Need help with LSTMs in Python? Take the FREE Mini-Course.

How to One Hot Encode Sequence Data in Python

by **Jason Brownlee** on July 12, 2017 in **Long Short-Term Memory Networks**

Machine learning algorithms cannot work with categorical data directly.

Categorical data must be converted to numbers.

This applies when you are working with a sequence classification type problem and plan on using deep learning methods such as Long Short-Term Memory recurrent neural networks.

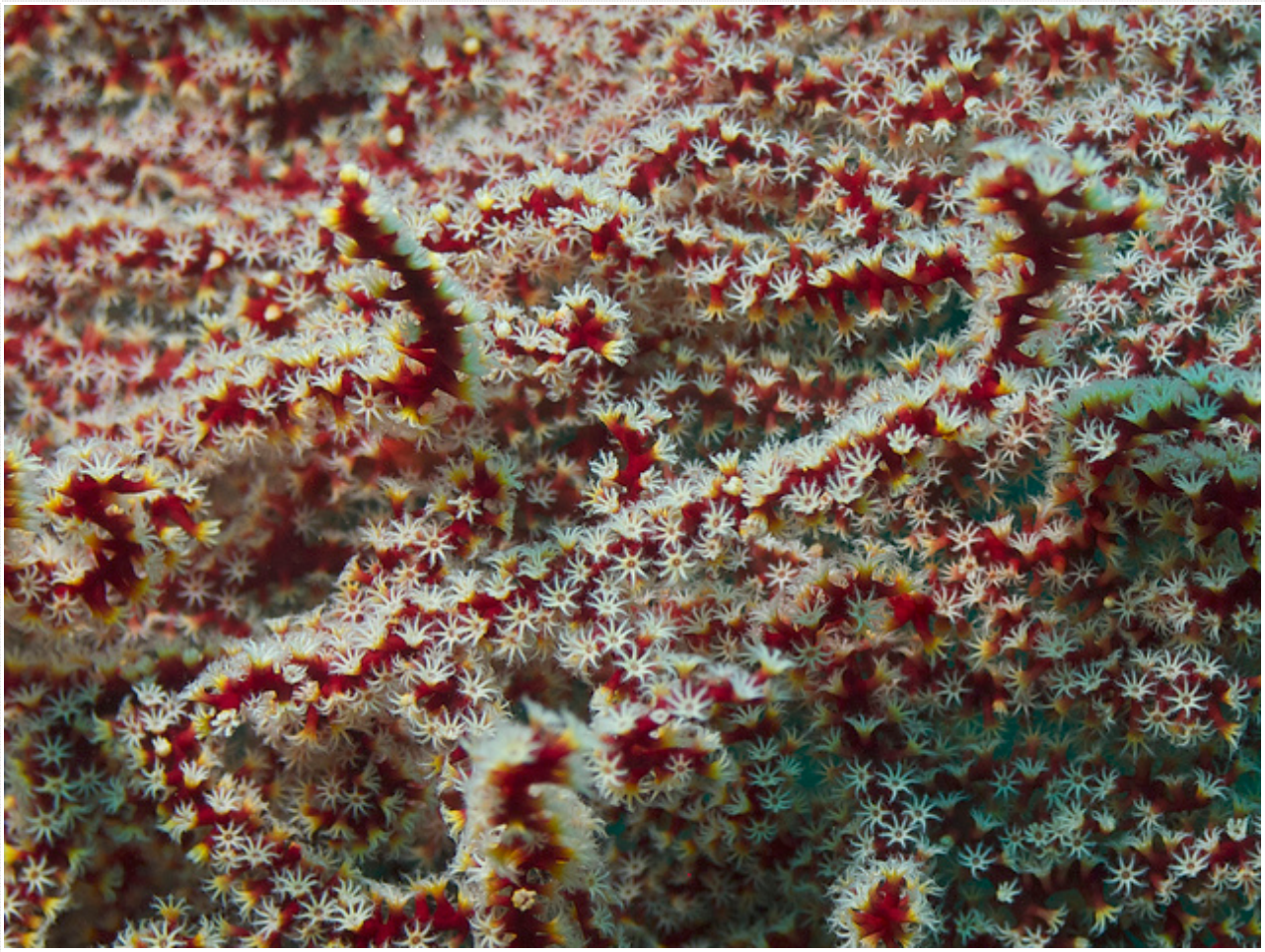
In this tutorial, you will discover how to convert your input or output sequence data to a one hot encoding for use in sequence classification problems with deep learning in Python.

After completing this tutorial, you will know:

- What an integer encoding and one hot encoding are and why they are necessary in machine learning.
- How to calculate an integer encoding and one hot encoding by hand in Python
- How to use the scikit-learn and Keras lib... a in

Python.

Let's get started.



How to One Hot Encode Sequence Classification Data in Python
Photo by Elias Levy, some rights reserved.

Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. What is One Hot Encoding?
2. Manual One Hot Encoding
3. One Hot Encode with scikit-learn
4. One Hot Encode with Keras

What is One Hot Encoding?

A one hot encoding is a representation of categorical variables as binary vectors.

This first requires that the categorical values be mapped to integer values.

Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

Worked Example of a One Hot Encoding

Let's make this concrete with a worked example.

Assume we have a sequence of labels with the values 'red' and 'green'.

We can assign 'red' an integer value of 0 and 'green' the integer value of 1. As long as we always assign these numbers to these labels, this is called an integer encoding. Consistency is important so that we can invert the encoding later and get labels back from integer values, such as in the case of making a prediction.

Next, we can create a binary vector to represent each integer value. The vector will have a length of 2 for the 2 possible integer values.

The 'red' label encoded as a 0 will be represented with a binary vector [1, 0] where the zeroth index is marked with a value of 1. In turn, the 'green' label encoded as a 1 will be represented with a binary vector [0, 1] where the first index is marked with a value of 1.

If we had the sequence:

```
1 'red', 'red', 'green'
```

We could represent it with the integer encoding:

```
1 0, 0, 1
```

And the one hot encoding of:

```
1 [1, 0]
2 [1, 0]
3 [0, 1]
```

Why Use a One Hot Encoding?

A one hot encoding allows the representation of categorical data to be more expressive.

Many machine learning algorithms cannot w

must be converted into numbers. This is required for both input and output variables that are categorical.

We could use an integer encoding directly, rescaled where needed. This may work for problems where there is a natural ordinal relationship between the categories, and in turn the integer values, such as labels for temperature ‘cold’, warm’, and ‘hot’.

There may be problems when there is no ordinal relationship and allowing the representation to lean on any such relationship might be damaging to learning to solve the problem. An example might be the labels ‘dog’ and ‘cat’

In these cases, we would like to give the network more expressive power to learn a probability-like number for each possible label value. This can help in both making the problem easier for the network to model. When a one hot encoding is used for the output variable, it may offer a more nuanced set of predictions than a single label.

Manual One Hot Encoding

In this example, we will assume the case where we have an example string of characters of alphabet letters, but the example sequence does not cover all possible examples.

We will use the input sequence of the following characters:

```
1 hello world
```

We will assume that the universe of all possible inputs is the complete alphabet of lower case characters, and space. We will therefore use this as an excuse to demonstrate how to roll our own one hot encoding.

The complete example is listed below.

```
from numpy import argmax
# define input string
data = 'hello world'
print(data)
# define universe of possible input values
alphabet = 'abcdefghijklmnopqrstuvwxyz '
# define a mapping of chars to integers
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
# integer encode input data
integer_encoded = [char_to_int[char] for char in data]
print(integer_encoded)
# one hot encode
onehot_encoded = list()
for value in integer_encoded:
    letter = [0 for _ in range(len(alphabet))]
    letter[value] = 1
    onehot_encoded.append(letter)
print(onehot_encoded)
# invert encoding
inverted = int_to_char[argmax(onehot_encoded[0])]
print(inverted)
```

Running the example first prints the input string.

A mapping of all possible inputs is created from char values to integer values. This mapping is then used to encode the input string. We can see that the first letter in the input ‘h’ is encoded as 7, or the index 7 in the array of possible input values (alphabet).

The integer encoding is then converted to a one hot encoding. This is done one integer encoded character at a time. A list of 0 values is created the length of the alphabet so that any expected character can be represented.

Next, the index of the specific character is marked with a 1. We can see that the first letter ‘h’ integer encoded as a 7 is represented by a binary vector with the length 27 and the 7th index marked with a 1.

Finally, we invert the encoding of the first letter and print the result. We do this by locating the index of in the binary vector with the largest value using the NumPy argmax() function and then using the integer value in a reverse lookup table of character values to integers.

Note: output was formatted for readability.

```
hello world

[7, 4, 11, 11, 14, 26, 22, 14, 17, 11, 3]

[[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
```

```
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]

h
```

Now that we have seen how to roll our own one hot encoding from scratch, let's see how we can use the scikit-learn library to perform this mapping automatically for cases where the input sequence fully captures the expected range of input values.

One Hot Encode with scikit-learn

In this example, we will assume the case where you have an output sequence of the following 3 labels:

```
"cold"
"warm"
"hot"
```

An example sequence of 10 time steps may be:

```
cold, cold, warm, cold, hot, hot, warm, cold, warm, hot
```

This would first require an integer encoding, such as 1, 2, 3. This would be followed by a one hot encoding of integers to a binary vector with 3 values, such as [1, 0, 0].

The sequence provides at least one example of every possible value in the sequence. Therefore we can use automatic methods to define the mapping of labels to integers and integers to binary vectors.

In this example, we will use the encoders from the scikit-learn library. Specifically, the LabelEncoder of creating an integer encoding of labels and the OneHotEncoder for creating a one hot encoding of integer encoded values.

The complete example is listed below.

```
from numpy import array
from numpy import argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import One
# define example
data = ['cold', 'cold', 'warm', 'col
```

```
values = array(data)
print(values)
# integer encode
label_encoder = LabelEncoder()
integer_encoded = label_encoder.fit_transform(values)
print(integer_encoded)
# binary encode
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
# invert first example
inverted = label_encoder.inverse_transform([argmax(onehot_encoded[0, :])])
print(inverted)
```

Running the example first prints the sequence of labels. This is followed by the integer encoding of the labels and finally the one hot encoding.

The training data contained the set of all possible examples so we could rely on the integer and one hot encoding transforms to create a complete mapping of labels to encodings.

By default, the OneHotEncoder class will return a more efficient sparse encoding. This may not be suitable for some applications, such as use with the Keras deep learning library. In this case, we disabled the sparse return type by setting the *sparse=False* argument.

If we receive a prediction in this 3-value one hot encoding, we can easily invert the transform back to the original label.

First, we can use the argmax() NumPy function to locate the index of the column with the largest value. This can then be fed to the LabelEncoder to calculate an inverse transform back to a text label.

This is demonstrated at the end of the example with the inverse transform of the first one hot encoded example back to the label value ‘cold’.

Again, note that input was formatted for readability.

```
['cold' 'cold' 'warm' 'cold' 'hot' 'hot' 'warm' 'cold' 'warm' 'hot']

[0 0 2 0 1 1 2 0 2 1]

[[ 1.  0.  0.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]
 [ 1.  0.  0.]
 [ 0.  0.  1.]
```

```
 [ 0.  1.  0.]]

['cold']
```

In the next example, we look at how we can directly one hot encode a sequence of integer values.

One Hot Encode with Keras

You may have a sequence that is already integer encoded.

You could work with the integers directly, after some scaling. Alternately, you can one hot encode the integers directly. This is important to consider if the integers do not have a real ordinal relationship and are really just placeholders for labels.

The Keras library offers a function called to_categorical() that you can use to one hot encode integer data.

In this example, we have 4 integer values [0, 1, 2, 3] and we have the input sequence of the following 10 numbers:

```
data = [1, 3, 2, 0, 3, 2, 2, 1, 0, 1]
```

The sequence has an example of all known values so we can use the to_categorical() function directly. Alternately, if the sequence was 0-based (started at 0) and was not representative of all possible values, we could specify the num_classes argument *to_categorical(num_classes=4)*.

A complete example of this function is listed below.

```
from numpy import array
from numpy import argmax
from keras.utils import to_categorical
# define example
data = [1, 3, 2, 0, 3, 2, 2, 1, 0, 1]
data = array(data)
print(data)
# one hot encode
encoded = to_categorical(data)
print(encoded)
# invert encoding
inverted = argmax(encoded[0])
print(inverted)
```

Running the example first defines and prints the input sequence.

The integers are then encoded as binary vectors and printed. We can see that the first integer value 1 is encoded as [0, 1, 0, 0] just like we would expect.

We then invert the encoding by using the NumPy argmax() function on the first value in the sequence that returns the expected value 1 for the first integer.

```
[1 3 2 0 3 2 2 1 0 1]

[[ 0.  1.  0.  0.]
 [ 0.  0.  0.  1.]
 [ 0.  0.  1.  0.]
 [ 1.  0.  0.  0.]
 [ 0.  0.  0.  1.]
 [ 0.  0.  1.  0.]
 [ 0.  0.  1.  0.]
 [ 0.  1.  0.  0.]
 [ 1.  0.  0.  0.]
 [ 0.  1.  0.  0.]]

1
```

Further Reading

This section lists some resources for further reading.

- What is one hot encoding and when is it used in data science? on Quora
- OneHotEncoder scikit-learn API documentation
- LabelEncoder scikit-learn API documentation
- to_categorical Keras API documentation
- Data Preparation for Gradient Boosting with XGBoost in Python
- Multi-Class Classification Tutorial with the Keras Deep Learning Library

Summary

In this tutorial, you discovered how to encode your categorical sequence data for deep learning using a one hot encoding in Python.

Specifically, you learned:

- What integer encoding and one hot encoding are and why they are necessary in machine learning.
- How to calculate an integer encoding and one hot encoding by hand in Python.
- How to use the scikit-learn and Keras libraries to automatically encode your sequence data in Python.

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

About Jason Brownlee

Dr. Jason Brownlee is a husband, proud father, academic researcher, author, professional developer and a machine learning practitioner. He is dedicated to helping developers get started and get good at applied machine learning. Learn more.

View all posts by Jason Brownlee →

‹ How to Remove Trends and Seasonality with a Difference Transform in Python

What is the Difference Between Test and Validation Datasets? ›

21 Responses to *How to One Hot Encode Sequence Data in Python*

Natallia Lundqvist July 12, 2017 at 7:02 pm #

REPLY

Thank you for this post! It came really timely. A question. Utility to_categorical(data) accepts vector as input. What would one do in case you have a 2D tensor as input? Would you loop over number of samples (could be several hundred of thousands entries + do the model training in a loop) or one would have to do seq = tokenizer.texts_to_sequences(inputseq) and then tokenizer.sequences_to_matrix(seq, mode='binary')?? The last option will give a 2D tensor as output in form array([0., 1., 1., …, 0., 0., 0.]). In that case model training goes as usual, but for decoding predictions one would have to loop to find all maximums (argmax give ONLY the first maximum).

Jason Brownlee July 13, 2017 at 9:52 am #

REPLY

Ouch, it is hard to give good advice without specifics. Choose a formulation that preserves the structure of your sequence.

Franco July 14, 2017 at 5:30 am #

REPLY

Awesome post again Jason! Thank you. I've just added it to my ML checklist.

Jason Brownlee July 14, 2017 at 8:34 am #

REPLY

Excellent Franco!

Rohit July 14, 2017 at 10:22 am #

REPLY

how do we perform integer encoding in keras ? Is there any inbuild function like LabelEncoder in scikit ?

Jason Brownlee July 15, 2017 at 9:36 am #

REPLY

Not really. You can do it manually.

If you're working with text, there are tools here:
https://keras.io/preprocessing/text/

Moe July 24, 2017 at 2:20 pm #

REPLY

Hi Jason,

Great article!

What if the sequence was not representative of all possible values, and we don't know the num of classes to set the num_classes argument?

Jason Brownlee July 25, 2017 at 9:26 am #

REPLY

Thanks.

Yes, that is a challenge.

Off the cuff, you may need to re-encode data in the future. Or choose an encoding that has "space" for new values that you have not seen yet. I expect there is some good research on this topic.

Malcolm August 17, 2017 at 9:44 pm #

REPLY

Informative post as always!

I understand how this is used to train a model.
However, how can you ensure that data you want to get predictions for is encoded in the same way as the training data? E.g. 'hot' maps to column 3.

For my application once the model is trained it will need to provide predictions at a later date and on different machines.

REPLY

Jason Brownlee August 18, 2017 at 6:19 am #

Great question!

It must be consistent. Either you use the same code, and/or you save the "model" that performs the transform.

sasi September 11, 2017 at 4:16 pm #

REPLY

Thank u!!!!!! great work…….

Jason Brownlee September 13, 2017 at 12:21 pm #

REPLY

Thanks.

NVK September 29, 2017 at 3:37 am #

REPLY

Hey there,
Great tutorial as always!
Quick question though – what if my dataset contains both categorical and continuous values?
Wouldn't OH encoding encode the entire dataset, when all I really need is just the categorical columns encoded?
Also would the recommended flow be to (a) scale the numeric data only (b) encode the whole dataset?
Or (a) encode the whole dataset and then (b) scale all values?

Jason Brownlee September 29, 2017 at 5:09 am #

REPLY

I would recommend only encoding the categorical variables.

ed October 4, 2017 at 5:03 am #

REPLY

Hi there,

I am working on model productization. To reduce the data processing gap between experiment and production, would like to embed one hot encoding in Keras model. The questions are

1. When using “to_categorical”, it will convert categorical on the fly and it seems break the encoding. For example, we have “apple”, “orange” and “banana” when training model. After rollout the model, there include unseen category such as “lemon”. How can we handle this kind of scenario?

2. As want to embed the encoding in Keras. Therefore, trying to not using sklearn label encoder and one hot encoder. Does it possible to do that?

Please kindly advise.

Jason Brownlee October 4, 2017 at 5:50 am #

REPLY

Perhaps you can develop your own small function to perform the encoding and perform it consistently?

Adhaar Sharma October 5, 2017 at 8:38 am #

REPLY

Hi Jason,

Love the website! Had a quick question:

If I have a X dataset of 6 categorical attributes, where each attribute has lets say the following number of unique categories respectively [4, 4, 4, 3, 3, 3]. Also lets say there are 600 instances

When I LabelEncode and then OneHotEncode this X dataset, I now rightly get a sequence of 21 0-1 values after the toarray() method on the fit_transform.

However, when I input the X dataset like that, i.e. with a shape of (600, 21), I get a much worse error than if I had just left it LabelEncoded and with a shape of (600, 6).

My question is am I doing something wrong? Should I be re-grouping the sequence of 21 integers that I get back into their respective clusters? For ex: I get this array for the first row as a result of the OneHotEncode:
[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0]
so should I group the digits back into something like this
[(0, 0, 0, 1), (0, 0, 0, 1), (0, 0, 0, 1), (0, 0, 1), (0, 0, 1), (0, 1, 0)]
where now we again have a (600, 6) shape for my input dataset?

I tried creating a numpy array with this formulation but the sci-kit decision tree classifier checks and tries to convert any numpy array where

validate.

Essentially, I want to know whether the (600, 21) shape is causing any data loss being in that format. And if it is what is the best way to regroup the encodings into their respective attributes so I can lower my error.

Thank you!

Jason Brownlee October 5, 2017 at 5:21 pm #

REPLY

Hmmm.

The data should be 500, 21 after the encoding, so far so good.

No need to group.

Skill better or worse depends on the algorithm and your specific data. Everything is a test to help us discover what works for our problem.

Consider trying more algorithms.
Consider trying encoding only some variables and leaving others as-is or integer encode.
Brainstorm more things to try, see this post for ideas:
http://machinelearningmastery.com/machine-learning-performance-improvement-cheat-sheet/

I hope that helps as a start.

Adhaar Sharma October 6, 2017 at 5:30 am #

REPLY

Okay, great! I just wasn't sure whether to debug the onehot encoder or try other things. Looks like trying other things is the way to go.

Thanks you so much for your help Jason!

Prashant October 10, 2017 at 4:24 am #

REPLY

Hii,
Great tutorial as always!
I am having a doubt…Why we are reshaping the integer_encoded vector "integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)"?

Jason Brownlee October 10, 2017 at 7:53 am #

REPLY

Great question, because the sklearn tools expect 2D data as input.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery

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

Read More

Deep Learning for Sequence Prediction

Cut through the

Get Your Start in Machine Learning

POPULAR

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

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

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

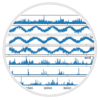
How to Setup a Python Environment for Machine Learning and Deep Learning with Anaconda
MARCH 13, 2017

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

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

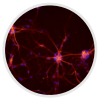
Multi-Class Classification Tutorial with t

JUNE 2, 2016



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

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



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