Need help with Deep Learning? Take the FREE Mini-Course

Understanding Stateful LSTM Recurrent Neural Networks in Python with Keras

by **Jason Brownlee** on July 28, 2016 in **Deep Learning**

A powerful and popular recurrent neural network is the long short-term model network or LSTM.

It is widely used because the architecture overcomes the vanishing and exposing gradient problem that plagues all recurrent neural networks, allowing very large and very deep networks to be created.

Like other recurrent neural networks, LSTM networks maintain state, and the specifics of how this is implemented in Keras framework can be confusing.

In this post you will discover exactly how state is maintained in LSTM networks by the Keras deep le

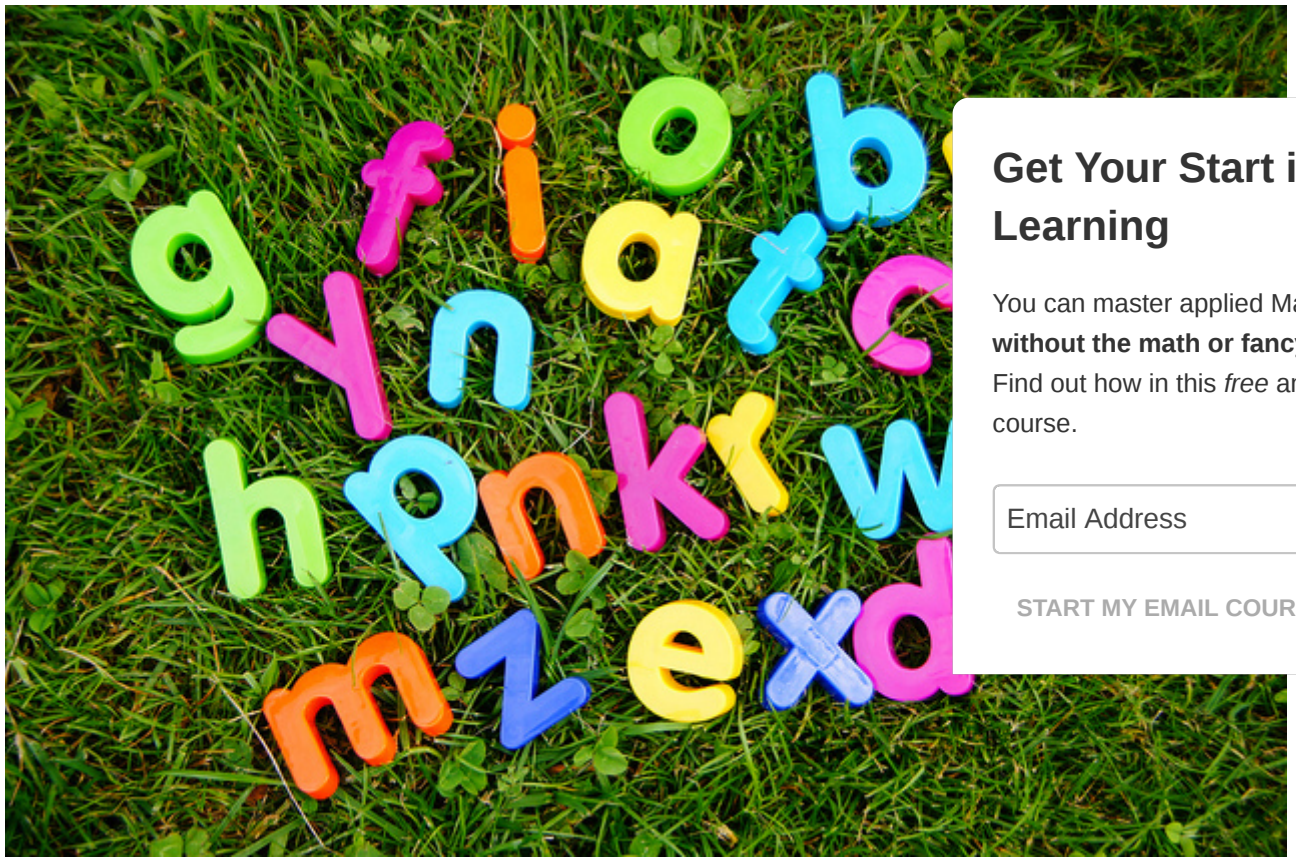
Get Your Start in Machine Learning

After reading this post you will know:

- How to develop a naive LSTM network for a sequence prediction problem.
- How to carefully manage state through batches and features with an LSTM network.
- Hot to manually manage state in an LSTM network for stateful prediction.

Let's get started.

- **Update Mar/2017**: Updated example for Keras 2.0.2, TensorFlow 1.0.1 and Theano 0.9.0.



Understanding Stateful LSTM Recurrent Neural Networks in Python with Keras
Photo by Martin Abegglen, some rights reserved.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Problem Description: Learn the Alphabet

In this tutorial we are going to develop and contrast a number of different LSTM recurrent neural network models.

The context of these comparisons will be a simple sequence prediction problem of learning the alphabet. That is, given a letter of the alphabet, predict the next letter of the alphabet.

This is a simple sequence prediction problem that once understood can be generalized to other sequence prediction problems like time series prediction and sequence classification.

Let's prepare the problem with some python code that we can reuse from example to example.

Firstly, let's import all of the classes and functions we plan to use in this tutorial.

```
1 import numpy
2 from keras.models import Sequential
3 from keras.layers import Dense
4 from keras.layers import LSTM
5 from keras.utils import np_utils
```

Next, we can seed the random number generator to ensure that the results are the same each time

```
1 # fix random seed for reproducibility
2 numpy.random.seed(7)
```

We can now define our dataset, the alphabet. We define the alphabet in uppercase characters for re

Neural networks model numbers, so we need to map the letters of the alphabet to integer values. W of the letter index to the character. We can also create a reverse lookup for converting predictions back into characters to be used later.

```
1 # define the raw dataset
2 alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3 # create mapping of characters to integers (0-25) and the reverse
4 char_to_int = dict((c, i) for i, c in enumerate(alphabet))
5 int_to_char = dict((i, c) for i, c in enumerate(alphabet))
```

Now we need to create our input and output pairs on which to train our neural network. We can do this by defining an input sequence length, then reading sequences from the input alphabet sequence.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

For example we use an input length of 1. Starting at the beginning of the raw input data, we can read off the first letter "A" and the next letter as the prediction "B". We move along one character and repeat until we reach a prediction of "Z".

```
# prepare the dataset of input to output pairs encoded as integers
seq_length = 1
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print seq_in, '->', seq_out
```

We also print out the input pairs for sanity checking.

Running the code to this point will produce the following output, summarizing input sequences of len

```
A -> B
B -> C
C -> D
D -> E
E -> F
F -> G
G -> H
H -> I
I -> J
J -> K
K -> L
L -> M
M -> N
N -> O
O -> P
P -> Q
Q -> R
R -> S
S -> T
T -> U
U -> V
V -> W
W -> X
X -> Y
Y -> Z
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

We need to reshape the NumPy array into a format expected by the LSTM networks, that is [*samples, time steps, features*].

```
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
```

Once reshaped, we can then normalize the input integers to the range 0-to-1, the range of the sigmoid activation functions used by the LSTM network.

```
# normalize
X = X / float(len(alphabet))
```

Finally, we can think of this problem as a sequence classification task, where each of the 26 letters represents a different class. As such, we can convert the output (y) to a one hot encoding, using the Keras built-in function **to_categorical()**.

```
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
```

We are now ready to fit different LSTM models.

Need help with Deep Learning in Pyth

Take my free 2-week email course and discover MLPs, CNNs and LST

Click to sign-up now and also get a free PDF Ebook version

Start Your FREE Mini-Course Now!

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Naive LSTM for Learning One-Char to One-Char Mapping

Let's start off by designing a simple LSTM to learn how to predict the next character in the alphabet given the context of just one character.

We will frame the problem as a random collection of one-letter input to one-letter output pairs. As we will see this is a difficult framing of the problem for the LSTM to learn.

Get Your Start in Machine Learning

Let's define an LSTM network with 32 units and an output layer with a softmax activation function for making predictions. Because this is a multi-class classification problem, we can use the log loss function (called "**categorical_crossentropy**" in Keras), and optimize the network using the ADAM optimization function.

The model is fit over 500 epochs with a batch size of 1.

```
# create and fit the model
model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=500, batch_size=1, verbose=2)
```

After we fit the model we can evaluate and summarize the performance on the entire training dataset.

```
# summarize performance of the model
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
```

We can then re-run the training data through the network and generate predictions, converting both t
character format to get a visual idea of how well the network learned the problem.

```
# demonstrate some model predictions
for pattern in dataX:
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
```

The entire code listing is provided below for completeness.

```
# Naive LSTM to learn one-char to one-char mapping
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
# fix random seed for reproducibility
numpy.random.seed(7)
# define the raw dataset
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# prepare the dataset of input to output pairs encoded as integers
seq_length = 1
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print seq_in, '->', seq_out
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
# normalize
X = X / float(len(alphabet))
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# create and fit the model
model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=500, batch_size=1, verbose=2)
# summarize performance of the model
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
# demonstrate some model predictions
for pattern in dataX:
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Running this example produces the following output.

```
Model Accuracy: 84.00%
['A'] -> B
['B'] -> C
['C'] -> D
['D'] -> E
['E'] -> F
```

Get Your Start in Machine Learning

```
['F'] -> G
['G'] -> H
['H'] -> I
['I'] -> J
['J'] -> K
['K'] -> L
['L'] -> M
['M'] -> N
['N'] -> O
['O'] -> P
['P'] -> Q
['Q'] -> R
['R'] -> S
['S'] -> T
['T'] -> U
['U'] -> W
['V'] -> Y
['W'] -> Z
['X'] -> Z
['Y'] -> Z
```

We can see that this problem is indeed difficult for the network to learn.

The reason is, the poor LSTM units do not have any context to work with. Each input-output pattern state of the network is reset after each pattern (each batch where each batch contains one pattern).

This is abuse of the LSTM network architecture, treating it like a standard multilayer Perceptron.

Next, let's try a different framing of the problem in order to provide more sequence to the network fro

Naive LSTM for a Three-Char Feature Window to One-Cha

A popular approach to adding more context to data for multilayer Perceptrons is to use the window method.

This is where previous steps in the sequence are provided as additional input features to the network. We can try the same trick to provide more context to the LSTM network.

Here, we increase the sequence length from 1 to 3, for example:

```
# prepare the dataset of input to output pairs encoded as integers
seq_length = 3
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Which creates training patterns like:

```
ABC -> D
BCD -> E
CDE -> F
```

Each element in the sequence is then provided as a new input feature to the network. This requires a modification of how the input sequences reshaped in the data preparation step:

```
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), 1, seq_length))
```

It also requires a modification for how the sample patterns are reshaped when demonstrating predictions from the model.

```
x = numpy.reshape(pattern, (1, 1, len(pattern)))
```

The entire code listing is provided below for completeness.

```
# Naive LSTM to learn three-char window to one-char mapping
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
# fix random seed for reproducibility
numpy.random.seed(7)
# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# prepare the dataset of input to output pairs encoded as integers
seq_length = 3
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print seq_in, '->', seq_out
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), 1, seq_length))
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
# normalize
X = X / float(len(alphabet))
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# create and fit the model
model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=500, batch_size=1, verbose=2)
# summarize performance of the model
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
# demonstrate some model predictions
for pattern in dataX:
    x = numpy.reshape(pattern, (1, 1, len(pattern)))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
```

Running this example provides the following output.

```
Model Accuracy: 86.96%
['A', 'B', 'C'] -> D
['B', 'C', 'D'] -> E
['C', 'D', 'E'] -> F
['D', 'E', 'F'] -> G
['E', 'F', 'G'] -> H
['F', 'G', 'H'] -> I
['G', 'H', 'I'] -> J
['H', 'I', 'J'] -> K
['I', 'J', 'K'] -> L
['J', 'K', 'L'] -> M
['K', 'L', 'M'] -> N
['L', 'M', 'N'] -> O
['M', 'N', 'O'] -> P
['N', 'O', 'P'] -> Q
['O', 'P', 'Q'] -> R
['P', 'Q', 'R'] -> S
['Q', 'R', 'S'] -> T
['R', 'S', 'T'] -> U
['S', 'T', 'U'] -> V
['T', 'U', 'V'] -> Y
['U', 'V', 'W'] -> Z
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
['V', 'W', 'X'] -> Z
['W', 'X', 'Y'] -> Z
```

We can see a small lift in performance that may or may not be real. This is a simple problem that we were still not able to learn with LSTMs even with the window method.

Again, this is a misuse of the LSTM network by a poor framing of the problem. Indeed, the sequences of letters are time steps of one feature rather than one time step of separate features. We have given more context to the network, but not more sequence as it expected.

In the next section, we will give more context to the network in the form of time steps.

Naive LSTM for a Three-Char Time Step Window to One-Char Mapping

In Keras, the intended use of LSTMs is to provide context in the form of time steps, rather than windo

We can take our first example and simply change the sequence length from 1 to 3.

```
seq_length = 3
```

Again, this creates input-output pairs that look like:

```
ABC -> D
BCD -> E
CDE -> F
DEF -> G
```

The difference is that the reshaping of the input data takes the sequence as a time step sequence of multiple features.

```
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
```

This is the correct intended use of providing sequence context to your LSTM in Keras. The full code example is provided below for completeness.

```
# Naive LSTM to learn three-char time steps to one-char mapping
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
# fix random seed for reproducibility
numpy.random.seed(7)
# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# prepare the dataset of input to output pairs encoded as integers
seq_length = 3
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print seq_in, '->', seq_out
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
# normalize
X = X / float(len(alphabet))
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# create and fit the model
model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=500, batch_size=1, verbose=2)
# summarize performance of the model
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
# demonstrate some model predictions
for pattern in dataX:
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Running this example provides the following output.

```
Model Accuracy: 100.00%
['A', 'B', 'C'] -> D
['B', 'C', 'D'] -> E
```

Get Your Start in Machine Learning

```
['C', 'D', 'E'] -> F
['D', 'E', 'F'] -> G
['E', 'F', 'G'] -> H
['F', 'G', 'H'] -> I
['G', 'H', 'I'] -> J
['H', 'I', 'J'] -> K
['I', 'J', 'K'] -> L
['J', 'K', 'L'] -> M
['K', 'L', 'M'] -> N
['L', 'M', 'N'] -> O
['M', 'N', 'O'] -> P
['N', 'O', 'P'] -> Q
['O', 'P', 'Q'] -> R
['P', 'Q', 'R'] -> S
['Q', 'R', 'S'] -> T
['R', 'S', 'T'] -> U
['S', 'T', 'U'] -> V
['T', 'U', 'V'] -> W
['U', 'V', 'W'] -> X
['V', 'W', 'X'] -> Y
['W', 'X', 'Y'] -> Z
```

We can see that the model learns the problem perfectly as evidenced by the model evaluation and th

But it has learned a simpler problem. Specifically, it has learned to predict the next letter from a sequ wn any random sequence of three letters from the alphabet and predict the next letter.

It can not actually enumerate the alphabet. I expect that a larger enough multilayer perception netwo the window method.

The LSTM networks are stateful. They should be able to learn the whole alphabet sequence, but by rk state after each training batch.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

LSTM State Within A Batch

The Keras implementation of LSTMs resets the state of the network after each batch.

This suggests that if we had a batch size large enough to hold all input patterns and if all the input patterns were ordered sequentially, that the LSTM could use the context of the sequence within the batch to better learn the sequence.

Get Your Start in Machine Learning

We can demonstrate this easily by modifying the first example for learning a one-to-one mapping and increasing the batch size from 1 to the size of the training dataset.

Additionally, Keras shuffles the training dataset before each training epoch. To ensure the training data patterns remain sequential, we can disable this shuffling.

```
model.fit(X, y, epochs=5000, batch_size=len(dataX), verbose=2, shuffle=False)
```

The network will learn the mapping of characters using the the within-batch sequence, but this context will not be available to the network when making predictions. We can evaluate both the ability of the network to make predictions randomly and in sequence.

The full code example is provided below for completeness.

```
# Naive LSTM to learn one-char to one-char mapping with all data in each batch
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences
# fix random seed for reproducibility
numpy.random.seed(7)
# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# prepare the dataset of input to output pairs encoded as integers
seq_length = 1
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print seq_in, '->', seq_out
# convert list of lists to array and pad sequences if needed
X = pad_sequences(dataX, maxlen=seq_length, dtype='float32')
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (X.shape[0], seq_length, 1))
# normalize
X = X / float(len(alphabet))
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# create and fit the model
model = Sequential()
model.add(LSTM(16, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=5000, batch_size=len(dataX), verbose=2, shuffle=False)
# summarize performance of the model
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
# demonstrate some model predictions
for pattern in dataX:
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
# demonstrate predicting random patterns
print "Test a Random Pattern:"
for i in range(0,20):
    pattern_index = numpy.random.randint(len(dataX))
    pattern = dataX[pattern_index]
    x = numpy.reshape(pattern, (1, len(pattern), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
```

Running the example provides the following output.

```
Model Accuracy: 100.00%
['A'] -> B
['B'] -> C
['C'] -> D
['D'] -> E
['E'] -> F
['F'] -> G
['G'] -> H
['H'] -> I
['I'] -> J
['J'] -> K
['K'] -> L
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
['L'] -> M
['M'] -> N
['N'] -> O
['O'] -> P
['P'] -> Q
['Q'] -> R
['R'] -> S
['S'] -> T
['T'] -> U
['U'] -> V
['V'] -> W
['W'] -> X
['X'] -> Y
['Y'] -> Z
Test a Random Pattern:
['T'] -> U
['V'] -> W
['M'] -> N
['Q'] -> R
['D'] -> E
['V'] -> W
['T'] -> U
['U'] -> V
['J'] -> K
['F'] -> G
['N'] -> O
['B'] -> C
['M'] -> N
['F'] -> G
['F'] -> G
['P'] -> Q
['A'] -> B
['K'] -> L
['W'] -> X
['E'] -> F
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

As we expected, the network is able to use the within-sequence context to learn the alphabet, achieving 100% accuracy on the training data.

Importantly, the network can make accurate predictions for the next letter in the alphabet for randomly selected characters. Very impressive.

Stateful LSTM for a One-Char to One-Char Mapping

We have seen that we can break-up our raw data into fixed size sequences and that this representation can be learned by the LSTM, but only to learn random mappings of 3 characters to 1 character.

Get Your Start in Machine Learning

We have also seen that we can pervert batch size to offer more sequence to the network, but only during training.

Ideally, we want to expose the network to the entire sequence and let it learn the inter-dependencies, rather than us define those dependencies explicitly in the framing of the problem.

We can do this in Keras by making the LSTM layers stateful and manually resetting the state of the network at the end of the epoch, which is also the end of the training sequence.

This is truly how the LSTM networks are intended to be used. We find that by allowing the network itself to learn the dependencies between the characters, that we need a smaller network (half the number of units) and fewer training epochs (almost half).

We first need to define our LSTM layer as stateful. In so doing, we must explicitly specify the batch size as a dimension on the input shape. This also means that when we evaluate the network or make predictions, we must also specify and adhere to s we are using a batch size of 1. This could introduce difficulties when making predictions when the ba made in batch and in sequence.

```
batch_size = 1
model.add(LSTM(16, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True)
```

An important difference in training the stateful LSTM is that we train it manually one epoch at a time his in a for loop. Again, we do not shuffle the input, preserving the sequence in which the input training

```
for i in range(300):
    model.fit(X, y, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
```

As mentioned, we specify the batch size when evaluating the performance of the network on the ent

```
# summarize performance of the model
scores = model.evaluate(X, y, batch_size=batch_size, verbose=0)
model.reset_states()
print("Model Accuracy: %.2f%%" % (scores[1]*100))
```

Finally, we can demonstrate that the network has indeed learned the entire alphabet. We can seed it with the first letter “A”, request a prediction, feed the prediction back in as an input, and repeat the process all the way to “Z”.

```
# demonstrate some model predictions
seed = [char_to_int[alphabet[0]]]
```

Get Your Start in Machine Learning

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
for i in range(0, len(alphabet)-1):
    x = numpy.reshape(seed, (1, len(seed), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    print int_to_char[seed[0]], "->", int_to_char[index]
    seed = [index]
model.reset_states()
```

We can also see if the network can make predictions starting from an arbitrary letter.

```
# demonstrate a random starting point
letter = "K"
seed = [char_to_int[letter]]
print "New start: ", letter
for i in range(0, 5):
    x = numpy.reshape(seed, (1, len(seed), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    print int_to_char[seed[0]], "->", int_to_char[index]
    seed = [index]
model.reset_states()
```

The entire code listing is provided below for completeness.

```
# Stateful LSTM to learn one-char to one-char mapping
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
# fix random seed for reproducibility
numpy.random.seed(7)
# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# prepare the dataset of input to output pairs encoded as integers
seq_length = 1
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
    seq_in = alphabet[i:i + seq_length]
    seq_out = alphabet[i + seq_length]
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
    dataX.append([char_to_int[char] for char in seq_in])
    dataY.append(char_to_int[seq_out])
    print seq_in, '->', seq_out
# reshape X to be [samples, time steps, features]
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
# normalize
X = X / float(len(alphabet))
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# create and fit the model
batch_size = 1
model = Sequential()
model.add(LSTM(16, batch_input_shape=(batch_size, X.shape[1], X.shape[2]), stateful=True))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
for i in range(300):
    model.fit(X, y, epochs=1, batch_size=batch_size, verbose=2, shuffle=False)
    model.reset_states()
# summarize performance of the model
scores = model.evaluate(X, y, batch_size=batch_size, verbose=0)
model.reset_states()
print("Model Accuracy: %.2f%%" % (scores[1]*100))
# demonstrate some model predictions
seed = [char_to_int[alphabet[0]]]
for i in range(0, len(alphabet)-1):
    x = numpy.reshape(seed, (1, len(seed), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    print int_to_char[seed[0]], "->", int_to_char[index]
    seed = [index]
model.reset_states()
# demonstrate a random starting point
letter = "K"
seed = [char_to_int[letter]]
print "New start: ", letter
for i in range(0, 5):
    x = numpy.reshape(seed, (1, len(seed), 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    print int_to_char[seed[0]], "->", int_to_char[index]
    seed = [index]
model.reset_states()
```

Running the example provides the following output.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
Model Accuracy: 100.00%
A -> B
B -> C
C -> D
D -> E
E -> F
F -> G
G -> H
H -> I
I -> J
J -> K
K -> L
L -> M
M -> N
N -> O
O -> P
P -> Q
Q -> R
R -> S
S -> T
T -> U
U -> V
V -> W
W -> X
X -> Y
Y -> Z
New start:  K
K -> B
B -> C
C -> D
D -> E
E -> F
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

We can see that the network has memorized the entire alphabet perfectly. It used the context of the dependency it needed to predict the next character in the sequence.

We can also see that if we seed the network with the first letter, that it can correctly rattle off the rest of the alphabet.

We can also see that it has only learned the full alphabet sequence and that from a cold start. When asked to predict the next letter from “K” that it predicts “B” and falls back into regurgitating the entire alphabet.

To truly predict “K” the state of the network would need to be warmed up iteratively fed the letters from “A” to “J”. This tells us that we could achieve the same effect with a “stateless” LSTM by preparing training data like:

Get Your Start in Machine Learning

```
---a -> b
--ab -> c
-abc -> d
abcd -> e
```

Where the input sequence is fixed at 25 (a-to-y to predict z) and patterns are prefixed with zero-padding.

Finally, this raises the question of training an LSTM network using variable length input sequences to predict the next character.

LSTM with Variable-Length Input to One-Char Output

In the previous section, we discovered that the Keras "stateful" LSTM was really only a shortcut to replaying the first n-sequences, but didn't really help us learn a generic model of the alphabet.

In this section we explore a variation of the "stateless" LSTM that learns random subsequences of th be given arbitrary letters or subsequences of letters and predict the next letter in the alphabet.

Firstly, we are changing the framing of the problem. To simplify we will define a maximum input sequ eed up training. This defines the maximum length of subsequences of the alphabet will be drawn for train alphabet (26) or longer if we allow looping back to the start of the sequence.

We also need to define the number of random sequences to create, in this case 1000. This too could lly required.

```
# prepare the dataset of input to output pairs encoded as integers
num_inputs = 1000
max_len = 5
dataX = []
dataY = []
for i in range(num_inputs):
    start = numpy.random.randint(len(alphabet)-2)
    end = numpy.random.randint(start, min(start+max_len,len(alphabet)-1))
    sequence_in = alphabet[start:end+1]
    sequence_out = alphabet[end + 1]
    dataX.append([char_to_int[char] for char in sequence_in])
    dataY.append(char_to_int[sequence_out])
    print sequence_in, '->', sequence_out
```

Running this code in the broader context will create input patterns that look like the following:

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

```
PQRST -> U
W -> X
O -> P
OPQ -> R
IJKLM -> N
QRSTU -> V
ABCD -> E
X -> Y
GHIJ -> K
```

The input sequences vary in length between 1 and **max_len** and therefore require zero padding. Here, we use left-hand-side (prefix) padding with the Keras built in **pad_sequences()** function.

```
X = pad_sequences(dataX, maxlen=max_len, dtype='float32')
```

The trained model is evaluated on randomly selected input patterns. This could just as easily be new randomly generated sequences of characters. I also believe this could also be a linear sequence seeded with “A” with outputs fes back in as single chara

The full code listing is provided below for completeness.

```
# LSTM with Variable Length Input Sequences to One Character Output
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences
from theano.tensor.shared_randomstreams import RandomStreams
# fix random seed for reproducibility
numpy.random.seed(7)
# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# prepare the dataset of input to output pairs encoded as integers
num_inputs = 1000
max_len = 5
dataX = []
dataY = []
for i in range(num_inputs):
    start = numpy.random.randint(len(alphabet)-2)
    end = numpy.random.randint(start, min(start+max_len,len(alphabet)-1))
    sequence_in = alphabet[start:end+1]
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
    sequence_out = alphabet[end + 1]
    dataX.append([char_to_int[char] for char in sequence_in])
    dataY.append(char_to_int[sequence_out])
    print sequence_in, '->', sequence_out
# convert list of lists to array and pad sequences if needed
X = pad_sequences(dataX, maxlen=max_len, dtype='float32')
# reshape X to be [samples, time steps, features]
X = numpy.reshape(X, (X.shape[0], max_len, 1))
# normalize
X = X / float(len(alphabet))
# one hot encode the output variable
y = np_utils.to_categorical(dataY)
# create and fit the model
batch_size = 1
model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], 1)))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=500, batch_size=batch_size, verbose=2)
# summarize performance of the model
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
# demonstrate some model predictions
for i in range(20):
    pattern_index = numpy.random.randint(len(dataX))
    pattern = dataX[pattern_index]
    x = pad_sequences([pattern], maxlen=max_len, dtype='float32')
    x = numpy.reshape(x, (1, max_len, 1))
    x = x / float(len(alphabet))
    prediction = model.predict(x, verbose=0)
    index = numpy.argmax(prediction)
    result = int_to_char[index]
    seq_in = [int_to_char[value] for value in pattern]
    print seq_in, "->", result
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Running this code produces the following output:

```
Model Accuracy: 98.90%
['Q', 'R'] -> S
['W', 'X'] -> Y
['W', 'X'] -> Y
['C', 'D'] -> E
['E'] -> F
['S', 'T', 'U'] -> V
['G', 'H', 'I', 'J', 'K'] -> L
['O', 'P', 'Q', 'R', 'S'] -> T
['C', 'D'] -> E
```

Get Your Start in Machine Learning

```
['O'] -> P
['N', 'O', 'P'] -> Q
['D', 'E', 'F', 'G', 'H'] -> I
['X'] -> Y
['K'] -> L
['M'] -> N
['R'] -> T
['K'] -> L
['E', 'F', 'G'] -> H
['Q'] -> R
['Q', 'R', 'S'] -> T
```

We can see that although the model did not learn the alphabet perfectly from the randomly generated subsequences, it did very well. The model was not tuned and may require more training or a larger network, or both (an exercise for the reader).

This is a good natural extension to the "*all sequential input examples in each batch*" alphabet model learned above in that it can handle ad hoc queries, but this time of arbitrary sequence length (up to the max length).

Summary

In this post you discovered LSTM recurrent neural networks in Keras and how they manage state.

Specifically, you learned:

- How to develop a naive LSTM network for one-character to one-character prediction.
- How to configure a naive LSTM to learn a sequence across time steps within a sample.
- How to configure an LSTM to learn a sequence across samples by manually managing state.

Do you have any questions about managing LSTM state or about this post?
Ask your questions in the comment and I will do my best to answer.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

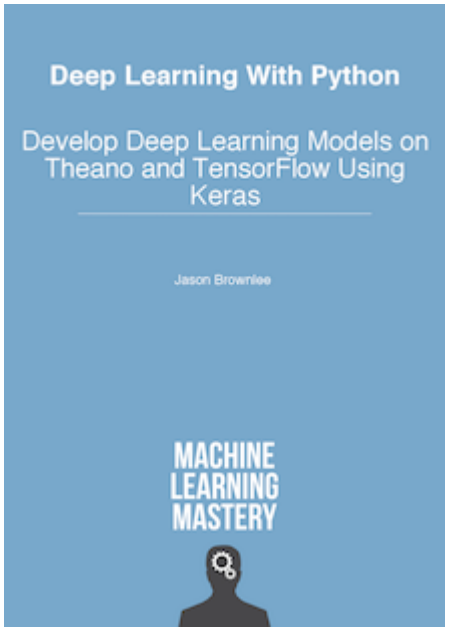
START MY EMAIL COURSE

Frustrated With Your Progress In Deep Learning?

What If You Could Develop A Network in Min

…with just a few lines of Python

Get Your Start in Machine Learning



Discover how in my new Ebook: Deep Learning With Python

It covers **self-study tutorials** and **end-to-end projects** on topics like:
Multilayer Perceptrons, *Convolutional Nets* and *Recurrent Neural Nets*, and more…

Finally Bring Deep Learning To Your Own Projects

Skip the Academics. Just Results.

Click to learn more.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

About Jason Brownlee

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

‹ How to Use Ensemble Machine Learning Algorithms in Weka

How To Compare the Performance of Machine Learning Algorithms in Weka ›

Get Your Start in Machine Learning

88 Responses to *Understanding Stateful LSTM Recurrent Neural Networks in Python with Keras*

Mark July 28, 2016 at 6:02 pm #

REPLY

Great series of posts on LSTM networks recently. Keep up the good work

Jason Brownlee July 29, 2016 at 6:28 am #

REPLY

Thanks Mark.

Kilian August 31, 2017 at 4:30 am #

For anyone, who wants to try out trained LSTMs, there is an interactive chat box: http://ww
tensorflow/

It offers models trained on Wikipedia, Congress speeches, Sherlock Holmes, South Park and Goeth

I can also recommend the tensorlm package on GitHub: https://github.com/batzner/tensorlm

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee August 31, 2017 at 6:26 am #

REPLY

Thanks for sharing.

Atlant July 29, 2016 at 4:37 pm #

REPLY

Get Your Start in Machine Learning

I like this post, it gave me many enlightenment.

Jason Brownlee July 29, 2016 at 4:53 pm #

REPLY

Thanks Atlant.

shanbe August 17, 2016 at 9:47 pm #

REPLY

I'm probably missing something here but could you please explain why LSTM units are needed in the alphabet example where any output depends directly on the input letter and there is no confusion between different input -> output pairs at all?

Jason Brownlee August 18, 2016 at 7:25 am #

It is a demonstration of the algorithms ability to learn a sequence. Not just input-output pairs

Randy October 2, 2016 at 8:19 pm #

hi, I got a little confused there. What does the LSTM units mean?
Thanks

Jason Brownlee October 3, 2016 at 5:19 am #

REPLY

Hi Randy, the LSTM units are the "memory units" or you can just call them the neurons.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

dolaameng August 24, 2016 at 10:28 pm #

REPLY

Thank you for sharing these educational examples! Appreciate it if you can elaborate on the “LSTM State Within A Batch” example. The confusing part is on the explanation “that the LSTM could use the context of the sequence within the batch to better learn the sequence.”, which may imply that the states of LSTM are reused within the training of a batch, and this motivated the setting of parameter shuffle as False.

But my understanding of Keras’s implementation is that the LSTM won’t reuse the states within a batch. In fact, the sequences in a batch is kinda like triggering the LSTM “in parallel” – in fact the states of LSM should be of shape (nsamples, nout) for both gates – separate states for each sequence – this is what id described [Keras document](https://keras.io/getting-started/faq/#how-can-i-use-stateful-rnns): states are reused by the ith instance for successive batchs.

This means that even parameter shuffle is set as True, it will still give you the observed performance. This also explains why the predictions on random patterns were also good, which was opposite to the observations in the next example “Stateful LSTM for a One-Char to One-Char Mapping”. The reason why setting a bigger batch size resulted in better performance than the first example, could be the bigger nb_

Appreciate your opinions on this! It’s a great article anyway!

Rodrigo Pinto October 29, 2016 at 4:35 am #

You are right, that state from sequence to sequence inside one batch.

Hadi September 5, 2016 at 2:52 am #

Thank you for your amazing tutorial.
However, what if I want to predict a sequence of outputs? If I add a dimension to the output it is gonna be like a features window and the model will not consider the outputs as a sequence of outputs. It is like outputs are independent. How can I fix that issue and have a model which for example generates “FGH” when I give it “BCDE”.

Wes March 11, 2017 at 1:10 am #

REPLY

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Hadi, I use a “one-hot” encoding for my features. This causes the output to be a probability distribution over the features. You may then use a sampling technique to choose “next” features, similar to the “Unreasonable Effectiveness of RNN” article. This method includes a “temperature” parameter that you can tune to produce outputs with more or less adherence to the LSTM’s predictions.

Wes March 11, 2017 at 9:32 am #

REPLY

One more follow-up to this problem… you may be interested in a different type of network, such as a Generative Adversarial Network (GAN) https://github.com/jacobgil/keras-dcgan

Alex September 15, 2016 at 5:18 pm #

Also for my part than you for the tutorial.

However, I have a few related questions (also posted in StackOverflow, http://stackoverflow.com/questio stateful-rnns): If I have a stateful RNN with just one timestep per batch, how is backpropagation handled accumulate updates for the entire sequence? I fear that it updates only the timesteps per batch and noth drawback? Or do you know a way to overcome this?

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee September 16, 2016 at 9:01 am #

Hi Alex,

I believe updates are performed after each batch. This can be a downside of having a small batch. A batch size of 1 will essentially perform online gradient descent (I would guess). Developers more familiar with Keras internals may be able to give you a more concrete answer.

Alex September 17, 2016 at 3:24 am #

REPLY

Hi Jason,

Get Your Start in Machine Learning

thank you very much for your answer.

I posed my question wrongly because I mixed up “batch size” and “time steps”. If I have sequences of shape (nb_samples, n, dims) and I process them one time step after the other with a stateful LSTM (feeding batches of shape (batch_size, 1, dims) to the network), will backpropagation go through the entire sequences as it would if I processed the entire sequence at once?

Jason Brownlee September 17, 2016 at 9:37 am #

REPLY

The answer does not change, updates happen after each batch.

Vishal September 21, 2016 at 11:28 am #

“The Keras implementation of LSTMs resets the state of the network after each batch.”

Could you please explain what you mean by “resets the state”? What happens to the network state after

Thanks!

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee September 22, 2016 at 8:05 am #

I don’t know if it is clear or just unreliable. But it is gone.

Your network will not behave as you expect.

Arnold September 21, 2016 at 10:43 pm #

REPLY

Hi Jason,

first, thanks for the amazing tutorial!

Get Your Start in Machine Learning

I got two quick questions:

1) if I want to train “LSTM with Variable Length Input to One-Char Output” on much simpler sequence (with inserted repetitive pattern):

seq_1 = “aaaabbbccdaaaabbbccdaaaabbbccd….”

or even

se1_2 = “abababababababababababababab…”

I can’t do any better than 50% of accuracy. What’s wrong?

2) If I want to amend your code to N-Char-Output, how is it possible? So, given a sequence “abcd” -> “ef” ?

Many thanks in advance!

Arnold September 21, 2016 at 10:56 pm #

I found my mistake: the char-integer encoding should be:

```
chars = sorted(list(set(alphabet)))
char_to_int = dict((c, i) for i, c in enumerate(chars))
int_to_char = dict((i, c) for i, c in enumerate(chars))
```

but, it does not help too much.

Also, I’m very curious to know how to predict N-characters

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee September 22, 2016 at 8:13 am #

REPLY

Hi Arnold,

Nice change. No idea why it’s not learning it. Perhaps you need more nodes/layers or longer training? Maybe stateful or stateless? Perhaps it’s the framing of the problem?

This is a sequence to sequence problem. I expect you can just change the output node from 1 neuro

Get Your Start in Machine Learning

Arnold September 25, 2016 at 8:55 pm #

REPLY

Hi Jason,

Thanks for your suggestions. I will try.

Riccardo Folloni October 20, 2016 at 9:02 pm #

REPLY

Arnold,

have you acheived that kind of implementation? 🙂
i'm also interested

Arnold September 21, 2016 at 11:47 pm #

Hi Jason,

Thanks for the amazing tutorial and other posts!

How to amend your code to predict N-next characters and not only one?

something like: "LSTM with Variable Length Input to N-Char Output"

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Madhav October 4, 2016 at 5:59 pm #

REPLY

Hi Jason,

Great tutorial as always. It was fun to run through your code line by line, work with a smaller alphabet (like "ABCDE"), change the sequence length etc just to figure out how the model behaves. I think I'm growing fond of LSTMs these days.

Get Your Start in Machine Learning

I have a very basic question about the shape of the input tensor. Keras requires our input to have the form [samples, time_steps, features]. Could you tell me what the attribute features exactly means?

Also, consider a scenario for training an LSTM network for binary classification of audio. Suppose I have a collection of 1000 files and for each file, I have extracted 13 features (MFCC values). Also suppose that every file is 500 frames long and I set the time_steps to 1.

What would be my input shape?

1. [Number of files, time_steps, features] = [1000, 1, 13]

or

2. [Number of files * frames_per_file, time_steps, features] = [1000*500, 1, 13]

Any answer is greatly appreciated !! Thanks.

Jason Brownlee October 5, 2016 at 8:27 am #

Thanks Madhav.

The features in the input refers to attributes or columns in your data.

Yes, your framing of the problem looks right to me [1000, 1, 13] or [samples, timesteps, features]

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Rob October 22, 2016 at 12:31 am #

Thank you for the enlightening series of articles on LSTMs!
Just a minor detail, the complete code for the final example is missing an import for pad_sequences:
from keras.preprocessing.sequence import pad_sequences

Jason Brownlee October 22, 2016 at 7:02 am #

REPLY

Fixed.

Get Your Start in Machine Learning

Thanks for pointing that out Rob!

Panand November 8, 2016 at 2:00 am #

REPLY

Hello Jason

What if I want to predict a non sequential network.(eg:- the next state of A may be B,C,Z or A itself depending on the previous values before A). Can I use this same logic for that problem?

Jason Brownlee November 8, 2016 at 9:55 am #

Yes Panand, LSTMs can learn arbitrary complex sequences, although you may need to sca… complexity.

Let me know how you go.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Veronica November 16, 2016 at 4:21 am #

Hi Jason,
Thank you for your tutorials, I learn a lot from them.
However, I still don't quite understand the meaning of batch size. In the last example you set batch_size… the sequence based on the whole sequence, or was it just based on the last letter every time?
What would have happened if you set batch_size=3 and all the sequences would be at minimal length 3?
Thank you

Manisha December 14, 2016 at 6:16 pm #

REPLY

Hi Jason ,

Get Your Start in Machine Learning

Thank you for this wonderful tutorial.
Motivated by this I got myself trying to generate a sine wave using RNN in theano

After running the code below

the sine curve predicted in green is generated well when I give an input to all the time steps at prediction

but the curve in blue is not predicted well where I give the input to only the first time step while prediction (This kind of prediction is you have used for character sequence generation)

Is there a way I could make that work. Cause I need a model for generating captions.

```
#Learning Sine wave
import theano
import numpy as np
import matplotlib.pyplot as plt
import theano.tensor as T
import math
theano.config.floatX = 'float64'

## data
step_radians = 0.01
steps_of_history = 200
steps_in_future = 1
index = 0

x = np.sin(np.arange(0, 20*math.pi, step_radians))

seq = []
next_val = []

for i in range(0, len(x)-steps_of_history, steps_of_history):
seq.append(x[i: i + steps_of_history])
next_val.append(x[i+1:i + steps_of_history+1])

seq = np.reshape(seq, [-1, steps_of_history, 1])
next_val = np.reshape(next_val, [-1, steps_of_history, 1])
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
trainX = np.array(seq)
trainY = np.array(next_val)

## model
n = 50
nin = 1
nout = 1

u = T.matrix()

t = T.matrix()

h0 = T.vector()
h_in = np.zeros(n).astype(theano.config.floatX)
lr = T.scalar()

W = theano.shared(np.random.uniform(size=(3,n, n), low=-.01, high=.01).astype(theano.config.floatX))
W_in = theano.shared(np.random.uniform(size=(nin, n), low=-.01, high=.01).astype(theano.config.floatX
W_out = theano.shared(np.random.uniform(size=(n, nout), low=-.01, high=.01).astype(theano.config.floa

def step(u_t, h_tm1, W, W_in, W_out):
h_t = T.tanh(T.dot(u_t, W_in) + T.dot(h_tm1, W[0]))
h_t1 = T.tanh(T.dot(h_t, W[1]) + T.dot(h_tm1, W[2]))
y_t = T.dot(h_t1, W_out)
return h_t, y_t

[h, y], _ = theano.scan(step,
sequences=u,
outputs_info=[h0, None],
non_sequences=[W, W_in, W_out])

error = ((y – t) ** 2).sum()
prediction = y
gW, gW_in, gW_out = T.grad(error, [W, W_in, W_out])

fn = theano.function([h0, u, t, lr],
error,
updates={W: W – lr * gW,
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
W_in: W_in – lr * gW_in,
W_out: W_out – lr * gW_out})
predict = theano.function([h0, u], prediction)

for e in range(10):
for i in range(len(trainX)):
fn(h_in,trainX[i],trainY[i],0.001)

print('End of training')

x = np.sin(np.arange(20*math.pi, 24*math.pi, step_radians))

seq = []

for i in range(0, len(x)-steps_of_history, steps_of_history):
seq.append(x[i: i + steps_of_history])

seq = np.reshape(seq, [-1, steps_of_history, 1])
testX = np.array(seq)

# Predict the future values
predictY = []
for i in range(len(testX)):
p = testX[i][0].reshape(1,1)
for j in range(len(testX[i])):
p = predict(h_in, p)
predictY= predictY + p.tolist()
print(predictY)
# Plot the results

plt.plot(x, 'r-', label='Actual')
plt.plot(np.asarray(predictY), 'gx', label='Predicted')
predictY = []
for i in range(len(testX)):
predictY= predictY + predict(h_in, testX[i]).tolist()
plt.plot(np.asarray(predictY), 'bo', label='Predicted')

plt.show()
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

Many Thanks

Louis Abraham December 19, 2016 at 4:11 am #

REPLY

Hi,

I think there is a mistake in the paragraph “LSTM State Within A Batch”.
You say “The Keras implementation of LSTMs resets the state of the network after each batch.”.
But the fact is that it resets its state event between the inputs of a batch.
You can see the poor performances (around .5) of this LSTM that tries to remember its last input : http://pastebin.com/u5NnAx9r

As a comparison, here is a stateful LSTM that performs well : http://pastebin.com/qEKBVqJJ

Mazen December 30, 2016 at 8:26 pm #

Thank you very much.
It is the best description I’ve ever seen about LSTM. I got a lot of benefits from your post!

Jason Brownlee December 31, 2016 at 7:03 am #

Thanks Mazen.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Fernando January 17, 2017 at 4:05 am #

REPLY

First of all, thaks for create this easy but very useful example to learn and understand better how the LSTM nets work.

I just want to ask you something about the first part, why do we use 32 units? was it a random decision or does it have a theoric fundament?

I will thank you your answer.

Get Your Start in Machine Learning

Jason Brownlee January 17, 2017 at 7:41 am #

REPLY

An ad hoc choice Fernando. Great question.

leila January 18, 2017 at 12:51 am #

REPLY

Hi Jason,

thanks for the amazing tutorial!

Jason Brownlee January 18, 2017 at 10:16 am #

I'm glad you found it useful leila.

floyd January 25, 2017 at 2:16 am #

Thank you for this post but i am a beginner and i have a question
you said "the state of the network is reset after each pattern" what does that mean?

Jason Brownlee January 25, 2017 at 10:07 am #

REPLY

LSTMs maintain an internal state, it is a benefit of using them.

This internal state can be reset automatically (after each batch) or manually when setting the "stateful" argument.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

atmaere March 18, 2017 at 8:38 pm #

REPLY

Great tutorial. Question: Is it possible to do a stateful LSTM with variable input (like the last example in the tutorial)? I am curious why you used a naive stateless LSTM for that.

Jason Brownlee March 19, 2017 at 6:10 am #

REPLY

Yes you can.

Bikram Gupta April 3, 2017 at 1:25 pm #

Thank you for the amazing tutorial. The walkthrough (starting with simple and building on it) rea

Jason Brownlee April 4, 2017 at 9:12 am #

You're welcome Bikram.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

selfdrivingvehicle April 28, 2017 at 9:36 pm #

I have one question in regards to the LSTM. I asked this question in details here on reddit, and after than came to know about your tutorial here.

I feel that with stateful LSTMs, I am getting the gist of the RNNs, but still has it been ever successfully used for video processing — like the problem I described here https://www.reddit.com/r/AskReddit/comments/681c76/understanding_lstm_stateful_stateless_how_to_go/

Basically, here, with stateful LSTM when the network was fed "K", it predicted B, which is wrong. Does it mean that it is just memorized sequence and will always predict B irrespective of what is fed. How can we extend it to general video prediction tasks. Please look at the link above.

thanks in advance

Get Your Start in Machine Learning

Jason Brownlee April 29, 2017 at 7:27 am #

REPLY

The link does not appear to work.

LSTMs require a very careful framing of your sequence problem in order to learn effectively. They also ofter require a much larger network trained for much longer than intuitions on MLPs may suggest.

Melih May 8, 2017 at 1:57 am #

REPLY

Hello.

I have a question about RNN-LSTM network, how can we decide input_shape size? how can we draw s

Jason Brownlee May 8, 2017 at 7:46 am #

Great question.

The LSTMs in keras expect the shape to be [samples, timesteps, features].

Each sample is a separate sequence. The sequence steps are time steps. Each observation measu

If you have one very long sequence, you can break it up into multiple smaller sequences.

I hope that helps.

Penhchet May 8, 2017 at 6:56 pm #

REPLY

Could we use the LSTMS RNN in Keras to create the Chatbot Conversation Model.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee May 9, 2017 at 7:39 am #

REPLY

Perhaps, I do not have an example sorry.

Yuanliang Meng May 22, 2017 at 8:59 am #

REPLY

Sometimes it is helpful to extract LSTM outputs from certain time steps. For example, we may want to get the outputs corresponding to the last 30 words from a text, or the 100th~200th words from a text body. It would be great to write an instruction about it.

Jason Brownlee May 23, 2017 at 7:47 am #

Thanks for the suggestion.

Can you give an example of what you mean?

Farooq Zaman May 29, 2017 at 3:52 pm #

Hi Dear Sir (Jason) .

i want to use LSTM for words instead of alphabets. how can i implement that. more over can i use that p
a sequential problem because it also dependent on context.
thanks

Jason Brownlee June 2, 2017 at 12:20 pm #

REPLY

You can, but I do not have any examples sorry.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Carlos June 1, 2017 at 11:24 pm #

REPLY

Hi Jason,

Thank you for this great tutorial. I really appreciated going through your LSTM code.
I have one question about the “features”: you are mentionning them but I don’t see where you are including them. Do they represent a multivariate problem? How do we handle that?

Many thanks,
Carlos

Jason Brownlee June 2, 2017 at 1:01 pm #

I do not have a multivariate example at this stage, but you can specify features in the input.

Reihaneh June 16, 2017 at 6:19 am #

Thanks for this great tutorial!
I have a question. I am deadling with kinda same problem. However, each character in a sequence is a v
Instead of [‘A’, ‘B’, ‘C’] -> D I have [[0, 0,1, 1.3], [6,3,1,1.5], [6, 4, 1.4, 4.5]] -> [1, 3, 4]

So considering all sequences, my data is in 3d shape.
Could someone help me how to configure the input for LSTM?

Jason Brownlee June 16, 2017 at 8:11 am #

REPLY

LSTM input must be 3D, the structure is [samples, timesteps, features].

If you one hot encode, then the number of labels (length of binary vectors) is the number of features

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Does that help?

Reihaneh June 17, 2017 at 2:58 am #

REPLY

It does! for clarification:
If I have 2000 sentences (sequence of words)
Each sentence is having 3 words,
and each word is converted into a vector of length 5 (5 features):

X1 = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12,13,14,15]], ….., X2000
Data_X = [X1, …, X2000]

So my X_train is gonna be like this?
X = numpy.reshape(Data_X, (2000, 3, 5))

Jason Brownlee June 17, 2017 at 7:34 am #

Yes.

But don’t be afraid to explore other representations that may result in better skill on your pro
sentences as time steps.

Reihaneh Amini June 21, 2017 at 3:30 am #

Thanks Jason!

I really appreciate your attempt in these tutorials.

I am working with a huge imbalanced sequential file, do you have any suggestion regarding improving these type of imbalanced files?

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee June 21, 2017 at 8:17 am #

I do not have experience with imbalanced sequence data.

Perhaps methods for imbalanced non-sequence data will give you some ideas:
http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/

Reihaneh June 17, 2017 at 2:59 am #

REPLY

btw, assuming that the output is not high dimentional:
Y1 = [11, 22, 33]

Fiora June 27, 2017 at 12:55 pm #

Hello, In this example you were able to go from ABC -> D (3 to 1)
Is there a way there Train a model to go from ABC -> BCD
or perhaps some other pattern like ABC- >CBA(switching first and last)

Jason Brownlee June 28, 2017 at 6:18 am #

Yes, you can have 3 neurons in the output layer.

Puzzled August 3, 2017 at 8:51 pm #

REPLY

Great tutorial and one that has come closest to ending my confusion. However I still not 100% clear. If I have time-series training data of length 5000, and each row of data consists of two features, can I shape my input data (x) as a single batch of 5000,1,2? And if this is the case — under what

Get Your Start in Machine Learning

×

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

circumstances would I ever need to increase the time step to more than 1? I'm struggling to see the value the time step dimension is adding if the LSTM remembers across an entire batch (which in my above scenario would be like a time step of 5000 right?)

Jason Brownlee August 4, 2017 at 7:00 am #

REPLY

Yes.

The number of time steps should be equal to the number of time steps in your input sequence. Setting time steps to 1 is generally not a good idea in practice beyond demonstrations.

Liuwei August 7, 2017 at 6:14 pm #

Hi Jason,
Thanks for your tutorials, I just have a confusion on stateful LSTM in keras.
I know the hidden state will pass through different timesteps. Will the hidden state pass among one batc
Or samples in one batch have different initialized hidden states?

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee August 8, 2017 at 7:45 am #

The internal state is reset at the end of each batch.

You can carry the state across batches by making the layer stateful and taking control of when the state is reset.

Does that help?

Aiy August 18, 2017 at 6:54 pm #

REPLY

Hi Jason,

Get Your Start in Machine Learning

Is one hot encoder needed for X? Since they are also categorical.

Jason Brownlee August 19, 2017 at 6:15 am #

REPLY

Yes, the input can be one hot encoded.

Nick Shawn August 24, 2017 at 1:19 pm #

REPLY

It's a awesome tutorial, but I still have some problem.When we use a LSTM/RNN, we usually initialize the state of LSTM/RNN with some random way like Orthogonal, therefore, when we use the LSTM for predicting, the initial state may be or must be prediction always.Even more, when we train a LSTM with stateful=False, the initial state will be reset, wh a right model?
wait for your answer, thank you!

Jason Brownlee August 24, 2017 at 4:26 pm #

See this post for advice on how to evaluate neural nets like LSTMs given their stochastic na
https://machinelearningmastery.com/evaluate-skill-deep-learning-models/

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Viktor August 30, 2017 at 7:14 pm #

REPLY

I have experiments with all the examples. It seemes for me that only one example "Stateful LSTM for a One-Char to One-Char Mapping" showing the nature of LSTM/RNN. All other examples simply work line ordinary Dense networks.
I've tried to work with sequence "ABCDEFABCXYZ"
It learned well on Stateful network.
Model Accuracy: 100.00%
['A'] -> B

Get Your Start in Machine Learning

['B'] -> C
['C'] -> D
['D'] -> E
['E'] -> F
['F'] -> A
['A'] -> B
['B'] -> C
['C'] -> X
['X'] -> Y
['Y'] -> Z

But in random test it makes mistakes:

Test a Random Pattern:
['Y'] -> B
['A'] -> C
['C'] -> D
['A'] -> E
['X'] -> F
['Y'] -> A
['D'] -> B
['B'] -> C
['A'] -> X
['A'] -> Y
['B'] -> Z
['C'] -> Z
['E'] -> Z
['B'] -> Z
['C'] -> Z
['C'] -> A
['B'] -> A
['C'] -> A
['A'] -> B
['F'] -> B

How to fix that?

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

Jason Brownlee August 31, 2017 at 6:16 am #

REPLY

A more generic model needs to be developed, perhaps one that uses the context for the last n characters (time steps) to predict the next character.

DrMcCleod September 1, 2017 at 2:36 am #

REPLY

In the final example with variable length inputs, are the vectors generated by the call to pad_sequences() sensible?
For example:

RS -> T

Becomes

[[0], [0], [0], [17], [18]]

which is the equivalent of

AAARS -> T

Is that really what is wanted?

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Jason Brownlee September 1, 2017 at 6:49 am #

That might be a fault. Thanks!

DrMcCleod September 1, 2017 at 10:49 pm #

REPLY

Yeah, I can't work out if it is a good thing or a bad thing.

Get Your Start in Machine Learning

In its favour, it does train (and eventually approaches 100% accuracy if you up the batch size and epochs a bit).
On the other hand, it does seem that the training data is wrong and that it is learning the correct result in spite of it. I have tried an alternative dataset where the padded data is a random selection of letters. It doesn't work particularly well for sequences of only 1 letter, but longer sequences seem OK. I have pasted the new version below. Note that I have increased the dataset size, batch size and training epochs.

This LSTM business is a bit more subtle than I initially thought…. thanks for posting this tutorial though, I really like the "this is how to do it wrong" style of the first few examples. Very useful.

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences

# fix random seed for reproducibility
numpy.random.seed(7)

# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))

# prepare the dataset of input to output pairs encoded as integers
num_inputs = 10000
max_len = 5
dataX = []
dataY = []
for i in range(num_inputs):
start = numpy.random.randint(len(alphabet)-2)
end = numpy.random.randint(start, min(start+max_len,len(alphabet)-1))
sequence_in = alphabet[start:end+1]
sequence_out = alphabet[end + 1]
dataX.append([char_to_int[char] for char in sequence_in])
```

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
dataY.append(char_to_int[sequence_out])
print(sequence_in, '->', sequence_out)

for n in range(len(dataX)):
while len(dataX[n]) ", result)

#test arbitrary patterns
pattern = [20, 21, 22]
while len(pattern) ", result)
```

DrMcCleod September 1, 2017 at 10:52 pm #

REPLY

Did that listing get trimmed?

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils
from keras.preprocessing.sequence import pad_sequences

# fix random seed for reproducibility
numpy.random.seed(7)

# define the raw dataset
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
# create mapping of characters to integers (0-25) and the reverse
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))

# prepare the dataset of input to output pairs encoded as integers
num_inputs = 10000
max_len = 5
dataX = []
dataY = []
```


Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

```
for i in range(num_inputs):
start = numpy.random.randint(len(alphabet)-2)
end = numpy.random.randint(start, min(start+max_len,len(alphabet)-1))
sequence_in = alphabet[start:end+1]
sequence_out = alphabet[end + 1]
dataX.append([char_to_int[char] for char in sequence_in])
dataY.append(char_to_int[sequence_out])
print(sequence_in, '->', sequence_out)

for n in range(len(dataX)):
while len(dataX[n]) ", result)

#test arbitrary patterns
pattern = [20, 21, 22]
while len(pattern) ", result)
```

DrMcCleod September 1, 2017 at 10:55 pm #

Hmm, I don't seem to be able to post the full listing. Sorry.

Jason Brownlee September 2, 2017 at 6:10 am #

Consider using pre HTML tags when posting code.

Jason Brownlee September 2, 2017 at 6:09 am #

REPLY

I think the pure solution is to re-map the letters in the range 1-26 and reserve 0 for padding.

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Get Your Start in Machine Learning

Dan DeDora September 27, 2017 at 4:45 am #

REPLY

Hey Prof Brownlee,

You do great work. I appreciate the diligent and detailed tutorials – thank you very much.

My question pertains to how lstm states might apply to non-“sequential” (but still time dependent) data, a la your air pollution tutorial – https://machinelearningmastery.com/multivariate-time-series-forecasting-lstms-keras/

1) Does setting an LSTM to [stateful = false] and using batch size 1 basically turn an LSTM into a more complicated Feedforward net, i.e., a neural net with no memory or knowledge of sequence?

2) After training a “stateful” model above, you reset the states and then make predictions. This means (wrt to the standard lstm equations, found here – https://en.wikipedia.org/wiki/Long_short-term_memory) that the previous cell state, and the hidden state model.get_layer(index = 1).states[0].eval(). This is also true of a non-stateful model – the states are liste resetting the state makes the forget gate and the “U” weights zero-out (as per the equations). Yet, as we accurate predictions! It makes me wonder why we have a forget gate and U weights at all?

If my questions are confusing in any way, please let me know. Thanks ahead of time for your attention, a

Jason Brownlee September 27, 2017 at 5:51 am #

Hi Dan, great questions.

No, LSTM memory units are still very different to simple neurons in a feed-forward network.

See this tutorial that forces the model to only use the internal state to predict the outcome: https://machinelearningmastery.com/memory-in-a-long-short-term-memory-network/

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

Leave a Reply

Get Your Start in Machine Learning

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

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

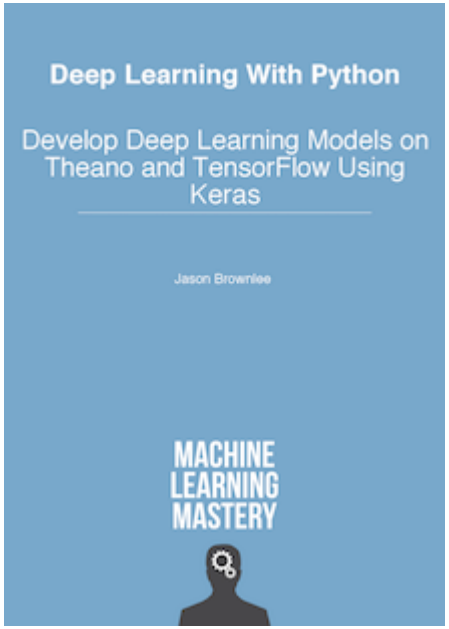
Finally Get Started With Deep Learning

Sick of the fancy math and need for super computers?
Looking for step-by-step tutorials?

Get Your Start in Machine Learning

Want end-to-end projects?

Get Started With Deep Learning in Python Today!



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

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

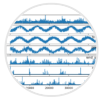
Get Your Start in Machine Learning

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

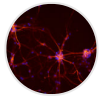
Multi-Class Classification Tutorial with the Keras Deep Learning Library
JUNE 2, 2016

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

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

Get Your Start in Machine Learning

You can master applied Machine Learning **without the math or fancy degree**.
Find out how in this *free* and *practical* email course.

Email Address

START MY EMAIL COURSE

© 2017 Machine Learning Mastery. All Rights Reserved.

Get Your Start in Machine Learning