# 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.

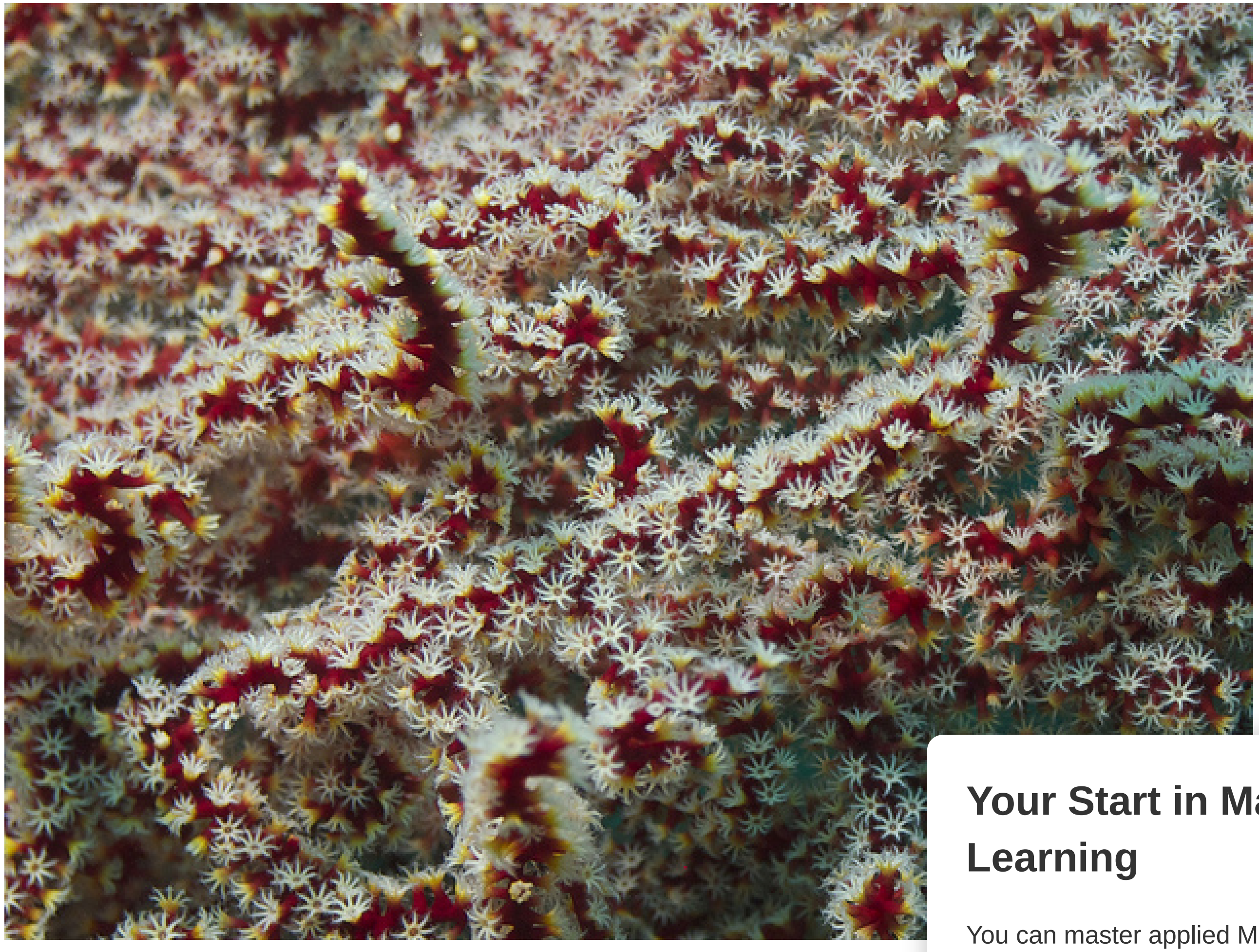
This applies when you are working with a sequence classification type problem and plan on using de
Memory recurrent neural networks.

In this tutorial, you will discover how to convert your input or output sequence data to a one hot enco vith 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 lea
- 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 P

Let’s get started.



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

## Tutorial Overview

This tutorial is divided into 4 parts; they are:

1. What is One Hot Encoding?

# 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

The 'red' label encoded as a 0 will be represented with a binary vector [1, 0] where the zeroth index el encoded as a 1 will be represented with a binary vector [0, 1] where the first index is marked with a v

If we had the sequence:

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

We could represent it with the integer encoding:

```
0, 0, 1
```

And the one hot encoding of:

```
[1, 0]
[1, 0]
[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 work with categorical data directly. The categories 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:

```
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.

```python
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 u he first letter in the input 'h' is encoded as 7, or the index 7 in the array of possible input values (alphabe

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

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, 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, 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, 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, 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, 1, 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, 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, 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]]

h
```

Now that we have seen how to roll our own one hot encoding from scratch, let's see how we can use 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 labe

```
"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 enc

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 argmax
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
# define example
data = ['cold', 'cold', 'warm', 'cold', 'hot', 'hot', 'warm', 'cold', 'warm', 'hot']
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 t

The training data contained the set of all possible examples so we could rely on the integer and one

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

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 he 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

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 direc ed 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 ect.

We then invert the encoding by using the NumPy argmax() function on the first value in the sequenc 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 learnin
- 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 P

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

## Develop LSTMs for Sequence Predict

### Develop Your Own LSTM models in Minute

…with just a few lines of python code