Develop Your First Neural Network in Python With Keras Step-By-Step**
MAY 24, 2016

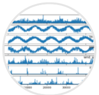

**Sequence Classification with LSTM Recurrent Neural Networks in Python with Keras**
JULY 26, 2016

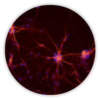
**How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda**
MARCH 13, 2017


**Time Series Forecasting with the Long Short-Term Memory Network in Python**
APRIL 7, 2017


**Multi-Class Classification Tutorial with the Keras Deep Learning Library**
JUNE 2, 2016


**Regression Tutorial with the Keras Deep Learning Library in Python**
JUNE 9, 2016


**Multivariate Time Series Forecasting with LSTMs in Keras**
AUGUST 14, 2017


**How to Implement the Backpropagation Algorithm From Scratch In Python**
NOVEMBER 7, 2016



Privacy | Contact | About



Get Your Start in Machine Learning