A Gentle Introduction to LSTM Autoencoders

by **Jason Brownlee** on November 5, 2018 in **Long Short-Term Memory Networks**

Last Updated on August 14, 2019

An LSTM Autoencoder is an implementation of an autoencoder for sequence data using an Encoder-Decoder LSTM architecture.

Once fit, the encoder part of the model can be used to encode or compress sequence data that in turn may be used in data visualizations or as a feature vector input to a supervised learning model.

In this post, you will discover the LSTM Autoencoder model and how to implement it in Python using Keras.

After reading this post, you will know:

- Autoencoders are a type of self-supervised learning model that can learn a compressed representation of input data.
- LSTM Autoencoders can learn a compressed representation of sequence data and have been used on video, text, audio, and time series sequence data.
- How to develop LSTM Autoencoder models in Python using the Keras deep learning library

Discover how to develop LSTMs such as stacked, bic q
and more in my new book, with 14 step-by-step tutor

Let's get started.

A Gentle Introduction to LSTM Autoencoders
Photo by Ken Lund, some rights reserved.

Overview

This post is divided into six sections; they are:

1. What Are Autoencoders?
2. A Problem with Sequences
3. Encoder-Decoder LSTM Models
4. What Is an LSTM Autoencoder?
5. Early Application of LSTM Autoencoder
6. How to Create LSTM Autoencoders in Keras

What Are Autoencoders?

An autoencoder is a neural network model that seeks

They are an unsupervised learning method, although t ning
methods, referred to as self-supervised. They are typi
attempts to recreate the input.

For example:

```
X = model.predict(X)
```

The design of the autoencoder model purposefully makes this challenging by restricting the architecture to a bottleneck at the midpoint of the model, from which the reconstruction of the input data is performed.

There are many types of autoencoders, and their use varies, but perhaps the more common use is as a learned or automatic feature extraction model.

In this case, once the model is fit, the reconstruction aspect of the model can be discarded and the model up to the point of the bottleneck can be used. The output of the model at the bottleneck is a fixed length vector that provides a compressed representation of the input data.

Input data from the domain can then be provided to the model and the output of the model at the bottleneck can be used as a feature vector in a supervised learning model, for visualization, or more generally for dimensionality reduction.

A Problem with Sequences

Sequence prediction problems are challenging, not least because the length of the input sequence can vary.

This is challenging because machine learning algorithms, and neural networks in particular, are designed to work with fixed length inputs.

Another challenge with sequence data is that the temporal ordering of the observations can make it challenging to extract features suitable for use as input to supervised learning models, often requiring deep expertise in the domain or in the field of signal processing.

Finally, many predictive modeling problems involving sequences require a prediction that itself is also a sequence. These are called sequence-to-sequence, or seq2seq, prediction problems.

You can learn more about sequence prediction problems here:

- Making Predictions with Sequences

Encoder-Decoder LSTM Models

Recurrent neural networks, such as the Long Short-Te
designed to support sequences of input data.

They are capable of learning the complex dynamics w
well as use an internal memory to remember or use in

The LSTM network can be organized into an architect
the model to be used to both support variable length
length output sequences.

This architecture is the basis for many advances in complex sequence prediction problems such as speech recognition and text translation.

In this architecture, an encoder LSTM model reads the input sequence step-by-step. After reading in the entire input sequence, the hidden state or output of this model represents an internal learned representation of the entire input sequence as a fixed-length vector. This vector is then provided as an input to the decoder model that interprets it as each step in the output sequence is generated.

You can learn more about the encoder-decoder architecture here:

- Encoder-Decoder Long Short-Term Memory Networks

What Is an LSTM Autoencoder?

An LSTM Autoencoder is an implementation of an autoencoder for sequence data using an Encoder-Decoder LSTM architecture.

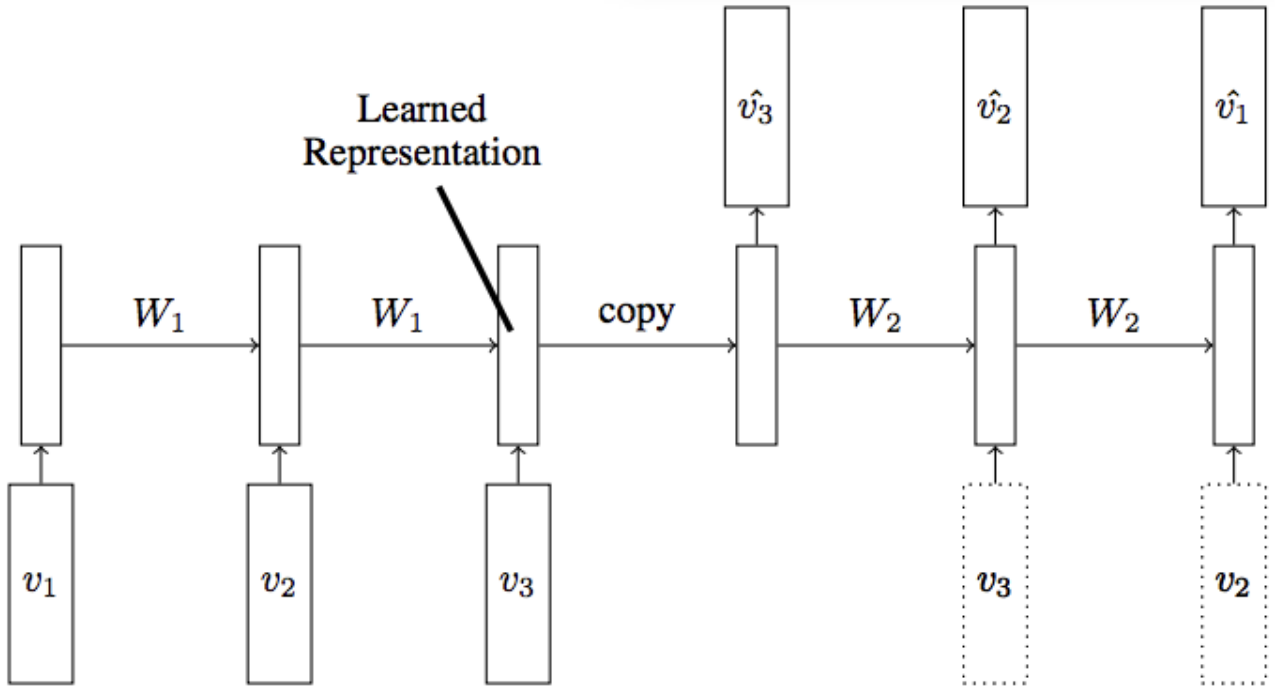
For a given dataset of sequences, an encoder-decoder LSTM is configured to read the input sequence, encode it, decode it, and recreate it. The performance of the model is evaluated based on the model's ability to recreate the input sequence.

Once the model achieves a desired level of performance recreating the sequence, the decoder part of the model may be removed, leaving just the encoder model. This model can then be used to encode input sequences to a fixed-length vector.

The resulting vectors can then be used in a variety of applications, not least as a compressed representation of the sequence as an input to another supervised learning model.

Early Application of LSTM Autoencoder

One of the early and widely cited applications of the LSTM Autoencoder was in the 2015 paper titled "Unsupervised Learning of Video Representations using LSTMs."



LSTM Autoencoder Model
Taken from "Unsupervised Learning of Video Representations using LSTMs"

In the paper, Nitish Srivastava, et al. describe the LSTM Autoencoder as an extension or application of the Encoder-Decoder LSTM.

They use the model with video input data to both reconstruct sequences of frames of video as well as to predict frames of video, both of which are described as an unsupervised learning task.

> *The input to the model is a sequence of vectors (image patches or features). The encoder LSTM reads in this sequence. After the last input has been read, the decoder LSTM takes over and outputs a prediction for the target sequence.*

— Unsupervised Learning of Video Representations using LSTMs, 2015.

More than simply using the model directly, the authors explore some interesting architecture choices that may help inform future applications of the model.

They designed the model in such a way as to recreate
order, claiming that it makes the optimization problem

> *The target sequence is same as the input sequ
sequence makes the optimization easier becau
low range correlations.*

— Unsupervised Learning of Video Representations u

They also explore two approaches to training the decoder model, specifically a version conditioned in the previous output generated by the decoder, and another without any such conditioning.

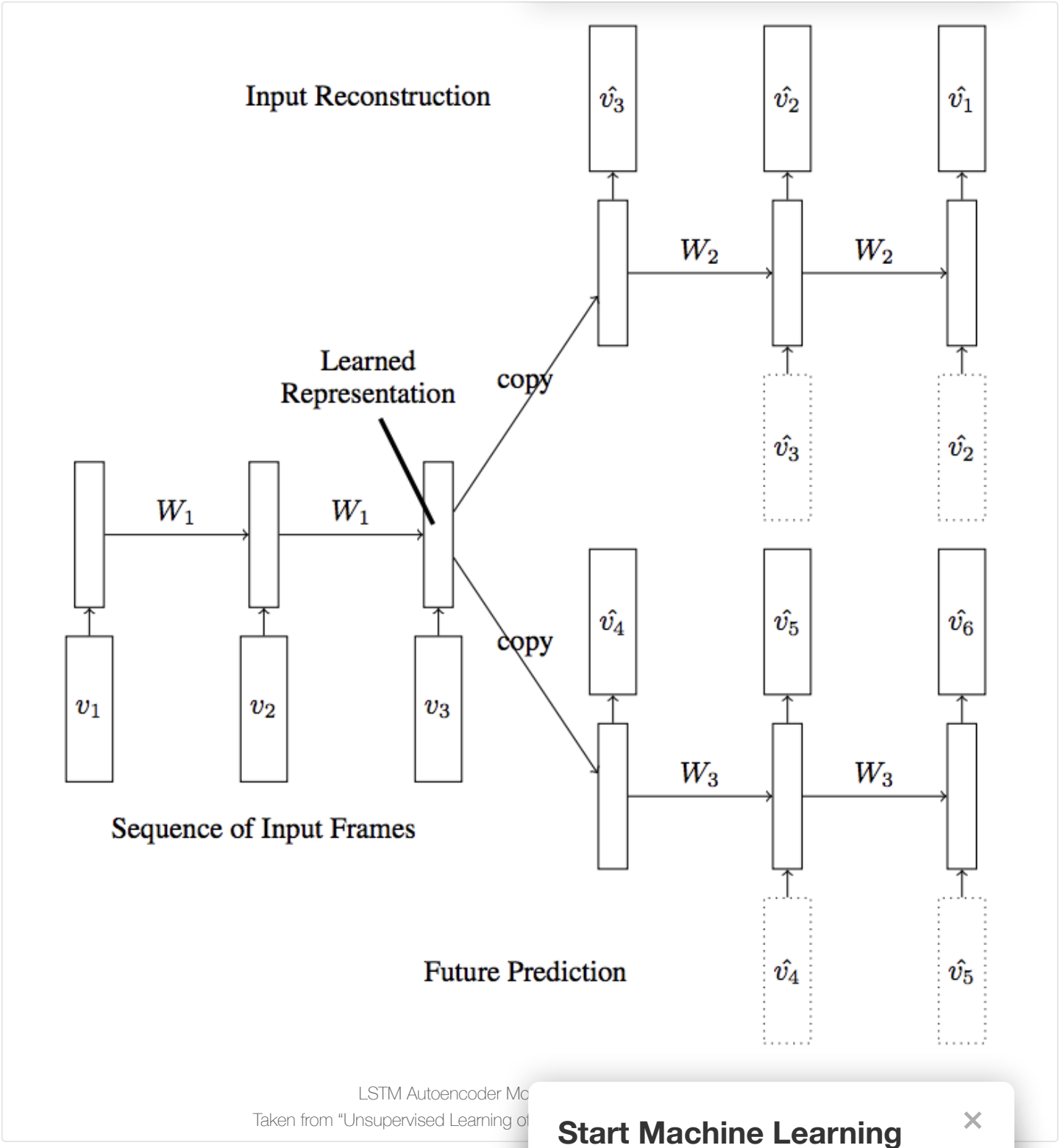
> *The decoder can be of two kinds – conditional or unconditioned. A conditional decoder receives the last generated output frame as input […]. An unconditioned decoder does not receive that input.*

— Unsupervised Learning of Video Representations using LSTMs, 2015.

A more elaborate autoencoder model was also explored where two decoder models were used for the one encoder: one to predict the next frame in the sequence and one to reconstruct frames in the sequence, referred to as a composite model.

> *… reconstructing the input and predicting the future can be combined to create a composite […]. Here the encoder LSTM is asked to come up with a state from which we can both predict the next few frames as well as reconstruct the input.*

— Unsupervised Learning of Video Representations using LSTMs, 2015.



LSTM Autoencoder Mo
Taken from "Unsupervised Learning of

The models were evaluated in many ways, including u
rather than using the output of the encoder as an inpu
LSTM classifier with the weights of the encoder mode
the implementation.

> *We initialize an LSTM classifier with the weight*

— Unsupervised Learning of Video Representations u

The composite model without conditioning on the decoder was found to perform the best in their experiments.

> *The best performing model was the Composite Model that combined an autoencoder and a future predictor. The conditional variants did not give any significant improvements in terms of classification accuracy after fine-tuning, however they did give slightly lower prediction errors.*

— Unsupervised Learning of Video Representations using LSTMs, 2015.

Many other applications of the LSTM Autoencoder have been demonstrated, not least with sequences of text, audio data and time series.

How to Create LSTM Autoencoders in Keras

Creating an LSTM Autoencoder in Keras can be achieved by implementing an Encoder-Decoder LSTM architecture and configuring the model to recreate the input sequence.

Let's look at a few examples to make this concrete.

Reconstruction LSTM Autoencoder

The simplest LSTM autoencoder is one that learns to reconstruct each input sequence.

For these demonstrations, we will use a dataset of one sample of nine time steps and one feature:

```
[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

We can start-off by defining the sequence and reshaping it into the preferred shape of [*samples, timesteps, features*].

```
# define input sequence
sequence = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(sequence)
sequence = sequence.reshape((1, n_in, 1))
```

Next, we can define the encoder-decoder LSTM architecture that expects input sequences with nine time steps and one feature and outputs a sequence with ni

```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_sh
model.add(RepeatVector(n_in))
model.add(LSTM(100, activation='relu', return_s
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
```

Next, we can fit the model on our contrived dataset.

```
# fit model
model.fit(sequence, sequence, epochs=300, verbo
```

The complete example is listed below.

The configuration of the model, such as the number of units and training epochs, was completely arbitrary.

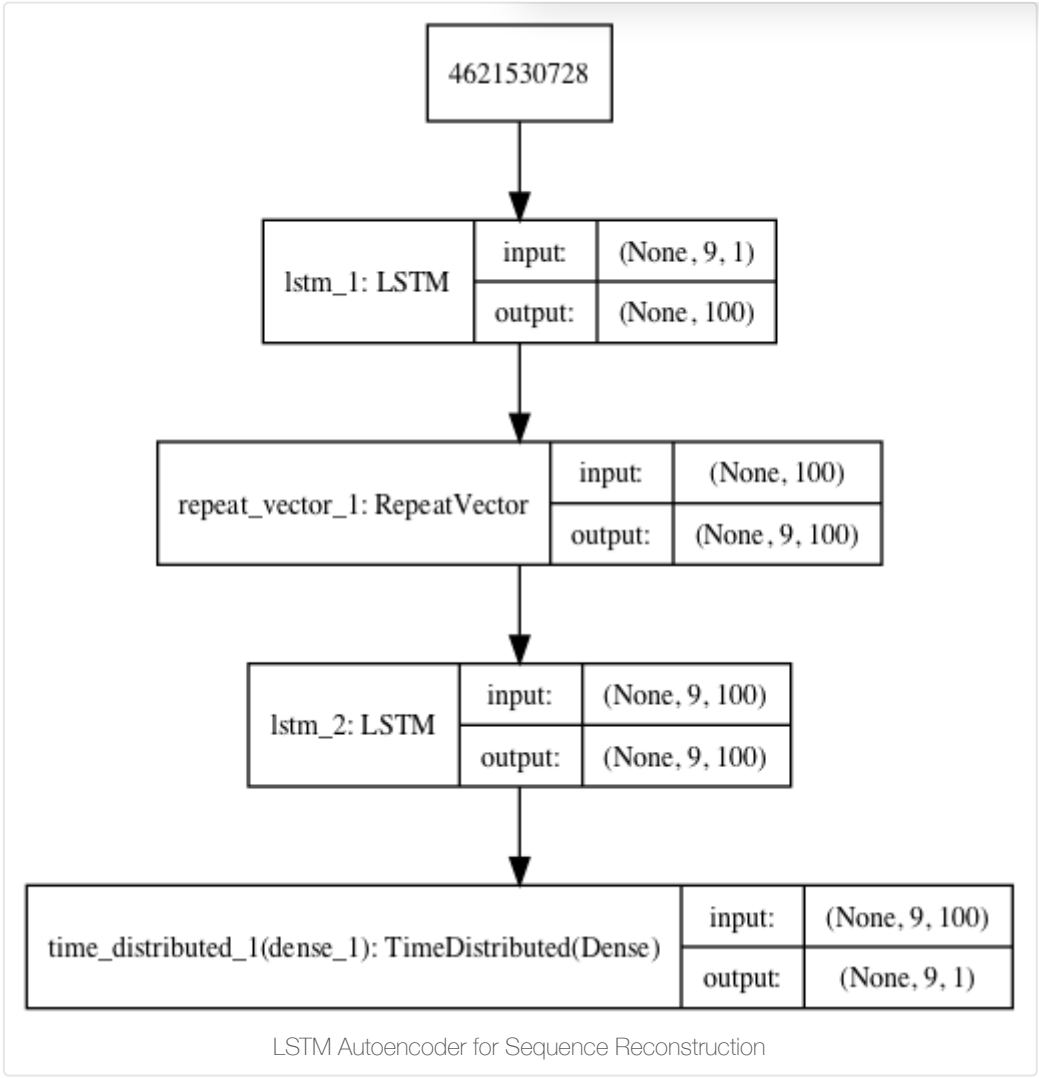
```
# lstm autoencoder recreate sequence
from numpy import array
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.utils import plot_model
# define input sequence
sequence = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(sequence)
sequence = sequence.reshape((1, n_in, 1))
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))
model.add(RepeatVector(n_in))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(sequence, sequence, epochs=300, verbose=0)
plot_model(model, show_shapes=True, to_file='reconstruct_lstm_autoencoder.png')
# demonstrate recreation
yhat = model.predict(sequence, verbose=0)
print(yhat[0,:,0])
```

Running the example fits the autoencoder and prints the reconstructed input sequence.

The results are close enough, with very minor rounding errors.

```
[0.10398503 0.20047213 0.29905337 0.3989646 0.4994707 0.60005534
 0.70039135 0.80031013 0.8997728 ]
```

A plot of the architecture is created for reference.



LSTM Autoencoder for Sequence Reconstruction

Prediction LSTM Autoencoder

We can modify the reconstruction LSTM Autoencoder to instead predict the next step in the sequence.

In the case of our small contrived problem, we expect the output to be the sequence:

```
1 [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

This means that the model will expect each input sequence to have nine time steps and the output sequence to have eight time steps.

```
1 # reshape input into [samples, timesteps, featu
2 n_in = len(seq_in)
3 seq_in = seq_in.reshape((1, n_in, 1))
4 # prepare output sequence
5 seq_out = seq_in[:, 1:, :]
6 n_out = n_in - 1
```

The complete example is listed below.

```
1 # lstm autoencoder predict sequence
2 from numpy import array
3 from keras.models import Sequential
4 from keras.layers import LSTM
```

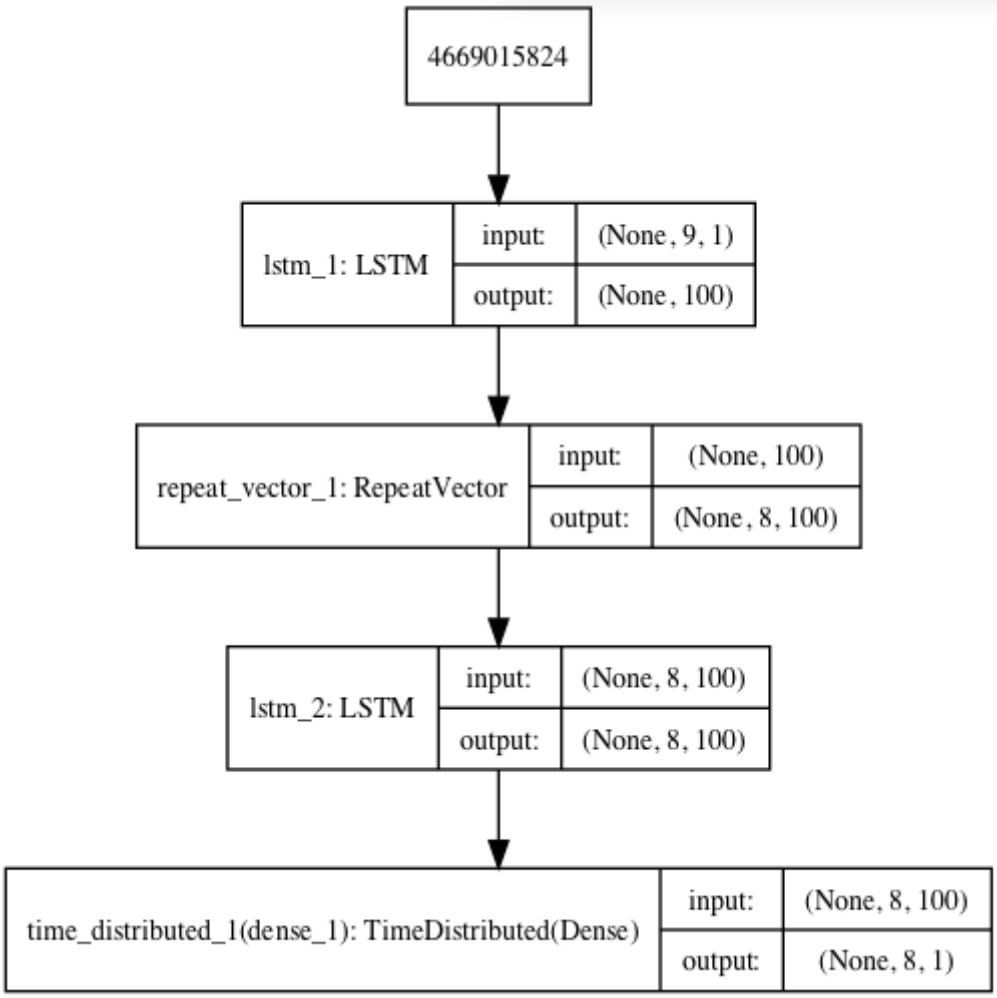
```
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.utils import plot_model
# define input sequence
seq_in = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(seq_in)
seq_in = seq_in.reshape((1, n_in, 1))
# prepare output sequence
seq_out = seq_in[:, 1:, :]
n_out = n_in - 1
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))
model.add(RepeatVector(n_out))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
plot_model(model, show_shapes=True, to_file='predict_lstm_autoencoder.png')
# fit model
model.fit(seq_in, seq_out, epochs=300, verbose=0)
# demonstrate prediction
yhat = model.predict(seq_in, verbose=0)
print(yhat[0,:,0])
```

Running the example prints the output sequence that predicts the next time step for each input time step.

We can see that the model is accurate, barring some minor rounding errors.

```
[0.1657285 0.28903174 0.40304852 0.5096578 0.6104322 0.70671254
 0.7997272 0.8904342 ]
```

A plot of the architecture is created for reference.



LSTM Autoencoder for Sequence Prediction

Composite LSTM Autoencoder

Finally, we can create a composite LSTM Autoencoder that has a single encoder and two decoders, one for reconstruction and one for prediction.

We can implement this multi-output model in Keras using the functional API. You can learn more about the functional API in this post:

- How to Use the Keras Functional API for Deep Learning

First, the encoder is defined.

```
# define encoder
visible = Input(shape=(n_in,1))
encoder = LSTM(100, activation='relu')(visible)
```

Then the first decoder that is used for reconstruction.

```
# define reconstruct decoder
decoder1 = RepeatVector(n_in)(encoder)
decoder1 = LSTM(100, activation='relu', return_
decoder1 = TimeDistributed(Dense(1))(decoder1)
```

Then the second decoder that is used for prediction.

```
# define predict decoder
decoder2 = RepeatVector(n_out)(encoder)
decoder2 = LSTM(100, activation='relu', return_sequences=True)(decoder2)
decoder2 = TimeDistributed(Dense(1))(decoder2)
```

We then tie the whole model together.

```
# tie it together
model = Model(inputs=visible, outputs=[decoder1, decoder2])
```

The complete example is listed below.

```
# lstm autoencoder reconstruct and predict sequence
from numpy import array
from keras.models import Model
from keras.layers import Input
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.utils import plot_model
# define input sequence
seq_in = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(seq_in)
seq_in = seq_in.reshape((1, n_in, 1))
# prepare output sequence
seq_out = seq_in[:, 1:, :]
n_out = n_in - 1
# define encoder
visible = Input(shape=(n_in,1))
encoder = LSTM(100, activation='relu')(visible)
# define reconstruct decoder
decoder1 = RepeatVector(n_in)(encoder)
decoder1 = LSTM(100, activation='relu', return_sequences=True)(decoder1)
decoder1 = TimeDistributed(Dense(1))(decoder1)
# define predict decoder
decoder2 = RepeatVector(n_out)(encoder)
decoder2 = LSTM(100, activation='relu', return_sequences=True)(decoder2)
decoder2 = TimeDistributed(Dense(1))(decoder2)
# tie it together
model = Model(inputs=visible, outputs=[decoder1, decoder2])
model.compile(optimizer='adam', loss='mse')
plot_model(model, show_shapes=True, to_file='composite_lstm_autoencoder.png')
# fit model
model.fit(seq_in, [seq_in,seq_out], epochs=300
# demonstrate prediction
yhat = model.predict(seq_in, verbose=0)
print(yhat)
```

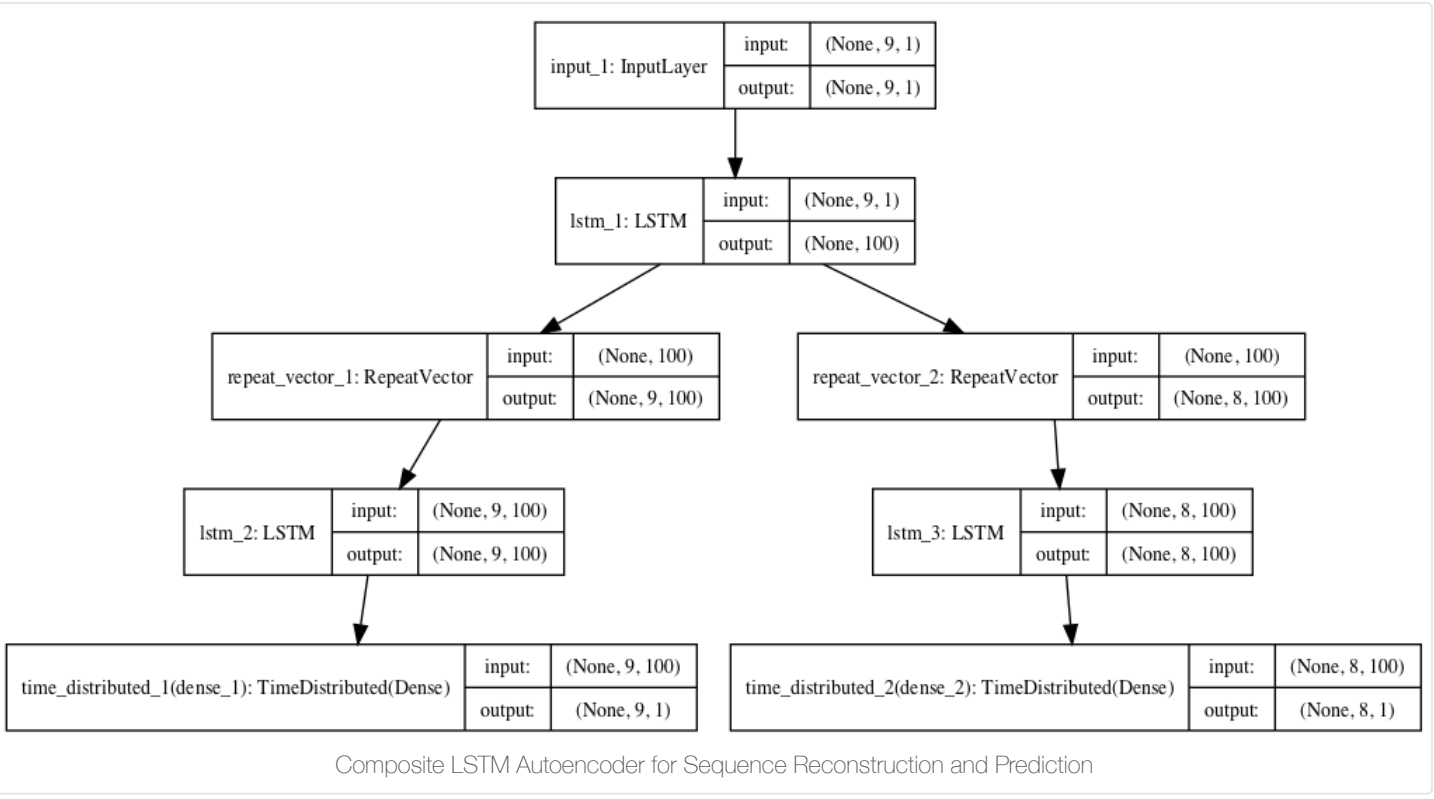
Running the example both reconstructs and predicts

```
[array([[[0.10736275],
[0.20335874],
[0.30020815],
[0.3983948 ],
[0.4985725 ],
[0.5998295 ],
[0.700336 ,
[0.8001949 ],
```

```
 [0.89984304]]], dtype=float32),

array([[[0.16298929],
 [0.28785267],
 [0.4030449 ],
 [0.5104638 ],
 [0.61162543],
 [0.70776784],
 [0.79992455],
 [0.8889787 ]]], dtype=float32)]
```

A plot of the architecture is created for reference.



Composite LSTM Autoencoder for Sequence Reconstruction and Prediction

Keep Standalone LSTM Encoder

Regardless of the method chosen (reconstruction, prediction, or composite), once the autoencoder has been fit, the decoder can be removed and the encoder can be kept as a standalone model.

The encoder can then be used to transform input sequences to a fixed length encoded vector.

We can do this by creating a new model that has the s
directly from the end of encoder model, before the *Re*

```
# connect the encoder LSTM as the output layer
model = Model(inputs=model.inputs, outputs=mode
```

A complete example of doing this with the reconstruc

```
# lstm autoencoder recreate sequence
from numpy import array
from keras.models import Sequential
from keras.models import Model
from keras.layers import LSTM
```

```
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed
from keras.utils import plot_model
# define input sequence
sequence = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(sequence)
sequence = sequence.reshape((1, n_in, 1))
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))
model.add(RepeatVector(n_in))
model.add(LSTM(100, activation='relu', return_sequences=True))
model.add(TimeDistributed(Dense(1)))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(sequence, sequence, epochs=300, verbose=0)
# connect the encoder LSTM as the output layer
model = Model(inputs=model.inputs, outputs=model.layers[0].output)
plot_model(model, show_shapes=True, to_file='lstm_encoder.png')
# get the feature vector for the input sequence
yhat = model.predict(sequence)
print(yhat.shape)
print(yhat)
```

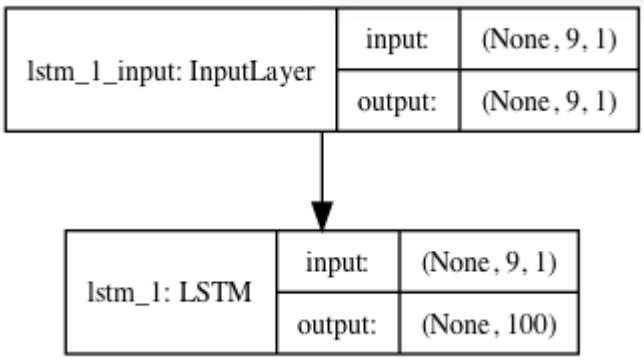
Running the example creates a standalone encoder model that could be used or saved for later use.

We demonstrate the encoder by predicting the sequence and getting back the 100 element output of the encoder.

Obviously, this is overkill for our tiny nine-step input sequence.

```
[[0.03625513 0.04107533 0.10737951 0.02468692 0.06771207 0.
  0.0696108  0.         0.         0.0688471  0.         0.
  0.         0.         0.         0.         0.         0.03871286
  0.         0.         0.05252134 0.         0.07473809 0.02688836
  0.         0.         0.         0.         0.         0.0460703
  0.         0.         0.05190025 0.         0.         0.11807001
  0.         0.         0.         0.         0.         0.
  0.         0.14514188 0.         0.         0.         0.
  0.02029926 0.02952124 0.         0.         0.         0.
  0.         0.08357017 0.08418129 0.         0.         0.
  0.         0.         0.09802645 0.07694854 0.         0.03605933
  0.         0.06378153 0.         0.05267526 0.02744672 0.
  0.06623861 0.         0.         0.         0.08133873 0.09208347
  0.03379713 0.         0.         0.         0.07517676 0.08870222
  0.         0.         0.         0.
  0.08123557 0.         0.08983088 0.0886112
  0.00616016 0.0620428  0.         0.
```

A plot of the architecture is created for reference.



Standalone Encoder LSTM Model

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Making Predictions with Sequences
- Encoder-Decoder Long Short-Term Memory Networks
- Autoencoder, Wikipedia
- Unsupervised Learning of Video Representations using LSTMs, ArXiv 2015.
- Unsupervised Learning of Video Representations using LSTMs, PMLR, PDF, 2015.
- Unsupervised Learning of Video Representations using LSTMs, GitHub Repository.
- Building Autoencoders in Keras, 2016.
- How to Use the Keras Functional API for Deep Learning

Summary

In this post, you discovered the LSTM Autoencoder model and how to implement it in Python using Keras.

Specifically, you learned:

- Autoencoders are a type of self-supervised learning model that can learn a compressed representation of input data.
- LSTM Autoencoders can learn a compressed representation of sequence data and have been used on video, text, audio, and time series sequence data.
- How to develop LSTM Autoencoder models in Py

Do you have any questions?
Ask your questions in the comments below and I will

About Jason Brownlee

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

View all posts by Jason Brownlee →

‹ LSTM Model Architecture for Rare Event Time Series Forecasting

How to Use the TimeseriesGenerator for Time Series Forecasting in Keras ›

212 Responses to *A Gentle Introduction to LSTM Autoencoders*

samaksh kumar November 5, 2018 at 11:35 am #

REPLY

Nice Explained…….

Jason Brownlee November 5, 2018 at 2:27 pm

Thanks.

ramar October 11, 2019 at 8:31 pm #

Very Good and Detailed representation of LSTM.

I have a csv file which contains 3000 values, when i run it in Google colab or jupyter notebook it was much slow What may be the reason?

REPLY

Jason Brownlee October 12, 2019 at 6:56 am #

Thanks.

Perhaps try running on a faster machine, like EC2?
Perhaps try using a more efficient implementation?
Perhaps try using less training data?

REPLY

Ali Alwehaibi November 6, 2018 at 8:16 am #

Thanks for the great posts! I have learn a lot from them.
Can this approach for classification problems such as sentiment analysis?

REPLY

Jason Brownlee November 6, 2018 at 2:16 pm #

Perhaps.

REPLY

TJ Chen November 7, 2018 at 1:21 pm #

Hi Jason,
Thanks for the posts, I really enjoy reading this.
I'm trying to use this method to do time series data anomaly detection and I got few questions here:
When you reshape the sequence into [samples, timesteps, features], samples and features always equal to 1. What is the guidance to choose the value here? If the input sequences have variable length, how to set timesteps, always choose max length?

Also, if the input is two dimension tabular data with ea
reshape or normalization?
Thanks in advance!

Jason Brownlee November 7, 2018 at 2:49 pr

The time steps should provide enough his
observations recorded at each time step.

More on preparing data for LSTMs here:
https://machinelearningmastery.com/faq/single-faq/how-do-i-prepare-my-data-for-an-lstm

Soheila November 8, 2018 at 5:52 pm #

REPLY

Hi,

I am wondering why the output of encoder has a much higher dimension(100), since we usually use encoders to create lower dimensions!

Could you please bring examples if I am wrong?

And what about variable length of samples? You keep saying that LSTM is useful for variable length. So how does it deal with a training set like:

dataX[0] = [1,2,3,4]
dataX[1] = [2,5,7,8,4]
dataX[2] = [0,3]

I am really confused with my second question and I'd be very thankful for your help! 🙂

Jason Brownlee November 9, 2018 at 5:19 am #

REPLY

The model reproduces the output, e.g. a 1D vector with 9 elements.

You can pad the variable length inputs with 0 and use a masking layer to ignore the padded values.

SB December 25, 2018 at 5:31 pm #

REPLY

"I am wondering why the output of encoder has a much higher dimension(100), since we usually use encoders to create lower dimensions!", I have the same question, can you please explain more?

Jason Brownlee December 26, 201

It is a demonstration of the archite
configuration for your specific problem.

J.V May 14, 2019 at 7:10 am #

Great article. But reading through it I thought you were tackling the most important problem with sequences – that is they have variable lengths. Turns out it wasn't. Any chance you could write a tutorial on using a mask to neutralise the padded value? This seems to be more difficult than the rest of the model.

Jason Brownlee May 14, 2019 at 7:54 am #

REPLY

Yes, I believe I have many tutorials on the topic.

Perhaps start here:
https://machinelearningmastery.com/handle-missing-timesteps-sequence-prediction-problems-python/

Ufanc November 9, 2018 at 6:17 pm #

REPLY

I really likes your posts and they are important.I got a lot of knowledge from your post. Today, am going to ask your help. I am doing research on local music classifications. the key features of the music is it sequence and it uses five keys out of the seven keys, we call it scale.

1. C – E – F – G – B. This is a major 3rd, minor 2nd, major 2nd, major 3rd, and minor 2nd
2. C – Db – F – G – Ab. This is a minor 2nd, major 3rd, major 2nd, minor 2nd, and major 3rd.
3. C – Db – F – Gb – A. This is a minor 2nd, major 3rd, minor 2nd, minor 3rd, and a minor 3rd.
4. C – D – E – G – A. This is a major 2nd, major 2nd, minor 3rd, major 2nd, and a minor 3rd

it is not dependent on range, rythm, melody and other features.

This key has to be in order. Otherwise it will be out of scale.

So, which tools /algorithm do i need to use for my research purpose and also any sampling mechanism to take 30 sec sample music from each track without affecting the sequence of the keys ?

Regards

Jason Brownlee November 10, 2018 at 5:59 a

REPLY

Perhaps try a suite of models and discove

More here:
https://machinelearningmastery.com/faq/single-fa

lungen November 9, 2018 at 7:43 pm #

Hi, can you please explain the use of repeat v
Encoder is encoding 1-feature time-series into fixed le

should take this 100-length vector and transform it into 1-feature time-series.
So, encoder is like many-to-one lstm, and decoder is one-to-many (even though that 'one' is a vector of length 100). Is this understanding correct?

Jason Brownlee November 10, 2018 at 6:01 am #

REPLY

The RepeatVector repeats the internal representation of the input n times for the number of required output steps.

Abraham May 10, 2019 at 4:02 am #

REPLY

Hi Jason?
What is the intuition behind "representing of the input n times for the number of required output steps?"Here n times denotes, let say as in simple LSTM AE, 9 i.e. output step number.
I understand from repeatvector that here sequence are being read and transformed into a single vector(9×100) which is the same 100 dim vector, then the model uses that vector to reconstruct the original sequence.Is it right?
What about using any number except for 9 for the number of required output steps?
Thanks from now on.

Jason Brownlee May 10, 2019 at 8:19 am #

REPLY

To provide input for the LSTM on each output time step for one sample.

rekha November 10, 2018 at 4:10 am #

REPLY

Which model is most suited for stock market prediction

Jason Brownlee November 10, 2018 at 6:10 a

None, a time series of prices is a random

More here:
https://machinelearningmastery.com/faq/single-fa
finance-or-the-stock-market

MJ November 10, 2018 at 4:41 pm #

REPLY

Hi,

thanks for the instructive post!

I am trying to repeat your first example (Reconstruction LSTM Autoencoder) using a different syntax of Keras; here is the code:

```
import numpy as np
from keras.layers import Input, LSTM, RepeatVector
from keras.models import Model

timesteps = 9
input_dim = 1
latent_dim = 100

# input placeholder
inputs = Input(shape=(timesteps, input_dim))

# “encoded” is the encoded representation of the input
encoded = LSTM(latent_dim,activation=’relu’)(inputs)

# “decoded” is the lossy reconstruction of the input
decoded = RepeatVector(timesteps)(encoded)
decoded = LSTM(input_dim, activation=’relu’, return_sequences=True)(decoded)

sequence_autoencoder = Model(inputs, decoded)
encoder = Model(inputs, encoded)

# compile model
sequence_autoencoder.compile(optimizer=’adadelta’, loss=’mse’)

# run model
sequence_autoencoder.fit(sequence,sequence,epochs=300, verbose=0)

# prediction
sequence_autoencoder.predict(sequence,verbose=0)
```

I did not know why, but I always get a poor result than the model using your code.
So my question is: is there any difference between the two method (syntax) under the hood? or they are actually the same ?

Thanks.

Jason Brownlee November 11, 2018 at 5:58 a

If you have trouble with the code in the tu
higher and TensorFlow is up to date.

J Hogue November 29, 2018 at 5:01 am #

REPLY

I feel like a bit more description could go into how to setup the LSTM autoencoder. Particularly how to tune the bottleneck. Right now when I apply this to my data its basically just returning the mean for everything, which suggests the its too aggressive but I'm not clear on where to change things.

Jason Brownlee November 29, 2018 at 7:48 am #

REPLY

Thanks for the suggestion.

Dimitre Oliveira December 10, 2018 at 11:59 am #

REPLY

Hi Jason, thanks for the wonderful article, I took some time and wrote a kernel on Kaggle inspired by your content, showing regular time-series approach using LSTM and another one using a MLP but with features encoded by and LSTM autoencoder, as shown here, for anyone interested here's the link: https://www.kaggle.com/dimitreoliveira/time-series-forecasting-with-lstm-autoencoders

I would love some feedback.

Jason Brownlee December 10, 2018 at 2:17 pm #

REPLY

Well done!

Simranjit Singh December 27, 2018 at 10:53 pm #

REPLY

Hey! I am trying to compact the data single row of 217 rows. After running the program it is returning nan values for prediction Can you guide me where did i do wrong?

Jason Brownlee December 28, 2018 at 5:57 a

I have some advice here:
https://machinelearningmastery.com/faq/single-fa

Net December 31, 2018 at 9:32 pm #

Dear Jason
After building and training the model above, how to evaluate the model? (like model.evaluate(train_x, train_y…) in common LSTM)?

Thanks a lot

Jason Brownlee January 1, 2019 at 6:15 am #

REPLY

The model is evaluated by its ability to reconstruct the input. You can use evaluate function or perform the evaluation of the predictions manually.

Andy Hung January 2, 2019 at 6:28 am #

REPLY

I have learned a lot from your website. Autoencoder can be used as dimension reduction. Is it possible to merge multiple time-series inputs into one using RNN autoencoder? My data shape is (9500, 20, 5) => (sample size, time steps, features). How to encode-decode into (9500, 20, 1)?

Thank you very much,

Jason Brownlee January 2, 2019 at 6:44 am #

REPLY

Perhaps, that may require some careful design. It might be easier to combine all data to a single input.

Andy Hung January 3, 2019 at 3:47 am #

REPLY

Thank you for your reply. Will (9500,100,1) => (9500,20,1) be easier?

Jason Brownlee January 3, 2019 at 6:14 am #

Perhaps test it and see.

Jimmy Joe January 5, 2019 at 9:41 am #

Hi Jason,
I'm a regular reader of your website, I learned a lot fro
This one is also very informative, but there's one thing

0.2, …, 0.9] and the expected decoder output is [0.2, 0.3, …, 0.9], that's basically a part of the input sequence. I'm not sure why you say it's "predicting next step for each input step". Could you please explain? Is an autoencoder a good fit for multi-step time series prediction?
Another question: does training the composite autoencoder imply that the error is averaged for both expected outputs ([seq_in, seq_out])?

Jason Brownlee January 6, 2019 at 10:15 am #

REPLY

I am demonstrating two ways to learn the encoding, by reproducing the input and by predicting the next step in the output.

Remember, the model outputs a single step at a time in order to construct a sequence.

Good question, I assume the reported error is averaged over both outputs. I'm not sure.

Junetae Kim January 27, 2019 at 4:12 pm #

REPLY

Hi, I am JT.
First of all, thanks for your post that provides an excellent explanation of the concept of LSTM AE models and codes.

If I understand your AE model correclty, features from your LSTM AE vector layer [shape (,100)] does not seem to be time dependent.

So, I have tried to build a time-dependent AE layer by modifying your codes.

Could you check my codes whether my codes are correct to build an AE model that incpude a time-wise AE layer, if you don't mind?

My codes are below.

```
from numpy import array
from keras.models import Model
from keras.layers import Input
from keras.layers import LSTM
from keras.layers import Dense
from keras.layers import RepeatVector
from keras.layers import TimeDistributed

## Data generation
# define input sequence
seq_in = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
# reshape input into [samples, timesteps, features]
n_in = len(seq_in)
seq_in = seq_in.reshape((1, n_in, 1))
# prepare output sequence
```

```
seq_out = array([3, 5, 7, 9, 11, 13, 15, 17, 19])
seq_out = seq_out.reshape((1, n_in, 1))

## Model specification
# define encoder
visible = Input(shape=(n_in,1))
encoder = LSTM(60, activation='relu', return_sequences=True)(visible)

# AE Vector
AEV = LSTM(30, activation='relu', return_sequences=True)(encoder)

# define reconstruct decoder
decoder1 = LSTM(60, activation='relu', return_sequences=True)(AEV)
decoder1 = TimeDistributed(Dense(1))(decoder1)

# define predict decoder
decoder2 = LSTM(30, activation='relu', return_sequences=True)(AEV)
decoder2 = TimeDistributed(Dense(1))(decoder2)

# tie it together
model = Model(inputs=visible, outputs=[decoder1, decoder2])
model.summary()
model.compile(optimizer='adam', loss='mse')

# fit model
model.fit(seq_in, [seq_in,seq_out], epochs=2000, verbose=2)

## The model that feeds seq_in to predict seq_out
hat1= model.predict(seq_in)

## The model that feeds seq_in to predict AE Vector values
model2 = Model(inputs=model.inputs, outputs=model.layers[2].output)
hat_ae= model2.predict(seq_in)

## The model that feeds AE Vector values to predict seq_out
input_vec = Input(shape=(n_in,30))
dec2 = model.layers[4](input_vec)
dec2 = model.layers[6](dec2)
model3 = Model(inputs=input_vec, outputs=dec2)
hat_= model3.predict(hat_ae)
```

Thank you very much

Jason Brownlee January 28, 2019 at 7:11 am

I'm happy to answer questions, but I don'
sorry.

Anirban Ray January 28, 2019 at 5:33 pm #

REPLY

Thanks for the nice post. Being a beginner in machine learning, your posts are really helpful.

I want to build an auto-encoder for data-set of names of a large number of people. I want to encode the entire field instead of doing it character or wise, for example [“Neil Armstrong”] instead of [“N”, “e”, “i”, “l”, ” “, “A”, “r”, “m”, “s”, “t”, “r”, “o”, “n”, “g”] or [“Neil”, “Armstrong”]. How can I do it?

Jason Brownlee January 29, 2019 at 6:09 am #

REPLY

Wrap your list of strings in one more list/array.

Benjamin February 15, 2019 at 11:05 am #

REPLY

Hey, thanks for the post, I have found it helpful… Although I am confused about one, in my opinion, major point..

– If autoencoders are used to obtain a compressed representation of the input, what is the purpose of taking the output after the encoder part if it is now 100 elements instead of 9? I’m struggling to find a meaning of the 100 element data and how one could use this 100 element data to predict anomalies. It sort of seems like doing the exact opposite of what was stated in the explanation prior to the example. An explanation would be greatly appreciated.

– In the end I’m trying to really understand how after learning the weights by minimizing the reconstruction error of the training set using the AE, how to then use this trained model to predict anomalies in the cross validation and test sets.

–

Jason Brownlee February 15, 2019 at 2:22 pm #

REPLY

It is just a demonstration, perhaps I could

For example, you could scale up the input to be 1,

Ahmad May 19, 2019 at 10:22 pm #

Hi Jason, Benjamin is right. The last e
encoder. The input sequence is 9 elements bu
explaining in the first part of the tutorial that er
be used as a feature vector. I am also confuse

as a feature representation of 9 elements of the input sequence. A more detail explanation will help. Thank you!

Jason Brownlee May 20, 2019 at 6:29 am #

REPLY

Thanks for the suggestion.

Cloudy February 17, 2019 at 7:51 pm #

REPLY

Thank your great post.
As you mentioned in the first section, “Once fit, the encoder part of the model can be used to encode or compress sequence data that in turn may be used as a feature vector input to a supervised learning model”. I fed the feature vector (encode part) to 1 feedforward neural network 1 hidden layer: n_dimensions=50

```
def get_model(n_dimensions):
    inputs = Input(shape=(timesteps, input_dim))
    encoded = LSTM(n_dimensions, return_sequences=False, name="encoder")(inputs)
    decoded = RepeatVector(timesteps)(encoded)
    decoded = LSTM(input_dim, return_sequences=True, name='decoder')(decoded)
    decoded = TimeDistributed(Dense(features_n))(decoded)

    autoencoder = Model(inputs, decoded)
    encoder = Model(inputs, encoded)
    mid = Dense(num_units, activation='relu')(encoded) #FFNN (1in, 1hid, 1out)
    out = Dense(num_classes, activation='softmax')(mid) #FFNN
    full_model=Model(inputs, out)
     return autoencoder, encoder, full_model
autoencoder, encoder, full_model = get_model(n_dimensions)
history = autoencoder.fit(train_x, train_x, batch_size=100, epochs=epochs, validation_dat
train_encoded = encoder.predict(train_x)
val_encoded = encoder.predict(train_x)
history_class=full_model.fit(train_encoded, train_y, epochs=3, batch_size=256, validation_
```

Error when fit(): ValueError: Error when checking input: expected input_1 to have 3 dimensions, but got array with shape (789545, 50).

I mix Autoencoder to FFNN and is my method, right? Can you help me shape the feature vector before fed to FFNN

Jason Brownlee February 18, 2019 at 6:30 ar

Change the LSTM to not return sequence

Cloudy February 19, 2019 at 6:14 pm #

Thank for your reply soon.
I saw your post, LSTM layer at the decoder is set “return_sequences=True” and I follow and then error as you saw. Actually, I thought the decoder is not a stacked LSTM (only 1 LSTM layer), so “return_sequences=False” is suitable. I changed as you recommend. Another error:
decoded = TimeDistributed(Dense(features_n))(decoded)
File “/usr/local/lib/python3.4/dist-packages/keras/engine/topology.py”, line 592, in __call__
self.build(input_shapes[0])
File “/usr/local/lib/python3.4/dist-packages/keras/layers/wrappers.py”, line 164, in build
assert len(input_shape) >= 3
AssertionError.
Can you give me an advice?
Thank you

Jason Brownlee February 20, 2019 at 7:52 am #

REPLY

I’m not sure about this error, sorry. Perhaps post code and error to stackoveflow or try debugging?

Cloudy February 22, 2019 at 7:45 pm #

Hi Jason,
I found another way to build full_model. I don’t use autoencoder.predict(train_x) to input to full_model. I used orginal inputs, saved weights of the encoder part in autoencoder model, then set that weights to encoder model. Something like this:
autoencoder.save_weights(‘autoencoder.h5’)
for l1,l2 in zip(full_model.layers[:a],autoencoder.layers[0:a]): #a:the num_layer in the encoder part
l1.set_weights(l2.get_weights())
train full_model:
full_model.compile(loss=’binary_crossentropy’, optimizer=’adam’, metrics=[‘accuracy’])
history_class=full_model.fit(train_x, train_y, epochs=2, batch_size=256, validation_data=(val_x, val_y))

My full_model run, but the result so ba

Jason Brownlee February 23,

Interesting, sounds like more

Anshuman Singh Bhadauria February 21, 201

Hi Jason,

Thank you for putting in the effort of writing the posts, they are very helpful.

Is it possible to learn representations of multiple time series at the same time? By multiple time-series I don't mean multivariate.

For eg., if I have time series data from 10 sensors, how can I feed them simultaneously to obtain 10 representations, not a combined one.

Best,
Anshuman

Jason Brownlee February 21, 2019 at 8:13 am #

REPLY

Yes, each series would be a different sample used to train one model.

JasOlean March 5, 2019 at 9:42 pm #

REPLY

Hi Jason,

I use MinMaxScaler function to normalize my training and testing data.

After that I did some training process to get model.h5 file.

And then I use this file to predict my testing data.

After that I got some prediction results with range (0,1).
I reverse my original data using inverse_transform function from MinMaxScaler.

But, when I compare my original data (before scaler) with my predictions data, the x,y coordinates are changed like this:

Ori_data =
[7.6291,112.74,43.232,96.636,61.033,87.311,91.55,115.28,121.22,136.48,119.52,80.53,172.08,77.987,199.21,94.94,228.03,110.2,117.83,104.26,174.62,103.42,211.92,109.35,204.29,122.91,114.44,125.46,168.69,124.61,194.97,134.78,173.77,141.56,104.26,144.11,125.46,166.99,143.26,185.64,165.3,205.14]

Predicted_data = [94.290375, 220.07372, 112.91617,
161.85602, 99.74797, 178.18903, 60.718987, 86.0122
111.641655, 90.18026, 134.16464, 82.28861, 155.125
99.82883, 145.162, 98.78825, 98.62861, 130.25414, 6
143.52762, 74.574684, 99.36809, 169.79303, 107.395
124.29078, 114.974014, 135.11014, 107.4492, 90.644
121.55309, 174.63484, 138.58575, 167.6933, 144.915

When I visualize these predictions data on my image,
is horizontal but predictions data is vertical).
Why I face this and how can I fix that?

Jason Brownlee March 6, 2019 at 7:54 am #

REPLY

You must ensure that the columns match when calling transform() and inverse_transform().

See this tutorial:
https://machinelearningmastery.com/machine-learning-data-transforms-for-time-series-forecasting/

saria March 11, 2019 at 6:09 am #

REPLY

Hi Jason,

Thank you so much for your great post. I wish you have done this with a real data set like 20 newsgroup data set.
It is at first not clear the different ways of preparing the data for different objectives.
My understanding is that with LSTM Autoencoder we can prepare data in different ways based on the goal. Am I correct?
Or can you please give me the link which is preparing the text data like 20 news_group for this kind of model?

Again thanks for your awesome material

Jason Brownlee March 11, 2019 at 6:58 am #

REPLY

If you are working with text data, perhaps start here:
https://machinelearningmastery.com/start-here/#nlp

saria March 11, 2019 at 3:06 pm #

REPLY

Thank you so much Jason for the link. I have already gone through lots of material, in detail the mini corse in the mentioned link months ag
My problem mainly is the label data here.

For example, in your code, in the reconstructi
label. however, in the prediction part you have
and their difference is that seq_out looking at

My question according to your example will be
purpose of topic modeling, Do I need to follow
prediction?

saria March 12, 2019 at 1:31 am #

REPLY

I think I got my answer. Thanks Jason 🙂

Jason Brownlee March 12, 2019 at 6:43 am #

REPLY

No. Perhaps this model would be more useful as a starting point: https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

saria March 11, 2019 at 6:35 am #

REPLY

Based on different objectives I meant, for example if we use this architecture for topic modeling, or sequence generation, or … is preparing the data should be different?

Mingkuan Wu March 14, 2019 at 9:27 am #

REPLY

Thanks for your post! When you use RepeatVectors(), I guess you are using the unconditional decoder, am I right?

Jason Brownlee March 14, 2019 at 9:32 am #

REPLY

I guess so. Are you referring to a specific model in comparison?

rekha March 18, 2019 at 4:47 am #

REPLY

Thanks for the post. Can this be formulated a

Jason Brownlee March 18, 2019 at 6:08 am #

The demonstration is a sequence predicti

Kristian March 20, 2019 at 11:32 pm #

Hi Jason,

what a fantastic tutorial!

I have a question about the loss function used in the composite model.
Say you have different loss functions for the reconstruction and the prediction/classification parts, and pre-trains the reconstruction part.

In Keras, would it be possible to combine these two loss functions into one when training the model, such that the model does not lose or diminish its reconstruction ability while traning the prediction/classification part?

If so; could you please point me in the right direction.

Kind regards

Kristian

Jason Brownlee March 21, 2019 at 8:16 am #

REPLY

Yes, great question!

You can specify a list of loss functions to use for each output of the network.

El March 26, 2019 at 4:54 am #

REPLY

Dear,

Would it make sense to set statefull = true on the LSTMs layers of an encoder decoder?

Thanks

Jason Brownlee March 26, 2019 at 8:12 am #

REPLY

It really depends on whether you want control over when the internal state is reset, or not.

---

saria March 28, 2019 at 3:34 am #

Thank you, Jason, but still, I have not got the
Lets put it another way. what is the latent space in this
data?
do you think if I use the architecture of Many to one, I
of data?

Why am I able to print out the clusters of the topics in autoencoder easily but when it comes to this architecture I am lost!

Jason Brownlee March 28, 2019 at 8:22 am #

REPLY

In some sense, yes, but a one value representation is an aggressive projection/compression of the input and may not be useful.

What problem are you having exactly?

JohnAlex March 28, 2019 at 2:01 pm #

REPLY

Hi Jason,
I am appreciate your tutorial!
Now I'm implementing the paper '"Unsupervised Learning of Video Representations using LSTMs."But my result is not very well.The predict pictures are blurred,not good as the paper's result.
（You can see my result at here:
https://i.loli.net/2019/03/28/5c9c374d68af2.jpg
https://i.loli.net/2019/03/28/5c9c37af98c65.jpg）

I don't think there exists difference between my keras model and the paper's model.But the problem has confused me for 2 weeks,I can not get a good solution.I really appreciate your help!

This my keras model's code:

```
Input_seqlen=10
Output_seqlen=10
Pic_size=64
Channels=1
Num_units = 2048

def Basic_Encoder_Decoder(input_img_shape=(Input_seqlen,Pic_size,Pic_size,Channels)):

    encoder_x = keras.layers.Input(shape=input_img_shape[1:])
    encoder_flatten = keras.layers.Flatten()(encoder_x)
    conv_model=keras.Model(encoder_x,encoder_flatten)

    decode_in = keras.layers.Input(shape=(N
    decode_dense=keras.layers.Dense(Pic_siz
    decode_reshape = keras.layers.Reshape(t
    decode_model=keras.Model(decode_in,decc


    #TensorShape([Dimension(None), Dimensic
    Encoder_inp = keras.layers.Input(shape=
    Encode_seq = keras.layers.TimeDistribut


    Encoder_Lstm,Encoder_h,Encoder_c= keras
                                      retu

    copylayer= keras.layers.RepeatVector(Ou
    decoder_lstm,_,_= keras.layers.LSTM(uni
```

```
        return_state=True,dropout=0.2,activation='relu')(copylayer)
    Decode_seq = keras.layers.TimeDistributed(decode_model)(decoder_lstm)

    cae = keras.Model(Encoder_inp,Decode_seq)

    rms = keras.optimizers.RMSprop(lr=0.001)

    def ae_loss(inp,outp):
        inp = K.flatten(inp)
        outp = K.flatten(outp)
        xent_loss = keras.losses.binary_crossentropy(inp, outp)
        return xent_loss
    cae.compile(optimizer=rms,loss=ae_loss)
```

Jason Brownlee March 28, 2019 at 2:43 pm #

REPLY

Sounds like a great project!

Sorry, I don't have the capacity to debug your code, I have some suggestions here though: https://machinelearningmastery.com/faq/single-faq/can-you-read-review-or-debug-my-code

JohnAlex March 28, 2019 at 10:33 pm #

REPLY

Thank for your reply.
And I wanna know that what may cause the image of the output to be blurred according to your experience ?Thank you~

Jason Brownlee March 29, 2019 at 8:34 am #

REPLY

In what context exactly?

Birish April 5, 2019 at 2:26 am #

REPLY

How can I use the cell state of this "Standalor
another model? Suppose in your code for "Keep Stan
option for the encoder LSTM layer and create the moc

model = Model(inputs=model.inputs, outputs=[model.

Then one can retrieve the cell state by: model.outputs

The problem is that this will return a "Tensor" and kera
input for 'Model()'. How can I feed this cell state to and

Jason Brownlee April 5, 2019 at 6:20 am #

REPLY

I think it would be odd to use cell state as an input, I'm not sure I follow what you want to do.

Nevertheless, you can use Keras to evaluate the tensor, get the data, create a numpy array and provide it as input to the model.

Also, this may help:
https://machinelearningmastery.com/return-sequences-and-return-states-for-lstms-in-keras/

Birish April 5, 2019 at 4:21 pm #

REPLY

That's the approach used in this paper: https://arxiv.org/pdf/1709.01907.pdf

"After the encoder-decoder is pre-trained, it is treated as an intelligent feature-extraction blackbox. Specifically, the last LSTM cell states of the encoder are extracted as learned embedding. Then, a prediction network is trained to forecast the next one or more timestamps using the learned embedding as features."

They trained an LSTM autoencoder and fed the last cell states of last encoder layer to another model. Did I misunderstand it?

Jason Brownlee April 6, 2019 at 6:40 am #

REPLY

Sounds odd, perhaps confirm with the authors that they are not referring to hidden states (outputs) instead?

George April 11, 2019 at 1:07 am #

REPLY

Hello Jason,

Is there any way to stack the LSTM autoencoder?

for example:

```
model = Sequential()
model.add(LSTM(units, activation=activation, input_sh
model.add(RepeatVector(n_in))
model.add(LSTM( units/2, activation=activation))
model.add(RepeatVector(n_in)
model.add(LSTM(units/2, activation=activation,return_
model.add(LSTM(units, activation=activation, return_s
model.add(TimeDistributed(Dense(d)))
```

is this a correct approach?

Do you see any benefits by stacking the autoencoder?

Jason Brownlee April 11, 2019 at 6:43 am #

REPLY

I have never seen something like this 🙂

Leland Hepworth September 14, 2019 at 5:53 am #

REPLY

Hi George,

Stacked encoder / decoders with a narrowing bottleneck are used in a tutorial on the Keras website in the section “Deep autoencoder”

https://blog.keras.io/building-autoencoders-in-keras.html

The tutorial claims that the deeper architecture gives slightly better results than the more shallow model definition in the previous example. This tutorial uses simple dense layers in its models, so I wonder if something similar could be done with LSTM layers.

Jason Brownlee September 14, 2019 at 6:25 am #

REPLY

Thanks for sharing.

Rojin April 12, 2019 at 3:36 am #

REPLY

I have a theoretical question about autoencoders. I know that autoencoders are suppose to construct the input at the output, and by doing so they will learn a lower-dim representation of the input. Now I want to know if it is possible to use autoencoders to construct something else at the output (let’s say a something that is a modified version of the input).

Jason Brownlee April 12, 2019 at 7:53 am #

Sure.

Perhaps check out conditional generative models,

Rojin April 12, 2019 at 10:35 am #

Thanks for the response. I will check those out. I though about denoising autoencoders, but was not sure if that is applicable to my situations.

Let's say that I have two versions of a feature vector, one is X, and the other one is X', which has some meaningful noise (technically not noise, meaningful information). Now my question is whether it is appropriate to use denoising autoencoders in this case to learn about the transition between X to X' ?

Jason Brownlee April 12, 2019 at 2:43 pm #

REPLY

Conditional GANs do this for image to image translation.

Nick April 16, 2019 at 4:18 pm #

REPLY

Hi Jason, could you explain the difference between RepeatVector and return_sequence?
It looks like they both repeat vector several times but what's the difference?
Can we only use return_sequence in the last LSTM encoder layer and don't use RepeatVector before the first LSTM decoder layer?

Jason Brownlee April 17, 2019 at 6:53 am #

REPLY

Yes, they are very different in operation, but similar in effect.

The "return_sequence" argument, returns the LSTM layer outputs for each input time step.

The "RepeatVector" layer copies the output from the LSTM for the last input time step and repeats it n times.

Nick April 18, 2019 at 3:44 am #

REPLY

Thank you, Jason, now I understand t
question, can we do like this:
"'
encoder = LSTM(100, activation='relu', input_s
(no RepeatVector layer here, but return_seque
decoder = LSTM(100, activation='relu', return_
decoder = TimeDistributed(Dense(1))(decoder)
"'

If yes, what's the difference between this one a
between encoder and decoder, but return_seq

Jason Brownlee April 18, 2019 at 8:54 am #

REPLY

The repeat vector allows the decoder to use the same representation when creating each output time step.

A stacked LSTM is different in that it will read all input time steps before formulating an output, and in your case, will output an activation for each input time step.

There's no "best" way, test a suite of models for your problem and use whatever works best.

Nick April 18, 2019 at 2:53 pm #

Thank you for answering my questions.

TS May 6, 2019 at 3:27 pm #

Dear Sir,

One point I would like to mention is the Unconditioned Model that Srivastava et al use. a) They do not supply any inputs in the decoder model.. Is this tutorial only using the conditioned model?

b) Even if we are using the any of the 2 models that is mentioned in the paper, we should be passing the hidden state or maybe even the cell state of the encoder model to the models first time step and not to all the time steps..

The tutorial over here shows us that the repeat vector is supplying inputs to all the time steps in the decoder model which should not be the case in any of the models

Also the target time steps in the auto reconstruction decoder model should have been reversed.

Please correct me if I am wrong in understanding the paper. Awaiting for you to clarify my doubt. Thanking you in advance.

Jason Brownlee May 7, 2019

Perhaps.

You can consider the implementation i

TS May 9, 2019 at 1:16 am #

Thank you for the clarification.. Thank you for the post, it helped

Jason Brownlee May 9, 2019 at 6:46 am #

You're welcome.

Taraka Rama April 18, 2019 at 5:47 pm #

REPLY

Hi Jason,

The blog is very interesting. A paper that I published sometime ago uses LSTM autoencoders for German and Dutch dialect analysis.

Best,
Taraka

Jason Brownlee April 19, 2019 at 6:04 am #

REPLY

Thanks.

Taraka Rama April 18, 2019 at 5:48 pm #

REPLY

Hi Jason,

(Forgot to paste the paper link)

The blog is very interesting. A paper that I published sometime ago uses LSTM autoencoders for German and Dutch dialect analysis.

https://www.aclweb.org/anthology/W16-4803

Best,
Taraka

Jason Brownlee April 19, 2019 at 6:05 am #

Thanks for sharing.

Geralt Xu May 4, 2019 at 8:01 pm #

REPLY

Hi Jason,

Thanks for the tutorial, it really helps.

Here is a question about connection between Encoder and Decoder.

In your implementation, you copy the H-dimension hidden vector from Encoder for T times, and convey it as a T*H time series, into the Decoder.

Why chose this way? I'm wondering, there are some another ways to do:

Take hidden vector as the initial state at the first time-step of Decoder, with zero inputs series.

Can this way work?

Best,
Geralt

Jason Brownlee May 5, 2019 at 6:26 am #

REPLY

Because it is an easy way to achieve the desired effect from the paper using the Keras library.

No, I don't think you're approach is the spirit of the paper. Try it and see what happens!?

Atefeh May 6, 2019 at 11:29 am #

REPLY

Hello Mr.Jason
i want to start a handwritten isolated charactor recognition with RNN and lstm.
i mean, we have a number of charactor images and i want a code to recognize that charactor.
would you please help me to find a basic python code for this purpose, ans so i could start the work?

thank you

Jason Brownlee May 6, 2019 at 2:33 pm #

Sounds like a great problem.

Perhaps a CNN-LSTM model would be a good fit!

Xinyang May 13, 2019 at 12:37 pm #

Hi, Dr Brownlee

Thanks for your post, here I want to use LSTM to prediction a time series. For example the series like (1 2 3 4 5 6 7 8 9), and use this series for training. Then the output series is the series of multi-step prediction until it reach the ideal value, like this(9.9 10.8 11.9 12 13.1)

REPLY

Jason Brownlee May 13, 2019 at 2:32 pm #

See this post:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

REPLY

Xinyang May 13, 2019 at 12:57 pm #

Sorry, maybe I didn't make it clear. Here I want to use LSTM to prediction a time series. the sequence may like this[10,20,30,40,50,60,70],and use it for training,if time_step is 3. When input[40,50,60],we want the output is 70. when finish training the model, the prediction begin. when input [50,60,70], the output maybe 79 and then use it for next step prediction, the input is [60,70,79] and output might be 89. Until satisfying certain condition(like the output>=100) the the iteration is over.
So how could I realize the prediction process above and where can I find the code
Please, hope to get your reply

REPLY

Jason Brownlee May 13, 2019 at 2:32 pm #

Yes, you can get started with time series forecasting with LSTMs in this post:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

I have more advanced posts here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

REPLY

Xinyang May 13, 2019 at 5:55 pm #

Thanks for your quick reply.

And I still have a question, the multi-step LSTM
forecast the next two time steps. But in my ca
Lithium-ion battery, and for example let the da
number<160) as the training data, then I want
the certain value(maybe <=0.7Ah) –failure thres
of 250 or so. And between the cycling number
So I have no idea how to define time-steps an
60(220-160=60),the how should I define the tir

I am extremely hope to get your reply, Thank y

Jason Brownlee May 14, 2019 at 7:41 am #

REPLY

You can define the model with any number of inputs or outputs that you require.

If you are having trouble working with numpy arrays, perhaps this will help:
https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/

Kishore Surendra May 15, 2019 at 8:34 pm #

REPLY

Dear Prof,

I have a list as follows :

[5206, 1878, 1224, 2, 329, 89, 106, 901, 902, 149, 8]

When I'm passing it as an input to the reconstruction LSTM (with an added LSTM and repeat vector layer and 1000 epochs) , I get the following predicted output :

[5066.752 1615.2777 1015.1887 714.63916 292.17035 250.14038
331.69427 356.30664 373.15497 365.38977 335.48383]

While some values are almost accurate, most of the others have large deviations from original.

What can be the reason for this, and how do you suggest I fix this ?

Jason Brownlee May 16, 2019 at 6:30 am #

REPLY

No model is perfect.

You can expect error in any model, and variance in neural nets over different training runs, more here:
https://machinelearningmastery.com/faq/single-faq/why-do-i-get-different-results-each-time-i-run-the-code

Kishore Surendra May 21, 2019 at 11:5

Thanks, professor.

If I have varying numbers such as 2 and 1000
dividing each element by the highest element
to the autoencoder ?

Jason Brownlee May 22, 2019 at 8:10 am #

REPLY

Yes, normalizing input is a good idea in general:
https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

Harsh May 15, 2019 at 11:17 pm #

REPLY

Hi,

I have two questions, would be grateful if you can help –

1) The above sequence is very small.
How about if the length of vector is around 800. I tried, but its taking too long.
What do you suggest.

2) Also, is it possible to encode a matrix into vector ?

thanks

Jason Brownlee May 16, 2019 at 6:32 am #

REPLY

Perhaps reduce the size of the sequence?
Perhaps try running on a faster computer?
Perhaps try an alternate approach?

Harsh May 16, 2019 at 6:30 pm #

REPLY

thanks for you quick response… I have a confusion, right now when you mention 'training', it is only one vector… how can truly train it with batches of multiple vectors.

Jason Brownlee May 17, 2019 at 5:51 am #

If you are new to Keras, perhaps start wit
https://machinelearningmastery.com/5-step-life-cy

snowbear May 26, 2019 at 5:51 pm #

Hello Jason, I really appreciate your informati

Question 1. Does `model.add(LSTM(100, activation='relu', input_shape=(n_in,1)))` mean that you are creating an LSTM layer with 100 hidden state?

LSTM structure needs hidden state(h_t) and cell state(c_t) in addition to the input_t, right? So the number 100 there means that with the data whose shape is (9,1) (timestep = 9, input_feature_number = 1), the LSTM layer produces 100-unit long hidden state (h_t)?

Question 2. how small did it get reduced in terms of 'dimension reduction?' Can you tell me how smaller the (9, 1) data got to be reduced in the latent vector?

Jason Brownlee May 27, 2019 at 6:46 am #

REPLY

100 refers to 100 units/nodes in the first hidden layer, perhaps this will help:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

You can experiment with different sized bottlenecks to see what works well/best for your specific dataset.

hassam May 30, 2019 at 3:07 am #

REPLY

hi jason! can this approach is used for sentence correction? i.e spelling or grammatical mistakes of the input text.
for example I have a huge corpus of unlabelled text, and I trained it using autoencoder technique. I want to built a model that takes input (a variable length) sentence, and output the most probable or corrected sentence based on the training data distribution, is it possible?

Jason Brownlee May 30, 2019 at 9:05 am #

REPLY

Perhaps, I'd encourage you to review the literature first.

John June 11, 2019 at 9:35 am #

How do I shape the data for autoencoder if I h

Jason Brownlee June 11, 2019 at 2:22 pm #

Perhaps this will help:
https://machinelearningmastery.com/faq/single-fa

timesteps-and-features-for-lstm-input

Jose Luis July 14, 2019 at 7:48 am #

REPLY

Hi Jason, thanks for your greats articles! I have a work where I get several hundreds of galaxy spectra (a graphic where I have a continuous number of frecuencies in the x axis and the number of received photons from each galaxy in the y axis; it's something like a continuos histogram). I need to make an unsupervised clustering with all this spectra. Do you thing this LSTM autoencoder can be a good option I can use? (Each spectrum has 4000 pairs frecuency-flux).

I was thinking about passing the feature space of the autoencoder with a K-means algorithm or something similar to make the clusters (or better, something like this: https://arxiv.org/abs/1511.06335).

Jason Brownlee July 14, 2019 at 8:18 am #

REPLY

Perhaps try it and evaluate the result?

Xing Wang Tong July 17, 2019 at 10:30 pm #

REPLY

hello and thanks for your tutorial… do you have a similar tutorial with LSTM but with multiple features?

The reason I ask for multiple feature is because I built multiple autoencoder models with different structures but all had timesteps = 30… during training the loss, the rmse, the val_loss and the val_rmse seem all to be within acceptable range ~ 0.05, but when I do prediction and plot the prediction with the original data in one graph, it seems that they both are totally different.

I used MinMaxScaler so I tried to plot the original data and the predictions before I inverse the transform and after, but still the original data and the prediction aren't even close. So, I think I am having trouble plotting the prediction correctly

Jason Brownlee July 18, 2019 at 8:27 am #

You could adapt the examples in this pos
https://machinelearningmastery.com/how-to-deve

sara July 18, 2019 at 1:15 am #

I would like to thank you for the great post, though I wish you have included more sophisticated model.

For example the same thing with 2 feature rather one feature.

REPLY

Jason Brownlee July 18, 2019 at 8:30 am #

Thanks for the suggestion.

The examples here will be helpful:
https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/

REPLY

Shiva July 25, 2019 at 4:30 am #

Hi Jason
Thanks for the tutorial.
I have a sequence A B C. Each A B and C are vectors with length 25.
my samples are like this: A B C label, A' B' C' label',….
How should I reshape the data?
what is the size of the input dimension?

REPLY

Jason Brownlee July 25, 2019 at 7:58 am #

Sorry, I don't follow.

What is the problem that you are having exactly?

REPLY

Shiva July 25, 2019 at 7:39 pm #

my dataset is an array with the shape (10,3,25).(3 features and each feature has 25 features in a vector form)
is it necessary to reshape it?
and what is the value of input_shape for this a

Jason Brownlee July 26, 2019 at 8:

Perhaps read this first to confirm
https://machinelearningmastery.com/faq/s
timesteps-and-features-for-lstm-input

Shiva July 27, 2019 at 9:24 pm #

REPLY

Thank you, Jason.

Jason Brownlee July 28, 2019 at 6:43 am #

REPLY

You're welcome.

Felix August 12, 2019 at 4:11 am #

REPLY

Hi Jason,

Thank you for the great work.

I have one doubt about the layer concept. Is the LSTM layer (100) means, a hidden layer of 100 neurons from the first LSTM layer output and the data from all these 100 layer will consider as the final state value. Is that correct?

Jason Brownlee August 12, 2019 at 6:39 am #

REPLY

Yes.

Felix August 13, 2019 at 5:56 am #

REPLY

Hello Jason,

Thank you for the quick response and appreciate your kind to respond my doubt. Still I am confused with the diagram provided by Keras.

https://github.com/MohammadFneish7/Keras_LSTM_Diagram

Here they have explained as the output of eac
timesteps"

My doubt is like is the output size is "Y – predi

Thanks

Jason Brownlee August 13, 2019 a

Perhaps ask the authors of the di

I have some general advice here that might help:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

---

Felix August 13, 2019 at 6:55 am #

Thank you Jason for the reply.

I have gone through your post and I am clear about the input format to the initial LSTM layer.

I have the below doubt about the internal structure of Keras.

Suppose I have a code as below.

```
step_size = 3
model = Sequential()
model.add(LSTM(32, input_shape=(2, step_size), return_sequences = True))
model.add(LSTM(18))
model.add(Dense(1))
model.add(Activation('linear'))
```

I am getting below summary.

```
_________________________________________________________________
Layer (type) Output Shape Param #
=================================================================
lstm_1 (LSTM) (None, 2, 32) 4608
_________________________________________________________________
lstm_2 (LSTM) (None, 18) 3672
_________________________________________________________________
dense_1 (Dense) (None, 1) 19
_________________________________________________________________
activation_1 (Activation) (None, 1) 0
=================================================================
Total params: 8,299
Trainable params: 8,299
Non-trainable params: 0
_________________________________________________________________
None
```

And I have the below internal layer ma

Layer 1

(3, 128)

(32, 128)

(128,)

Layer 2

(32, 72)

(18, 72)

(72,)

Layer 3

(18, 1)

(1,)

I can not find any relation between output size and the matrix size in each layer. But in each layer the parameter size specified is the total of weight matrix size. Can you please help me to get an idea of the implementation of these numbers.

Jason Brownlee August 13, 2019 at 2:36 pm #

I believe this will help you understand the input shape to an LSTM model: https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

Felix August 13, 2019 at 5:07 pm #

REPLY

Hi Jason,

Thanks for the reply.

I have gone through the post https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

I can able to understand the structure of the input data into the first LSTM layer. But I am not able to identify the matrix structure in the first Layer and the connection with Second Layer. Can you please give me more guidelines to understand the matrix dimensions in the Layers.

Thanks

Jason Brownlee August 14, 2019 at 6:35 am

If the LSTM has return_states=False then
node, e.g. LSTM(50) returns a vector with 50 elem

If the LSTM has return_states=True, such as when
sequence for each node where the length of the se

LSTM(50, input_shape=(100,1)) then the output will be (100,50) or 100 time steps for 50 nodes.

Does that help?

REPLY

Felix August 14, 2019 at 6:52 am #

Thank you Jason for the reply.

Really appreciate the time and effort to give me the answer. It helped me a lot. Thank you very much. You are teaching the whole world. Great !!!

REPLY

Jason Brownlee August 14, 2019 at 2:07 pm #

Thanks, I'm glad it helped.

REPLY

Hossein September 4, 2019 at 6:15 pm #

hi, I am a student and I want to forecast a time-series (electrical load) for the next 24 hr.
I want to do it by using an autoencoder boosting with LSTM.
I am looking for a suitable topology and structure for it.Is it possible to help me?
best regards

REPLY

Jason Brownlee September 5, 2019 at 6:50 am #

Perhaps some of the tutorials here will help as a first step:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

REPLY

Shreeram Bhattarai September 18, 2019 at 11:19

Hi,
I have a question regarding compositive model. In you
And decorder1 tries to reconstruct whatever it has bee
to predict the next sequence.

My question is that once encoder has seen all the data
has already seen all day, definitely it can predict well e

I don't know how encoder part works? Does it works
single encoded latent space from which both part doe

Could you please help me to figure it out. Thank you.

Jason Brownlee September 19, 2019 at 6:01 am #

REPLY

Perhaps focus on the samples aspect of the data, the model receives a sample, and predicts the output, then the next sample is processed, and predicts an output, so on.

It just so happens when we train the model we provide all the samples in a dataset together.

Does that help?

Shreeram Bhattarai September 25, 2019 at 9:52 pm #

REPLY

Thanks for your reply but still not clear to me.

For examples:
we have a 10 time steps data of size 120 (N,10,120). (N is sample numbers)
f5 = frist 5 time steps
l5 = last 5 time steps

while training :
1 Option()
seq_in = (N,f5, 120)
seq_out = (N,l5,120)

model.fit(seq_in, [seq_in,seq_out], epochs=300, verbose=0)

2 Option()

seq_in = (N,10, 120)
seq_out = (N,l5,120)

model.fit(seq_in, [seq_in,seq_out], epochs=300, verbose=0)

Could you please help me to understand that difference between above options? Which way is the correct way to train a network? Thank you.

Jason Brownlee September 26, 20

I don't follow, sorry.

len(f5) == 5?

Then you're asking the difference between

The difference is the number of time steps

If you are new to array shapes, this will help:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

---

Shreeram Bhattarai September 26, 2019 at 6:28 pm #

Sorry for inconvenience .

I am trying to ask with you that whether we have to pass all time steps(in this case 10), or pass first 5 time steps (in this case) to predict the next 5 steps. (I have a data of 10 time steps, my wish is to train a network with two decoder. First decorder should return the reconstruction of input, and second decorder predict the next value).

The question is if I pass all 10 time steps to the network then it will see all the time steps which means it encodes all seen data. from encoding space two decorde will try to reconstruct and predict. It seems that both decoder looks similar then what is the significance of using reconstruction branch decoder? How it helps to prediction decorder in composite model?

Thank you once again.

Jason Brownlee September 27, 2019 at 7:51 am #

Yes, the goal is not to train a predictive model, it is to train an effective encoding of the input.

Using a prediction model as a decoder does not guarantee a better encoding, it is just an alternate strategy to try that may be useful on some problems.

If you want to use an LSTM for time series prediction, you can start here:
https://machinelearningmastery.com/start-here/#deep_learning_time_series

---

Shreeram Bhattarai Septem

Thank you very much your an

Marvi Waheed September 23, 2019 at 8:07 am #

Hi Jason,

I get NaN values when i apply the reconstruction autoe
What can be the reason for it and how to resolve?

I am exploring how reshaping data works for LSTMs and have tried dividing my data into batches of 5 with 200 timesteps each but wanted to check how (1,1000,1) works

REPLY

Jason Brownlee September 23, 2019 at 10:04 am #

Sorry to hear that, this might help with reshaping data:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

REPLY

Marvi Waheed September 23, 2019 at 5:15 pm #

Thanks for replying.

can u identify the lstm model used for reconstruction? is it 1to1 or manyto1?

where can i find explicit examples for lstm models on the website?

REPLY

Jason Brownlee September 24, 2019 at 7:41 am #

You can get started with LSTMs here:
https://machinelearningmastery.com/start-here/#lstm

Including tutorials, a mini-course, and my book.

REPLY

Sounak Ray September 29, 2019 at 2:26 am #

Hello,

I had a question. If I am using the Masking layer in the first part of the network, then does the RepeatVector() layer support masking. Because if it does not support masking and replicates each timestep with the same value, then our output loss will not be c r each example we do not want to include the timestep Could you please share how to ignore the zero padde

Jason Brownlee September 29, 2019 at 6:14

Masking is only needed for input.

The bottle beck will have a internal representation

Masked values are skipped from input.

Sounak Ray September 29, 2019 at 8:34 pm #

REPLY

Hello,

But if the reconstructed timesteps corresponding to the padded part is not zero, then the mean square error loss will be vary large I suppose? Can you tell me if I am wrong here because my mse loss is becoming “nan” after certain number of epochs. And is it best to do post padding or pre padding?

Thanks,
Sounak Ray.

Jason Brownlee September 30, 2019 at 6:08 am #

REPLY

Correct.

The goal is not to create a great predictive model, it is to learn a great intermediate representation.

Sorry to hear that you are getting NANs, I have some suggestions here that might help:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

James December 8, 2019 at 3:55 am #

Hi Jason, thanks for the article. I’m struggling the same problem with Sounak that the mask actually get lost when LSTM return_sequence = False (also the RepeatVector does not explicitly support masking because it actually change the Timestep dimension), since the mask cannot be passed to the end of the model, the loss will be calculated also for those padded timesteps (I’ve validated this on a simple example), which are not preferred.

Jason Brownlee December 8,

I wonder if you can do experi
representation that is learned?

Alireza Hadj October 10, 2019 at 4:58 am #

Hi Jason,
I really enjoy your posts. Thanks for sharing your expertise. Really appreciate it!

I also have a question regarding this post. In the “Prediction Autoencoder” shouldn’t you split the time sequence in half and try to predict the second half by feeding the first half to the encoder. They way that you have implemented the decoder does not truly predict the sequence because the entire sequence had been summarized and given to it by the encoder. Is that true, or am I missing something here?

Jason Brownlee October 10, 2019 at 7:05 am #

REPLY

You can try that – there are many ways to frame a sequence prediction problem, but that is not the model used in this example.

Recall, we are not developing a prediction model, instead an autoencoder.

Marvi Waheed October 21, 2019 at 5:22 pm #

REPLY

Hello,

I’m working on data reconstruction where input is [0:8] columns of the dataset and required output is the 9th column. However the LSTM autoencoder model returns the same value as output after 10 to 15 timesteps. I have applied the model on different datasets but facing similar issue.

What parameter adjustments must I do to obtain unique reconstructed values?

Jason Brownlee October 22, 2019 at 5:44 am #

REPLY

Perhaps try using a different model architecture or different training hyperparameters?

Xi Zhu October 26, 2019 at 5:38 am #

REPLY

Fantastic! I hope you are getting paid for your

Jason Brownlee October 26, 2019 at 5:47 am

Thanks.

Yes, some readers purchase ebooks to support m
https://machinelearningmastery.com/products/

wysohn October 29, 2019 at 4:47 pm #

REPLY

Hello,

Thank you for the amazing article!

I've read comments regarding the RepeatVector(), yet I'm still skeptical if I understood it correctly.

We are merely copying the last output of the encoder LSTM and feed it to each cell of the decoder LSTM to produce the unconditional sequence. Is it correct?

Also, I'm curious that what happens to the internal state of the encoder LSTM. Is it just discarded and will never be used for the decoder? I wonder if using the internal state of the final LSTM cell of the encoder for the initial internal state of LSTM of the decoder would have any kind of benefit. Or is it just completely unnecessary since all we want is to train the encoder?

Thank you for your time!

Jason Brownlee October 30, 2019 at 5:57 am #

REPLY

Correct.

The internal state from the encoder is discarded. The decoder uses state to create the output.

The construction of each output step is conditional on the bottleneck vector and the state from creating the prior output step.

Syed November 12, 2019 at 7:55 am #

REPLY

Really appreciate your hard work and the tutorials are great. I have learned a lot. Can you please write a tutorial on teacher forcing method in encoder decoder architecture? That would be really helpful.

Jason Brownlee November 12, 2019 at 2:01 pm #

REPLY

Thanks!

Yes, I believe all of my tutorials for the encoder-de

Chrysostome November 17, 2019 at 1:27 am #

I would know what is the point to doing an au
it seem equivalent to build one side an encoder decod

two ouput don't seem being used by each other.
Maybe it would be meaningler to use the decodeur as discriminant for the prediction like a GAN

REPLY

Jason Brownlee November 17, 2019 at 7:15 am #

It can be used as a feature extraction model for sequence data.

E.g. you could fit a decoder or any model and make predictions.

REPLY

Meenal December 9, 2019 at 5:09 am #

Is there any way of building an overfitted autoencoder(is overfitting needs to be taken care while training an autoencoder).
and how can one justify that the encoded features obtained are the best compression possible for reconstruction of original.

Also, can you please explain the time distributed layer in terms of the input to this layer. What is the use of time distributed layer. is this layer only useful if working with LSTM layer?

Thanks for all your posts and books, they are very useful in understanding concepts and applying them.

REPLY

Jason Brownlee December 9, 2019 at 6:55 am #

Good question!

Yes. E.g. an autoencoder that does well on a training set but cannot reconstruct test data well .

More on the time distributed layer here:
https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/

Gideon Prior December 12, 2019 at 8:04 am #

I am having trouble seeing the bottle neck. Is
normally, without a trivial data set for your example, be

Jason Brownlee December 12, 2019 at 1:41 p

Yes, the output of the first hidden layer – t

Jenna December 16, 2019 at 12:53 am #

REPLY

Hi Jason,
Thank you so much for writing this great post. But I have a question that really confusing me. Here it is. As the Encoder-Decoder LSTM can benefit the training for output variable length, I'm wondering if it can support the variable multi-step output. I am trying to vary the length of output steps with the "Multiple Parallel Input and Multi-Step Output" example from another post https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/, so the output sequence like:
[[[40 45 85]
[0 0 0]
[0 0 0]]

[[50 55 105]
[60 65 125]
[70 75 145]]

[[60 65 125]
[70 75 145]
[0 0 0]]

[[70 75 145]
[80 85 165]
[0 0 0]]

[[80 85 165]
[0 0 0]
[0 0 0]]]
But my prediction results turned out to be not good. Could you give me some guidance? Is the padding value 0 not suitable? Is the Encoder-Decoder LSTM cannot support the variable length of steps?

Thanks again.

Jason Brownlee December 16, 2019 at 6:18 am #

REPLY

Yes, but you must pad the values. If you cannot use padding with 0, perhaps try -1.

Alternately, you can use a dynamic LSTM and proc
https://machinelearningmastery.com/develop-enc
keras/

Jenna December 16, 2019 at 8:49 pm #

Thank you for suggesting me to proce
no need to make the output time steps variabl
get it right? Besides, I think there is no rationa

models from these two posts except for predicting different timesteps and using different Keras function. Is this understanding correct?
Hope to hear from you. Thanks again.

Jason Brownlee December 17, 2019 at 6:34 am #

REPLY

Perhaps.

jackson January 10, 2020 at 7:21 pm #

REPLY

I have tried your model with my input. The loss was getting convergenced before 10 epochs as excepting. However, the loss became bigger after a point in 10th epochs.
5536/42706 [==>……] – ETA: 39s – loss: 0.4187
5600/42706 [==>……] – ETA: 39s – loss: 0.4190
5664/42706 [==>……] – ETA: 39s – loss: 0.4189
5728/42706 [===>……] – ETA: 39s – loss: 0.4188
5792/42706 [===>……] – ETA: 39s – loss: 0.4189
5856/42706 [===>……] – ETA: 39s – loss: 0.4184
5920/42706 [===>……] – ETA: 38s – loss: 0.4185
5984/42706 [===>……] – ETA: 38s – loss: 0.4188
6048/42706 [===>……] – ETA: 38s – loss: 7.7892
6112/42706 [===>……] – ETA: 38s – loss: 8.6366
6176/42706 [===>……] – ETA: 38s – loss: 8.5517
6240/42706 [===>……] – ETA: 38s – loss: 8.4680
6304/42706 [===>……] – ETA: 38s – loss: 8.3862
6368/42706 [===>……] – ETA: 38s – loss: 8.3056
6432/42706 [===>……] – ETA: 38s – loss: 8.2270
6496/42706 [===>……] – ETA: 38s – loss: 8.1499
6560/42706 [===>……] – ETA: 38s – loss: 8.0738
6624/42706 [===>……] – ETA: 38s – loss: 7.9993
6688/42706 [===>……] – ETA: 38s – loss: 7.9269
6752/42706 [===>……] – ETA: 38s – loss: 7.8556
6816/42706 [===>……] – ETA: 38s – los
6880/42706 [===>……] – ETA: 37s – los
6944/42706 [===>……] – ETA: 37s – los
7008/42706 [===>……] – ETA: 37s – los
7072/42706 [===>……] – ETA: 37s – los
7136/42706 [====>……] – ETA: 37s – lo
7200/42706 [====>……] – ETA: 37s – lo
7264/42706 [====>……] – ETA: 37s – lo
7328/42706 [====>……] – ETA: 37s – lo
7392/42706 [====>……] – ETA: 37s – lo
7456/42706 [====>……] – ETA: 37s – lo

```
7520/42706 [====>.....] – ETA: 37s – loss: 7.0928
7584/42706 [====>.....] – ETA: 37s – loss: 7.0363
7648/42706 [====>.....] – ETA: 37s – loss: 6.9807
7712/42706 [====>.....] – ETA: 37s – loss: 6.9260
7776/42706 [====>.....] – ETA: 37s – loss: 6.8724
7840/42706 [====>.....] – ETA: 37s – loss: 6.8196
7904/42706 [====>.....] – ETA: 36s – loss: 6.7676
7968/42706 [====>.....] – ETA: 36s – loss: 6.7163
8032/42706 [====>.....] – ETA: 36s – loss: 6.6655
8096/42706 [====>.....] – ETA: 36s – loss: 6.6160
8160/42706 [====>.....] – ETA: 36s – loss: 6.5667
8224/42706 [====>.....] – ETA: 36s – loss: 6.5184
8288/42706 [====>.....] – ETA: 36s – loss: 6.4707
8352/42706 [====>.....] – ETA: 36s – loss: 6.4239
8416/42706 [====>.....] – ETA: 36s – loss: 6.3782
8480/42706 [====>.....] – ETA: 36s – loss: 2378.7514
8544/42706 [=====>.....] – ETA: 36s – loss: 27760.9716
8608/42706 [=====>.....] – ETA: 36s – loss: 27755.8645
8672/42706 [=====>.....] – ETA: 36s – loss: 27978.9607
8736/42706 [=====>.....] – ETA: 36s – loss: 28032.9492
8800/42706 [=====>.....] – ETA: 35s – loss: 28025.2542
8864/42706 [=====>.....] – ETA: 35s – loss: 27902.1603
8928/42706 [=====>.....] – ETA: 35s – loss: 27837.8133
8992/42706 [=====>.....] – ETA: 35s – loss: 27830.6104
9056/42706 [=====>.....] – ETA: 35s – loss: 27731.7000
9120/42706 [=====>.....] – ETA: 35s – loss: 27630.7813
9184/42706 [=====>.....] – ETA: 35s – loss: 27768.5311
9248/42706 [=====>.....] – ETA: 35s – loss: 28076.0159
```

Jason Brownlee January 11, 2020 at 7:23 am #

REPLY

Nice work.

Perhaps try fitting the model again to see if you get a different result?

sampath January 18, 2020 at 8:46 am #

Hi Jason,

I am trying to implement a LSTM autoencoder using e
functional API of keras and also NOT have my decode
LSTM will not have any input but just the hidden and c
encoder output to preserve all the information necessa
inputs to the decoder). Is something like this possible

Jason Brownlee January 18, 2020 at 8:56 am #

REPLY

Yes, I believe that is the normal architecture described in the above tutorial.

If not, perhaps I don't understand what you're trying to achieve.

sampath January 18, 2020 at 8:59 am #

REPLY

I mean in a functional API(the above mentioned is a sequential api).
this is the code from one of your article:

```
def define_models(n_input, n_output, n_units):
    # define training encoder
    encoder_inputs = Input(shape=(None, n_input))
    encoder = LSTM(n_units, return_state=True)
    encoder_outputs, state_h, state_c = encoder(encoder_inputs)
    encoder_states = [state_h, state_c]
    # define training decoder
    decoder_inputs = Input(shape=(None, n_output))
    decoder_lstm = LSTM(n_units, return_sequences=True, return_state=True)
    decoder_outputs, _, _ = decoder_lstm(decoder_inputs, initial_state=encoder_sta
    decoder_dense = Dense(n_output, activation='softmax')
    decoder_outputs = decoder_dense(decoder_outputs)
    model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
    # define inference encoder
    encoder_model = Model(encoder_inputs, encoder_states)
    # define inference decoder
    decoder_state_input_h = Input(shape=(n_units,))
    decoder_state_input_c = Input(shape=(n_units,))
    decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]
    decoder_outputs, state_h, state_c = decoder_lstm(decoder_inputs, initial_state
    decoder_states = [state_h, state_c]
    decoder_outputs = decoder_dense(decoder_outputs)
    decoder_model = Model([decoder_inputs] + decoder_states_inputs, [decoder_outpu
    # return all models
    return model, encoder_model, decoder_model
```

what if i want my 'decoder_lstm' to not have any inputs(in this code, it is give 'decoder_inputs' as inputs)

Jason Brownlee January 18, 2020 a

Ah I see. Thanks.

Some experimentation will be required, I d

sampath January 18, 2020 at 9:0

the reason I want to use functional API is because I want to use stacked LSTM(multiple layers) and I want the hidden_state from all layers at the last time step of encoder. This is possible only with functional API right?

Jason Brownlee January 19, 2020 at 7:03 am #

Most likely, yes.

Rajnish Pandey February 3, 2020 at 11:53 pm #

REPLY

Hey, @Jason Brownlee, I am working on textual data could you please explain this concept regarding the text? I am calculating errors with glove pre-trained vector but my result is not up to the mark

Thank you in advance

Jason Brownlee February 4, 2020 at 7:55 am #

REPLY

Perhaps start here:
https://machinelearningmastery.com/start-here/#nlp

Nattachai February 23, 2020 at 7:57 pm #

REPLY

Hi Jason,
I am working on time series data.
Can I use RNN Autoencode as time series representation like SAX, PAA

Thank you

Jason Brownlee February 24, 2020 at 7:39 ar

Perhaps try it and see?

David March 2, 2020 at 1:20 pm #

Great article! Thanks!

Jason Brownlee March 3, 2020 at 5:54 am #

REPLY

Thanks, I'm happy it helped.

Leung Lau March 9, 2020 at 3:40 pm #

REPLY

Hi Jason, I have a question. is last 100*1 vector you printed in the end of article the feature of the sequence? Can this vector be later used as, for example, sequence classification or regression? Thanks!

Jason Brownlee March 10, 2020 at 5:38 am #

REPLY

In most of the examples we are reconstructing the input sequence of numeric values. Regression, but not really.

The final example is the feature vector.

Han March 21, 2020 at 5:44 pm #

REPLY

Hello, dr. Jason, thanks for this useful tutorial!
I built a convolutional Autoencoder (CAE), the result of the reconstructed image from the decoder is better than the original image, and i think if a classifer took a better image it would provide a good output..

so I want to classify the input weather it is a bag, shoes .. etc
Is it better to:
1- delete the decoder and make the encoder as a classifier? (if I did this will it be like a normal CNN?)
2- or do the same as "Composite LSTM Autoencoder in this tutorial" to my CAE
3- take the output of the decoder (better image) to a classifier

I do not know, and I am really new to AI world, your reply will be so useful to me.
Thank you.

Jason Brownlee March 22, 2020 at 6:52 am #

You would keep the encoder and use the model.

Han March 22, 2020 at 2:41 pm #

so I take the output of the encoder (maybe 8*8 matrix) and make it as input to model that takes the same size (8*8)? no need to connect both CNNs (encoder, classifier)?

REPLY

Jason Brownlee March 23, 2020 at 6:11 am #

You can connect them if you want or use the encoder as a feature extractor.

Typically extracted features are a 1d vector, e.g. a bottleneck layer.

REPLY

Rekha March 28, 2020 at 1:05 am #

Is it possible to use autoencoders for lstm time series prediction

REPLY

Jason Brownlee March 28, 2020 at 6:21 am #

Sure. They could extract features, then feed these features into another model to make predictions.

REPLY

Rekha March 28, 2020 at 2:17 pm #

Will there be a blog on autoencoders for lstm time series prediction in machinelearningmastery.com

REPLY

Jason Brownlee March 29, 2020 at 5:49 am #

The above tutorial is exactly this.

Augustus Van Dusen March 29, 2020 at 5:13 am

Jason, I ran the Prediction LSTM Autoencode

2020-03-28 14:01:53.115186: E tensorflow/core/grapp
Iteration = 0, topological sort failed with message: The
2020-03-28 14:01:53.120793: E tensorflow/core/grapp
Iteration = 1, topological sort failed with message: The
2020-03-28 14:01:53.127457: E tensorflow/core/grapp
Invalid argument: The graph couldn't be sorted in topo

2020-03-28 14:01:53.190262: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:502] remapper failed: Invalid argument: The graph couldn't be sorted in topological order.
2020-03-28 14:01:53.194523: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:502] arithmetic_optimizer failed: Invalid argument: The graph couldn't be sorted in topological order.
2020-03-28 14:01:53.198763: E tensorflow/core/grappler/optimizers/dependency_optimizer.cc:697] Iteration = 0, topological sort failed with message: The graph couldn't be sorted in topological order.
2020-03-28 14:01:53.204018: E tensorflow/core/grappler/optimizers/dependency_optimizer.cc:697] Iteration = 1, topological sort failed with message: The graph couldn't be sorted in topological order.

However, the code ran and the answer was equivalent to your answer. Have you seen this error? If so, do you know what it means?

Thanks.

Jason Brownlee March 29, 2020 at 6:05 am #

REPLY

I have not seen these warnings before, sorry.

Perhaps try searching/posting on stackoverflow?

Leave a Reply

Name (required)

Email (will not be published) (requ

Website

SUBMIT COMMENT

Welcome!

My name is *Jason Brownlee* PhD, and I **help developers** get results with **machine learning**.

Read more

Never miss a tutorial:

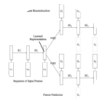
Picked for you:

How to Reshape Input Data for Long Short-Term Memory Networks in Keras

How to Develop an Encoder-Decoder Model for Sequence-to-Sequence Prediction in Keras

How to Develop an Encoder-Decoder Model with Attention in Keras

How to Use the TimeDistributed Layer in Keras



A Gentle Introduction to LSTM Autoencoders

Loving the Tutorials?

The LSTMs with Python EBook
is where I keep the

SEE WHAT