Navigation

Want help with deep learning for text? Take the FREE Mini-Course

Search...

How to Use Word Embedding Layers for Deep Learning with Keras

by **Jason Brownlee** on October 4, 2017 in **Deep Learning for Natural Language Processing**

Tweet Share Share G+

Word embeddings provide a dense representation of words and their relative meanings.

They are an improvement over sparse representations used in simpler bag of word model representations.

Word embeddings can be learned from text data and reused among projects. They can also be learned as part of fitting a neural network on text data.

In this tutorial, you will discover how to use word embeddings for deep learning in Python with Keras.

After completing this tutorial, you will know:

- About word embeddings and that Keras supports word embeddings via the Embedding layer.
- How to learn a word embedding while fitting a neural network.
- How to use a pre-trained word embedding in a neural network.

Let’s get started.

- **Update Feb/2018**: Fixed a bug due to a change in the underlying APIs.

How to Use Word Embedding Layers for Deep Learning with Keras
Photo by thisguy, some rights reserved.

Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. Word Embedding
2. Keras Embedding Layer
3. Example of Learning an Embedding
4. Example of Using Pre-Trained GloVe Embedding

1. Word Embedding

A word embedding is a class of approaches for representing words and documents using a dense vector representation.

It is an improvement over more the traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. These representations were sparse because the vocabularies were vast and a given word or document would be represented by a large vector comprised mostly of zero values.

Instead, in an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space.

The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used.

The position of a word in the learned vector space is referred to as its embedding.

Two popular examples of methods of learning word embeddings from text include:

- Word2Vec.
- GloVe.

In addition to these carefully designed methods, a word embedding can be learned as part of a deep learning model. This can be a slower approach, but tailors the model to a specific training dataset.

2. Keras Embedding Layer

Keras offers an Embedding layer that can be used for neural networks on text data.

It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset.

It is a flexible layer that can be used in a variety of ways, such as:

- It can be used alone to learn a word embedding that can be saved and used

in another model later.
- It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- It can be used to load a pre-trained word embedding model, a type of transfer learning.

The Embedding layer is defined as the first hidden layer of a network. It must specify 3 arguments:

It must specify 3 arguments:

- **input_dim**: This is the size of the vocabulary in the text data. For example, if your data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
- **output_dim**: This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger. Test different values for your problem.
- **input_length**: This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 1000 words, this would be 1000.

For example, below we define an Embedding layer with a vocabulary of 200 (e.g. integer encoded words from 0 to 199, inclusive), a vector space of 32 dimensions in which words will be embedded, and input documents that have 50 words each.

```
e = Embedding(200, 32, input_length=50)
```

The Embedding layer has weights that are learned. If you save your model to file, this will include weights for the Embedding layer.

The output of the *Embedding* layer is a 2D vector with one embedding for each word in the input sequence of words (input document).

If you wish to connect a *Dense* layer directly to an Embedding layer, you must first flatten the 2D output matrix to a 1D vector using the *Flatten* layer.

Now, let’s see how we can use an Embedding layer in practice.

3. Example of Learning an Embedding

In this section, we will look at how we can learn a word embedding while fitting a neural network on a text classification problem.

We will define a small problem where we have 10 text documents, each with a comment about a piece of work a student submitted. Each text document is classified as positive “1” or negative “0”. This is a simple sentiment analysis problem.

First, we will define the documents and their class labels.

```
# define documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
```

Next, we can integer encode each document. This means that as input the Embedding layer will have sequences of integers. We could experiment with other more sophisticated bag of word model encoding like counts or TF-IDF.

Keras provides the one_hot() function that creates a hash of each word as an efficient integer encoding. We will estimate the vocabulary size of 50, which is much larger than needed to reduce the probability of collisions from the hash function.

```
# integer encode the documents
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
print(encoded_docs)
```

The sequences have different lengths and Keras prefers inputs to be vectorized and all inputs to have the same length. We will pad all input sequences to have the length of 4. Again, we can do this with a built in Keras function, in this case the pad_sequences() function.

```
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

We are now ready to define our *Embedding* layer as part of our neural network model.

The *Embedding* has a vocabulary of 50 and an input length of 4. We will choose a small embedding space of 8 dimensions.

The model is a simple binary classification model. Importantly, the output from the *Embedding* layer will be 4 vectors of 8 dimensions each, one for each word. We flatten this to a one 32-element vector to pass on to the *Dense* output layer.

```
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
# summarize the model
print(model.summary())
```

Finally, we can fit and evaluate the classification model.

```
# fit the model
model.fit(padded_docs, labels, epochs=50, verbose=0)
# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

The complete code listing is provided below.

```
from numpy import array
from keras.preprocessing.text import one_hot
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
# define documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
# integer encode the documents
vocab_size = 50
encoded_docs = [one_hot(d, vocab_size) for d in docs]
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
# define the model
model = Sequential()
model.add(Embedding(vocab_size, 8, input_length=max_length))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
# summarize the model
```

```
print(model.summary())
# fit the model
model.fit(padded_docs, labels, epochs=50, verbose=0)
# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

Running the example first prints the integer encoded documents.

```
[[6, 16], [42, 24], [2, 17], [42, 24], [18], [17], [22, 17], [27, 42], [22, 24], [4
```

Then the padded versions of each document are printed, making them all uniform length.

```
[[ 6 16  0  0]
 [42 24  0  0]
 [ 2 17  0  0]
 [42 24  0  0]
 [18  0  0  0]
 [17  0  0  0]
 [22 17  0  0]
 [27 42  0  0]
 [22 24  0  0]
 [49 46 16 34]]
```

After the network is defined, a summary of the layers is printed. We can see that as expected, the output of the Embedding layer is a 4×8 matrix and this is squashed to a 32-element vector by the Flatten layer.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 8)              400
_________________________________________________________________
flatten_1 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 433
Trainable params: 433
Non-trainable params: 0
_________________________________________________________________
```

Finally, the accuracy of the trained model is printed, showing that it learned the training dataset perfectly (which is not surprising).

```
Accuracy: 100.000000
```

You could save the learned weights from the Embedding layer to file for later use in other models.

You could also use this model generally to classify other documents that have the same kind vocabulary seen in the test dataset.

Next, let’s look at loading a pre-trained word embedding in Keras.

4. Example of Using Pre-Trained GloVe Embedding

The Keras Embedding layer can also use a word embedding learned elsewhere.

It is common in the field of Natural Language Processing to learn, save, and make freely available word embeddings.

For example, the researchers behind GloVe method provide a suite of pre-trained word embeddings on their website released under a public domain license. See:

- GloVe: Global Vectors for Word Representation

The smallest package of embeddings is 822Mb, called "*glove.6B.zip*". It was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words. There are a few different embedding vector sizes, including 50, 100, 200 and 300 dimensions.

You can download this collection of embeddings and we can seed the Keras *Embedding* layer with weights from the pre-trained embedding for the words in your training dataset.

This example is inspired by an example in the Keras project: pretrained_word_embeddings.py.

After downloading and unzipping, you will see a few files, one of which is "*glove.6B.100d.txt*", which contains a 100-dimensional version of the embedding.

If you peek inside the file, you will see a token (word) followed by the weights (100 numbers) on each line. For example, below are the first line of the embedding ASCII text file showing the embedding for "*the*".

```
the -0.038194 -0.24487 0.72812 -0.39961 0.083172 0.043953 -0.39141 0.3344 -0.57545
```

As in the previous section, the first step is to define the examples, encode them as integers, then pad the sequences to be the same length.

In this case, we need to be able to map words to integers as well as integers to words.

Keras provides a Tokenizer class that can be fit on the training data, can convert text to sequences consistently by calling the *texts_to_sequences()* method on the *Tokenizer* class, and provides access to the dictionary mapping of words to integers in a *word_index* attribute.

```
# define documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(docs)
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(docs)
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

Next, we need to load the entire GloVe word embedding file into memory as a dictionary of word to embedding array.

```
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

This is pretty slow. It might be better to filter the embedding for the unique words in your training data.

Next, we need to create a matrix of one embedding for each word in the training dataset. We can do that by enumerating all unique words in the *Tokenizer.word_index* and locating the embedding weight vector from the loaded GloVe embedding.

The result is a matrix of weights only for words we will see during training.

```
# create a weight matrix for words in training docs
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Now we can define our model, fit, and evaluate it as before.

The key difference is that the embedding layer can be seeded with the GloVe word embedding weights. We chose the 100-dimensional version, therefore the Embedding layer must be defined with *output_dim* set to 100. Finally, we do not want to update the learned word weights in this model, therefore we will set the *trainable* attribute for the model to be *False*.

```
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=4, trainabl
```

The complete worked example is listed below.

```
from numpy import array
from numpy import asarray
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
# define documents
docs = ['Well done!',
        'Good work',
        'Great effort',
        'nice work',
        'Excellent!',
        'Weak',
        'Poor effort!',
        'not good',
        'poor work',
        'Could have done better.']
# define class labels
labels = array([1,1,1,1,1,0,0,0,0,0])
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(docs)
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(docs)
print(encoded_docs)
# pad documents to a max length of 4 words
max_length = 4
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
# load the whole embedding into memory
embeddings_index = dict()
f = open('../glove_data/glove.6B/glove.6B.100d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
# create a weight matrix for words in training docs
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

```
# define model
model = Sequential()
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=4, trainab
model.add(e)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
# summarize the model
print(model.summary())
# fit the model
model.fit(padded_docs, labels, epochs=50, verbose=0)
# evaluate the model
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Accuracy: %f' % (accuracy*100))
```

Running the example may take a bit longer, but then demonstrates that it is just as capable of fitting this simple problem.

```
[[6, 2], [3, 1], [7, 4], [8, 1], [9], [10], [5, 4], [11, 3], [5, 1], [12, 13, 2, 1

[[ 6  2  0  0]
 [ 3  1  0  0]
 [ 7  4  0  0]
 [ 8  1  0  0]
 [ 9  0  0  0]
 [10  0  0  0]
 [ 5  4  0  0]
 [11  3  0  0]
 [ 5  1  0  0]
 [12 13  2 14]]

Loaded 400000 word vectors.

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 100)            1500
_________________________________________________________________
flatten_1 (Flatten)          (None, 400)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 401
=================================================================
Total params: 1,901
Trainable params: 401
Non-trainable params: 1,500
_________________________________________________________________


Accuracy: 100.000000
```

In practice, I would encourage you to experiment with learning a word embedding using a pre-trained embedding that is fixed and trying to perform learning on top of a pre-trained embedding.

See what works best for your specific problem.

Further Reading

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

- Word Embedding on Wikipedia
- Keras Embedding Layer API
- Using pre-trained word embeddings in a Keras model, 2016
- Example of using a pre-trained GloVe Embedding in Keras
- GloVe Embedding
- An overview of word embeddings and their connection to distributional semantic models, 2016
- Deep Learning, NLP, and Representations, 2014

Summary

In this tutorial, you discovered how to use word embeddings for deep learning in Python with Keras.

Specifically, you learned:

- About word embeddings and that Keras supports word embeddings via the Embedding layer.
- How to learn a word embedding while fitting a neural network.
- How to use a pre-trained word embedding in a neural network.

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

Skip the Academics. Just Results.

Click to learn more.

Tweet Share Share G+

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 →

< How to Prepare Text Data for Deep Learning with Keras

How to Develop Word Embeddings in Python with Gensim >

255 Responses to *How to Use Word Embedding Layers for Deep Learning with Keras*

Mohammad October 4, 2017 at 7:58 am # REPLY

Thank you Jason,
I am excited to read more NLP posts.

Jason Brownlee October 4, 2017 at 8:03 am # REPLY

Thanks.

shiv October 5, 2017 at 10:07 am # REPLY

I split my data into 80-20 test-train and I'm still getting 100% accuracy. Any idea why? It is ~99% on epoch 1 and the rest its 100%.

Jason Brownlee October 5, 2017 at 5:22 pm # REPLY

Consider using the procedure in this post to evaluate your model:
https://machinelearningmastery.com/evaluate-skill-deep-learning-models/

trulia October 6, 2017 at 12:47 pm # REPLY

Use drop-out 20%, your model is overfit!!

Sandy October 6, 2017 at 2:44 pm # REPLY

Thank you Jason. I always find things easier when reading your post.
I have a question about the vector of each word after training. For example, the word "done" in sentence "Well done!" will be represented in different vector from that word in sentence "Could have done better!". Is that right? I mean the presentation of each word will depend on the context of each sentence?

Jason Brownlee October 7, 2017 at 5:48 am # REPLY

No, each word in the dictionary is represented differently, but the same word in different contexts will have the same representation.

It is the word in its different contexts that is used to define the representation of the word.

Does that help?

Sandy October 7, 2017 at 5:37 pm # REPLY

Yes, thank you. But I still have a question. We will train each context separately, then after training the first context, in this case is "Well done!", we will have a vector representation of the word "done". After

training the second context, “Could have done better”, we have another vector representation of the word “done”. So, which vector will we choose to be the representation of the word “done”?
I might misunderstand the procedure of training. Thank you for clarifying it for me.

Jason Brownlee October 8, 2017 at 8:32 am # REPLY

No. All examples where a word is used are used as part of the training of the representation of the word. There is only one representation for each word during and after training.

Sandy October 8, 2017 at 2:46 pm #

I got it. Thank you, Jason.

Chiedu October 7, 2017 at 5:36 pm # REPLY

Hi Jason,
any ideas on how to “filter the embedding for the unique words in your training data” as mentioned in the tutorial?

Jason Brownlee October 8, 2017 at 8:32 am # REPLY

The mapping of word to vector dictionary is built into Gensim, you can access it directly to retrieve the representations for the words you want: model.wv.vocab

mahna April 28, 2018 at 2:31 am # REPLY

HI Jason,
I am really appreciated the time U spend to write this tutorial and also replying.
My question is about “model.wv.vocab” you wrote. is it an address site? It does not work actually.

Jason Brownlee April 28, 2018 at 5:33 am # REPLY

No, it is an attribute on the model.

Abbey October 8, 2017 at 2:19 am # REPLY

Hi, Jason

Good day.

I just need your suggestion and example. I have two different dataset, where one is structured and the other is unstructured. The goal is to use the structured to construct a representation for the unstructured, so apply use word embedding on the two input data but how can I find the average of the two embedding and flatten it to one before feeding the layer into CNN and LSTM.

Looking forward to your response.
Regards
Abbey

Jason Brownlee October 8, 2017 at 8:40 am # REPLY

Sorry, what was your question?

If your question was if this is a good approach, my advice is to try it and see.

Abiodun Modupe October 9, 2017 at 7:46 pm # REPLY

Hi, Jason
How can I find the average of the word embedding from the two input?
Regards
Abbey

Jason Brownlee October 10, 2017 at 7:43 am # REPLY

Perhaps you could retrieve the vectors for each word and take their average?

Perhaps you can use the Gensim API to achieve this result?

Vinu October 9, 2017 at 5:54 pm # REPLY

Hi Jason…Could you also help us with R codes for using Pre-Trained GloVe Embedding

Jason Brownlee October 10, 2017 at 7:43 am # REPLY

Sorry, I don't have R code for word embeddings.

Hao October 12, 2017 at 5:49 pm # REPLY

Hi Jason, really appreciate that you answered all the replies! I am planning to try both CNN and RNN (maybe LSTM & GRU) on text classification. Most of my documents are less than 100 words long, but about 5 % are longer than 500 words. How do you suggest to set the max length when using RNN?If I set it to be 1000, will it degrade the learning result? Should I just use 100? Will it be different in the case of CNN?
Thank you!

Jason Brownlee October 13, 2017 at 5:45 am # REPLY

I would recommend experimenting with different configurations and see how the impact model skill.

ammara May 10, 2018 at 2:47 am # REPLY

Dear Hao,
Did you try RNN(LSTM or GRU) on text classification?If yes then can you plz provide me the code??

Jason Brownlee May 10, 2018 at 6:34 am # REPLY

Here is an example:
https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Michael October 13, 2017 at 10:22 am # REPLY

I'd like to thank you for this post. I've been struggling to understand this precise way of using keras for a week now and this is the only post I've found that actually explains what each step in the process is doing – and provides code that self-documents what the data looks like as the model is constructed and trained. This makes it so much easier to adapt to my particular requirements.

Jason Brownlee October 13, 2017 at 2:55 pm # REPLY

Thanks, I'm glad it helped.

Azim October 17, 2017 at 5:47 pm # REPLY

In the above Keras example, how can we predict a list of context words given a word? Lets say i have a word named 'sudoku' and want to predict the sourrounding words. how can we use word2vec from keras to do that?

Jason Brownlee October 18, 2017 at 5:32 am # REPLY

It sounds like you are describing a language model. We can use LSTMs to learn these relationships.

Here is an example:

https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/

Azim October 21, 2017 at 9:34 pm # REPLY

No, what i meant was for word2vec skip-gram model predicts a context word given the center word. So if i train a word2vec skip-gram model, how can i predict the list of context words if my center word is 'sudoku'?

Regards,

Azim

Jason Brownlee October 22, 2017 at 5:19 am # REPLY

I don't know Azim.

Kevin Toms November 16, 2018 at 7:20 pm # REPLY

You can get the cosine distance between the words, and the one that is having the least distance would surround it.. here is the link: https://github.com/Hvass-Labs/TensorFlow-Tutorials

Go to Natural Language Processing and you can find a cosine function there, use them to find yours..

Willie October 21, 2017 at 5:55 pm # REPLY

Hi Jason,

Thanks for your useful blog I have learned a lots.

I am wondering if I already have pretrained word embedding, is that possible to set keras embedding layer trainable to be true? If it is workable, will I get a better result, when I only use small size of data to pretrain the word embedding model. Many thanks!

REPLY

Jason Brownlee October 22, 2017 at 5:16 am #

You can. It is hard to know whether it will give better results. Try it.

cam October 28, 2017 at 5:34 am #

REPLY

Hey Jason,

Is it possible to perform probability calculations on the label? I am looking at a case where it is not simply +/- but that a given data entry could be both but more likely one and not the other.

Jason Brownlee October 29, 2017 at 5:48 am #

REPLY

Yes, a neural network with a sigmoid or softmax output can predict a probability-like score for each class.

David Stancu November 3, 2017 at 6:10 am #

REPLY

I'm doing something like this except with my own feature vectors — but to the point of the labels — I do ternary classification using categorical_crossentropy and a softmax output. I get back an answer of the probability of each label.

Jason Brownlee November 3, 2017 at 2:15 pm #

REPLY

Nice!

Ravil November 3, 2017 at 5:43 am #

REPLY

Hey Jason!

Thanks for a wonderful and detailed explanation of the post. It helped me a lot.

However, I'm struggling to understand how the model predicts a sentence as positive or negative.
i understand that each word in the document is converted into a word embedding, so how does our model evaluate the entire sentence as positive or negative? Does

it take the sum of all the word vectors? Perhaps average of them? I've not been able to figure this part out.

Jason Brownlee November 3, 2017 at 2:13 pm # REPLY

Great question!

The model interprets all of the words in the sequence and learns to associate specific patterns (of encoded words) with positive or negative sentiment

Ken April 2, 2018 at 8:37 am # REPLY

Hi Jason,

Thanks a lot for your amazing posts. I have the same question as Ravil. Can you elaborate a bit more on "learns to associate specific patterns?"

Jason Brownlee April 2, 2018 at 2:49 pm # REPLY

Good question Ken, perhaps this post will make it clearer how ml algorithms work (a functional mapping): http://machinelearningmastery.com/how-machine-learning-algorithms-work/

Does that help?

Ken April 2, 2018 at 10:40 pm #

Thanks for your reply. But I was trying to ask is that how does keras manage to produce a document level representation by having the vectors of each word? I don't seem to find how was this being done in the code.

Cheers.

Jason Brownlee April 3, 2018 at 6:34 am #

The model such as the LSTM or CNN will put this together.

In the case of LSTMs, you can learn more here:
https://machinelearningmastery.com/start-here/#lstm

Does that help?

Alexi September 27, 2018 at 1:48 am #

Hi Jason,

First, thanks for all your really useful posts.

If I understand well your post and answers to Ken and Ravil, the neural network you build in fact reduces the sequence of embedding vectors corresponding to all the words of a document to a one-dimensional vector with the Flatten layer, and you just train this flattening, as well as the embedding, to get the best classification on your training set, isn't it?

Thank you in advance for your answer.

Jason Brownlee September 27, 2018 at 6:04 am #

Sort of.

words => integers => embedding

The embedding has a vector per word which the network will use as a representation for the word.

We have a sequence of vectors for the words, so we flatten this sequence to one long vector for the Dense model to read. Alternately, we could wrap the dense in a timedistributed layer.

Alexi September 27, 2018 at 5:49 pm #

Aaah! So nothing tricky is done when flattening, more or less just concatenating the fixed number of embedding vectors that is the output of the embedding layer, and this is why the number of words per document has to be fixed as a setting of this layer. If this is correct, I think I'm finally understanding how all this works.

I'm sorry to bother you more, but how does the network works if a document much shorter than the longest document (the number of its words being set as the number of words per document to the embedding layer) is given to the network as training or testing? It just fills the embedding vectors of this non-appearing words as 0? I've been looking for ways to convert all the word embeddings of a text to some sort of document embedding, and this just seems a solution too simple to work, or that may work but for short documents (as well as other options like averaging the word vectors or taking the element-wise maximum of minimum).

I'm trying to do sentiment analysis for spanish news, and I have news with like 1000 or more words, and wanted to use pre-trained word embeddings of 300 dimensions each. Wouldn't it be a size way too huge per document for the network to train properly, or fast enough? I imagine you do not have a precise answer, but I'd like to know if you have tried the above method with long documents, or know that someone has.

Thank you again, I'm sorry for such a long question.

Jason Brownlee September 28, 2018 at 6:07 am #

Yes.

We can use padding for small documents and a Masking input layer to ignore padded values. More here: https://machinelearningmastery.com/handle-missing-timesteps-sequence-prediction-problems-python/

Try different sized embeddings and use results to guide the configuration.

Alexi October 1, 2018 at 5:26 pm #

Okay, thank you very much! I will give it a try.

chengcheng November 9, 2017 at 2:56 am # REPLY

the chinese word how to vector sequence

Jason Brownlee November 9, 2017 at 10:03 am # REPLY

Sorry?

lstmbot December 16, 2017 at 10:30 pm # REPLY

me bot trying interact comments born with lstm

Hilmi Jauffer November 16, 2017 at 4:30 pm # REPLY

Hi Jason,
I have successfully trained a model using the word embedding and Keras. The accuracy was at 100%.

I saved the trained model and the word tokens for predictions.
MODEL.save('model.h5', True)

TOKENIZER = Tokenizer(num_words=MAX_NB_WORDS)
TOKENIZER.fit_on_texts(TEXT_SAMPLES)
pickle.dump(TOKENIZER, open('tokens', 'wb'))

When predicting:
– Load the saved model.
– Setup the tokenizer, by loading the saved word tokens.
– Then predict the category of the new data.

I am not sure the prediction logic is correct, since I am not seeing the expected category from the prediction.

The source code is in Github: https://github.com/hilmij/keras-test/blob/master/predict.py

Appreciate if you can have a look and let me know what I am missing.

Best regards,
Hilmi.

Jason Brownlee November 17, 2017 at 9:20 am # REPLY

Your process sounds correct. I cannot review your code sorry.

What was the problem exactly?

Tony July 11, 2018 at 8:00 am # REPLY

Thank you, Jason! Your examples are very helpful. I hope to get your attention with my question. At training, you prepare the tokenizer by doing:

t = text.Tokenizer();
t.fit_on_texts(docs)

Which creates a dictionary of words:numbers. What do we do if we have a new doc with lost of new words at prediction time? Will all these words go the unknown token? If so, is there a solution for this, like can we fit the tokenizer on all the words in the English vocab?

Jason Brownlee July 11, 2018 at 2:52 pm # REPLY

You must know the words you want to support at training time. Even if you have to guess.

To support new words, you will need a new model.

Fabrício Melo November 17, 2017 at 7:35 am # REPLY

Hello Jason!

In Example of Using Pre-Trained GloVe Embedding, do you use the word embedding vectors as weights of the embedding layer?

Jason Brownlee November 17, 2017 at 9:30 am # REPLY

Yes.

Alex November 21, 2017 at 11:15 pm # REPLY

Very nice set of Blogs of NLP and Keras – thanks for writing them.

As a quick note for others

When I tried to load the glove file with the line:
f = open('../glove_data/glove.6B/glove.6B.100d.txt')

I got the error
UnicodeDecodeError: 'charmap' codec can't decode byte 0x9d in position 2776: character maps to

To fix I added:
f = open('../glove_data/glove.6B/glove.6B.100d.txt',encoding="utf8″)

This issue may have been caused by using Windows.

Jason Brownlee November 22, 2017 at 11:12 am # REPLY

Thanks for the tip Alex.

Liliana November 26, 2017 at 12:00 pm # REPLY

Hi Jason,

Wonderful tutorials!

I have a question. Why do we have to one-hot vectorize the labels? Also, if I have a pad sequence of ex. [2,4,0] what the one hot will be? I'm trying to understand better one hot vectorzer.

I appreciate your response!

Jason Brownlee November 27, 2017 at 5:47 am # REPLY

We don't one hot encode the labels, we one hot encode the words.

Perhaps this post will help you better understand one hot encoding:
https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

Wassim November 28, 2017 at 1:06 am # REPLY

Hi Jason,
Thank you for your excellent tutorial. Do you know if there is a way to build a network for a classification using both text embedded data and categorical data ?
Thank you

Jason Brownlee November 28, 2017 at 8:37 am # REPLY

Sure, you could have a network with two inputs:
https://machinelearningmastery.com/keras-functional-api-deep-learning/

Wassim November 28, 2017 at 10:50 pm # REPLY

Thank you Jason

ashish December 2, 2017 at 8:29 pm # REPLY

How to do sentence classification using CNN in keras ? please help

Jason Brownlee December 3, 2017 at 5:24 am # REPLY

See this tutorial:
https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/

Stuart December 7, 2017 at 12:11 am # REPLY

Fantastic explanation, thanks so much. I'm just amazed at how much easier this has become since the last time I looked at it.

Jason Brownlee December 7, 2017 at 7:59 am # REPLY

I'm glad the post helped Stuart!

Stuart December 11, 2017 at 3:30 pm # REPLY

Hi Jason …the 14 word vocab from your docs is "well done good work great effort nice excellent weak poor not could have better" For a vocab_size of 14, this one_shot encodes to [13 8 7 6 10 13 3 6 10 4 9 2 10 12]. Why does 10 appear 3 times, for "great", "weak" and "have"?

Jason Brownlee December 11, 2017 at 4:54 pm # REPLY

Sorry, I don’t follow Stuart. Could you please restate the question?

Stuart December 11, 2017 at 9:34 pm # REPLY

Hi Jason, the encodings that I provided in the example above came from kerasR with a vocab_size of 14. So let me ask the same question about the uniqueness of encodings using your Part 3 one_hot example above with a vocab_size of 50.
Here different encodings are produced for different new kernels (using Spyder3/Python 3.4):
[[31, 33], [27, 33], [48, 41], [34, 33], [32], [5], [14, 41], [43, 27], [14, 33], [22, 26, 33, 26]]
[[6, 21], [48, 44], [7, 26], [46, 44], [45], [45], [10, 26], [45, 48], [10, 44], [47, 3, 21, 27]]
[[7, 8], [16, 42], [24, 13], [45, 42], [23], [17], [34, 13], [13, 16], [34, 42], [17, 31, 8, 19]]

Pleas note that in the first line, “33” encodes for the words “done”, “work”, “work”, “work” & “done”. In the second line “45” encodes for the words “excellent” & “weak” & “not”. In the third line, “13” encodes “effort”, “effort” & “not”.

So I’m wondering why the encodings are not unique? Secondly, if vocab_size must be much larger then the actual size of the vocabulary?

Thanks

Jason Brownlee December 12, 2017 at 5:33 am # REPLY

The one_hot() function does not map words to unique integers, it uses a hash function that can have collisions, learn more here
https://keras.io/preprocessing/text/#one_hot

Stuart December 12, 2017 at 7:53 am # REPLY

Thanks Jason, In your Part 4 example, the Tokenizer approach always gives the same encodings and these appear to be unique.

Jason Brownlee December 12, 2017 at 4:03 pm # REPLY

Yes, I recommend using Tokenizer, see this post for more:
https://machinelearningmastery.com/prepare-text-data-deep-learning-keras/

Nadav December 25, 2017 at 8:28 am # REPLY

Great article Jason.
How do you convert back from an embedding to a one-hot? For example if you have a seq2seq model, and you feed the inputs as word embeddings, in your decoder you need to convert back from the embedding to a one-hot representing the dictionary. If you do it by using matrix multiplication that can be quite a large matrix (e.g embedding size 300, and vocab of 400k).

Jason Brownlee December 26, 2017 at 5:12 am # REPLY

The output layer can predict integers directly that you can map to words in your vocabulary. There would be no embedding layer on the output.

Hitkul January 9, 2018 at 9:05 pm # REPLY

Hi,
Very helpful article.
I have created word2vec matrix of a sentence using gensim and pre-trained Google News vector. Can I just flatten this matrix to a vector and use that as a input to my neural network.
For example:
each sentence is of length 140 and I am using a pre-trained model of 100 dimensions, therefore:- I have a 140*100 matrix representing the sentence, can i just flatten it to a 14000 length vector and feed it to my input layer?

Jason Brownlee January 10, 2018 at 5:25 am # REPLY

It depends on what you're trying to model.

Paul January 12, 2018 at 6:40 pm #

REPLY

Great article, could you shed some light on how do Param # of 400 and 1500 in two neural networks come from? Thanks

Paul Lo January 12, 2018 at 9:55 pm #

REPLY

Oh! Is it just vocab_size * # of dimension of embedding space?

1. 50 * 8 = 400
2. 15* 100 = 1500

Jason Brownlee January 13, 2018 at 5:31 am #

REPLY

What do you mean exactly?

Andy Brown January 14, 2018 at 2:02 pm #

REPLY

Great post! I'm working with my own corpus. How would I save the weight vector of the embedding layer in a text file like the glove data set?

My thinking is it would be easier for me to apply the vector representations to new data sets and/or machine learning platforms (mxnet etc) and make the output human readable (since the word is associated with the vector).

Jason Brownlee January 15, 2018 at 6:57 am #

REPLY

You could use get_weights() in the Keras API to retrieve the vectors and save directly as a CSV file.

jae January 17, 2018 at 8:10 am #

REPLY

Clear Short Good reading, always thank you for your work!

Jason Brownlee January 17, 2018 at 10:01 am #

REPLY

Thanks.

Murali Manohar January 17, 2018 at 4:58 pm # REPLY

Hello Jason,
I have a dataset with which I've attained 0.87 fscore by 5 fold cross validation using SVM.Maximum context window is 20 and one hot encoded.

Now, I've done whatever has been mentioned and getting an accuracy of 13-15 percent for RNN models where each one has one LSTM cell with 3,20,150,300 hidden units. Dimensions of my pre-trained embeddings is 300.

Loss is decreasing and even reaching negative values, but no change in accuracy.

I've tried the same with your CNN,basic ANN models you've mentioned for text classification .

Would you please suggest some solution. Thanks in advance.

Jason Brownlee January 18, 2018 at 10:05 am # REPLY

I have some ideas here:
http://machinelearningmastery.com/improve-deep-learning-performance/

Carsten January 17, 2018 at 8:35 pm # REPLY

When I copy the code of the first box I get the error:

AttributeError: 'int' object has no attribute 'ndim'

in the line :

model.fit(padded_docs, labels, epochs=50, verbose=0)

Where is the problem?

Jason Brownlee January 18, 2018 at 10:07 am # REPLY

Copy the the code from the "complete example".

Thiziri February 8, 2018 at 12:45 am # REPLY

Hi Jason,
I've got the same error, also will running the "compete example".
What can be the cause?

Gokul February 8, 2018 at 6:49 pm # REPLY

Try casting the labels to numpy arrays.

soren February 9, 2018 at 6:31 am # REPLY

i get the same!

Jason Brownlee February 9, 2018 at 9:22 am #

I have fixed and updated the examples.

ademyanchuk February 8, 2018 at 3:34 pm # REPLY

Carsten, you need labels to be numpy.array not just list.

Willie January 17, 2018 at 9:18 pm # REPLY

Hi Jason,

If I have unkown words in training set, how can I assign the same random initialize vector to all of the unkown words when using pretrained vector model like glove or w2v. thanks!!!

Jason Brownlee January 18, 2018 at 10:08 am # REPLY

Why would you want to do that?

Willie January 18, 2018 at 1:05 pm #

REPLY

If my data is in specific domain and I still want to leverage general word embedding model(e.g. glove.6b.100d trained from wiki), then it must has some OOV in domain data, so. no no mather in training time or inference time it propably may appear some unkown words.

Jason Brownlee January 19, 2018 at 6:27 am #

REPLY

It may.

You could ignore these words.

You could create a new embedding, set vectors from existing embedding and learn the new words.

Vladimir January 21, 2018 at 1:46 pm #

REPLY

Amazing Dr. Jason!
Thanks for a great walkthrough.

Kindly advice on the following.
On the step of encoding each word to integer you said: "We could experiment with other more sophisticated bag of word model encoding like counts or TF-IDF". Could you kindly elaborate on how can it be implemented, as tfidf encodes tokens with floats. And how to tie it with Keras, passing it to an Embedding layer please? I'm keen to experiment with it, hope it could yield better results.

Another question is about input docs. Suppose I've preprocessed text by the means of nltk up to lemmatization, thus, each sample is a list of tokens. What is the best approach to pass it to Keras embedding layer in this case then?

Jason Brownlee January 22, 2018 at 4:42 am #

REPLY

I have most posts on BoW, learn more here:
https://machinelearningmastery.com/?s=bag+of+words&submit=Search

You can encode your tokens as integers manually or use the Keras Tokenizer.

Vladimir January 22, 2018 at 11:54 pm # REPLY

Well, Keras Tokenizer can accept only texts or sequences. Seems the only way is to glue tokens together using ' '.join(token_list) and then pass onto the Tokenizer.

As for the BOW articles, I've walked through it theys are so very valuable. Thank you!

Using BOW differs so much from using Embeddings. As BOW would introduce huge sparse array of features for each sample, while Embeddings aim to represent those features (tokens) very densely up to a few hundreds items.

So, BOW in the other article gives incredibly good results with just very simple NN architecture (1 layer of 50 or 100 neurons). While I struggled to get good results using Embeddings along with convolutional layers…
From your experience, would you please advice on that please? Are Embeddings actually viable and it is just a matter of finding a correct architecture?

Jason Brownlee January 23, 2018 at 8:01 am # REPLY

Nice! And well done for working through the tutorials. I love to see that and few people actually "do the work".

Embeddings make more sense on larger/hard problems generally – e.g. big vocab, complex language model on the front end, etc.

Vladimir January 23, 2018 at 10:15 am #

I see, thank you.

joseph January 31, 2018 at 9:20 pm # REPLY

Thanks jason for another great tutorial.

I have some questions :

Isn't one hot definition is binary one, vector of 0's and 1?
so [[1,2]] would be encoded to [[0,1,0],[0,0,1]]

how embedding algorithm is done on keras word2vec/globe or simply dense

layer(or something else)
thanks
joseph

Jason Brownlee February 1, 2018 at 7:20 am # REPLY

Sorry, I don't follow your question. Perhaps you could rephrase it?

Anna February 4, 2018 at 9:40 pm # REPLY

Amazing Dr. Jason!
Thanks for a great walkthrough.

The dimension for each word vector like above example e.g. 8, is set randomly?

Thank you

Jason Brownlee February 5, 2018 at 7:45 am # REPLY

The dimensionality is fixed and specified.

In the first example it is 8, in the second it is 100.

Anna February 5, 2018 at 2:17 pm # REPLY

Thank you Dr. Jason for your quick feedback!

Ok, I see that the pre-trained word embedding is set to 100 dimensionality because the original file "glove.6B.100d.txt" contained a fixed number of 100 weights for each line of ASCII.

However, the first example as you mentioned in here, "The Embedding has a vocabulary of 50 and an input length of 4. We will choose a small embedding space of 8 dimensions."

You choose 8 dimensions for the first example. Does it means it can be set to any numbers other than 8? I've tried to change the dimension to 12. It doesn't appeared any errors but the accuracy drops from 100% to 89%

Layer (type) Output Shape Param #

```
=================================================
=====================
embedding_1 (Embedding) (None, 4, 12) 600

_________________________________________________
flatten_1 (Flatten) (None, 48) 0

_________________________________________________
dense_1 (Dense) (None, 1) 49
=================================================
=====================
Total params: 649
Trainable params: 649
Non-trainable params: 0
```

Accuracy: 89.999998

So, how dimensionality is set? Does the dimensions effects the accuracy performance?

Sorry I am trying to grasp the basic concept in understanding NLP stuff. Much appreciated for your help Dr. Jason.

Thank you

Jason Brownlee February 5, 2018 at 2:54 pm # REPLY

No problem, please ask more questions.

Yes, you can choose any dimensionality you like. Larger means more expressive, required for larger vocabs.

Does that help Anna?

Anna February 5, 2018 at 4:40 pm #

Yes indeed Dr. Now I can see that the dimensionality is set depends on the number of vocabs.

Thank you again Dr Jason for enlighten me! 🙂

Jason Brownlee February 6, 2018 at 9:11 am #

You asked good questions.

Miroslav February 5, 2018 at 7:09 am # REPLY

Hi Jason,
thanks for amazing tutorial.

I have a question. I am trying to do semantic role labeling with context window in Keras. How can I implement context window with embedding layer?

Thank you

Jason Brownlee February 5, 2018 at 7:54 am # REPLY

I don't know, sorry.

Gabriel February 6, 2018 at 4:08 am # REPLY

Hi, great website! I've been learning a lot from all the tutorials. Thank you for providing all these easy to understand information.

How would I go about using other data for the CNN model? At the moment, I am using just textual data for my model using the word embeddings. From what I understand, the first layer of the model has to be the Embeddings, so how would I use other input data such as integers along with the Embeddings?

Thank you!

Jason Brownlee February 6, 2018 at 9:21 am # REPLY

Great question!

You can use a multiple-input model, see examples here:
https://machinelearningmastery.com/keras-functional-api-deep-learning/

Gabriel February 6, 2018 at 4:22 pm # REPLY

Thank you for the fast reply!

Aditya February 6, 2018 at 4:59 am # REPLY

Hi Jason, this tutorial is simple and easy to understand. Thanks.

However, I have a question. While using the pre-trained embedding weights such as Glove or word2vec, what if there exists few words in my dataset, which weren't present in the dataset on which word2vec or Glove was trained. How does the model represent such words?

My understanding is that in your second section (Using Pre-Trained Glove Embeddin), you are mapping the words from the loaded weights to the words present in your dataset, hence the question above.

Correct me, if it's not the way I think it is.

Jason Brownlee February 6, 2018 at 9:22 am # REPLY

You can ignore them, or assign them to zero vector, or train a new model that includes them.

Han February 20, 2018 at 4:03 pm # REPLY

Hi Jason,

I am trying to train a Keras LSTM model on some text sentiment data. I am also using GridSearchCV in sklearn to find the best parameters. I am not quite sure what went wrong but the classification report from sklearn says:

UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

Below is what the classification report looks like:

precision recall f1-score support

negative 0.00 0.00 0.00 98
positive 0.70 1.00 0.83 232

avg / total 0.49 0.70 0.58 330

Do you know what the problem is?

Jason Brownlee February 21, 2018 at 6:35 am # REPLY

Perhaps you are trying to use keras metrics with sklearn? Watch the keywords you use when specifying the keras model vs the sklearn evaluation (CV).

Alberto Nogales Moyano March 2, 2018 at 2:10 am # REPLY

Hi Jason,
your blog is really really interesting. I have a question: Which is the diffrence between using word2vec and texts_to_sequences from Tokenizer in keras? I mean in the way the texts are represented.
Is any of the two options better than the other?
Thanks a lot.
Kind regards.

Jason Brownlee March 2, 2018 at 5:33 am # REPLY

word2vec encodes words (integers) to vectors. texts_to_seqences encodes words to integers. It is a step before word2vec or a step before bag of words.

Souraj Adhikary March 2, 2018 at 5:39 pm # REPLY

Hi Jason,

I have a dataframe which contains texts and corresponding labels. I have used gensim module and used word2vec to make a model from the text. Now I want to use that model for input into Conv1D layers. Can you please tell me how to load the word2vec model in Keras Embedding layer? Do I need to pre-process the model in some way before loading? Thanks in advance.

Jason Brownlee March 3, 2018 at 8:07 am # REPLY

Yes, load the weights into an Embedding layer, then define the rest of your network.

The tutorial above will help a lot.

Ankush Chandna March 13, 2018 at 1:57 am # REPLY

This is really helpful. You make us awesome at what we do. Thanks!!

Jason Brownlee March 13, 2018 at 6:31 am # REPLY

I'm glad to hear that.

R.L. March 20, 2018 at 4:13 pm # REPLY

Thank you for this extremely helpful blog post. I have a question regarding to interpreting the model. Is there a way to know / visualize the word importance after the model is trained? I am looking for a way to do so. For instance, is there a way to find like the top 10 words that would trigger the model to classify a text as negative and vice versa? Thanks a lot for your help in advance

Jason Brownlee March 21, 2018 at 6:31 am # REPLY

There may be methods, but I am not across them. Generally, neural nets are opaque, and even weight activations in the first/last layers might be misleading if used as importance scores.

Alex September 1, 2018 at 12:05 am # REPLY

Maybe look at Lime
https://github.com/marcotcr/lime

Jason Brownlee September 1, 2018 at 6:22 am # REPLY

Thanks.

Mohit March 30, 2018 at 9:35 pm # REPLY

Hi Jason ,

Can you please tell me the logic behind this:

vocab_size = len(t.word_index) + 1

Why we added 1 here??

Jason Brownlee March 31, 2018 at 6:36 am # REPLY

So that the word indexes are 1-offset, and 0 is reserved for padding / no data.

Ryan March 31, 2018 at 9:59 pm # REPLY

Hi Jason,

If I want to use this model to predict next word, can I just change the output layer to Dense(100, activation = 'linear') and change the loss function to MSE?

Many thanks,

Ray

Jason Brownlee April 1, 2018 at 5:49 am # REPLY

Perhaps look at some of the posts on training a language model for word generation:
https://machinelearningmastery.com/?s=language+model&submit=Search

Coach April 17, 2018 at 10:23 pm # REPLY

Thanks for this tutoriel ! Really clear and usefull !

Jason Brownlee April 18, 2018 at 8:07 am # REPLY

You're welcome.

Maryam April 28, 2018 at 4:53 am # REPLY

Hi Jason,
U R the best in keras tutorials and also replying the questions. I am really grateful.
Although I have understood the content of the context and codes U have written above, I am not able to understand what you mean about this sentence:[It might be better to filter the embedding for the unique words in your training data.].
what does "to filter the embedding" mean??
Thank you for replying.

Jason Brownlee April 28, 2018 at 5:34 am # REPLY

It means, only have the words in the embedding that you know exist in your dataset.

Maryam April 28, 2018 at 11:45 pm # REPLY

Hi Jason,
Thank U 4 replying but as I am not a native English speaker, I am not sure whether I got it or not. You mean to remove all the words which exist in the glove but do not exist in my own dataset?? in order to raise the speed of implementation?
I am sorry to ask it again as I did not understand clearly.
Thank U in advance Jason

Jason Brownlee April 29, 2018 at 6:28 am # REPLY

Exactly.

Aiza May 9, 2018 at 6:14 am #

REPLY

Hi,
This post is great. I am new to machine learning so i have a question which might be basic so i am not sure.As from what i understand, the model takes the embedding matrix and text along with labels at once.What i am trying to do is concatenate POS tag embedding with each pre-trained word embedding but POS tag can be different for the same word depending upon the context.It essentially means that i cant alter the embedding matrix at add to the network embedding layer.I want to take each sentence,find its embedding and concatenate with POS tag embedding and then feed into neural network.Is there a way to do the training sentence by sentence or something? Thanks

Jason Brownlee May 9, 2018 at 6:32 am #

REPLY

You might be able to use the embedding and pre-calculate the vector representations for each sentence.

Aiza May 9, 2018 at 7:06 am #

REPLY

Sorry but i didn't quite understand.Can you please elaborate a little?

Jason Brownlee May 9, 2018 at 2:54 pm #

REPLY

Sorry, I mean that you can prepare an word2vec model standalone. Then pass each sentence through it to build up a list of vectors. Concat the vectors together and you have a distributed sentence representation.

Aiza May 9, 2018 at 10:16 pm #

Thanks alot! One more thing, is it possible to pass other information to the embedding layer than just weights?For example i was thinking that what if i dont change the embedding matrix at all and create a separate matrix of POS tags for whole training data which is also passed to the embedding layer which concatenates these both sequentially?

Jason Brownlee May 10, 2018 at 6:32 am #

You could develop a model that has multiple inputs, for example see this post for examples:
https://machinelearningmastery.com/keras-functional-api-deep-learning/

Aiza May 15, 2018 at 8:46 am #

REPLY

Thanks.I saw this post.Your model has separate inputs but they get merged after flattening.In my case i want to pass the embeddings to first convolutional layer,only after they are concatenated. Uptil now what i did was that i have created another integerized sequence of my data according to POS_tags(embedding_pos) to pass as another input and another embedding matrix that contains the embeddings of all the POS tags.
e=(Embedding(vocab_size, 50, input_length=23, weights=[embedding_matrix], trainable=False))
e1=(Embedding(38, 38, input_length=23, weights=[embedding_matrix_pos], trainable=False))
merged_input = concatenate([e,e1], axis=0)
model_embed = Sequential()
model_embed.add(merged_input)
model_embed.fit(data,embedding_pos, final_labels, epochs=50, verbose=0)

I know this is wrong but i am not sure how to concat those both sequences and if you can direct me in right direction,it would be great.The error is
'Layer concatenate_6 was called with an input that isn't a symbolic tensor. Received type: . Full input: [,]. All inputs to the layer should be tensors.'

Jason Brownlee May 15, 2018 at 2:42 pm #

REPLY

Perhaps you could experiment and compare the performance of models with different merge layers for combining the inputs.

Franco May 16, 2018 at 6:51 pm # REPLY

Hi Jason, awesome post as usual!

Your last sentence is tricky though. You write:

“In practice, I would encourage you to experiment with learning a word embedding using a pre-trained embedding that is fixed and trying to perform learning on top of a pre-trained embedding.”

Without the original corpus, I would argue, that’s impossible.

In Google’s case, the original corpus of around 100 billion words is not publicly available. Solution? I believe you’re suggesting “Transfer Learning for NLP.” In this case, the only solution I see is to add manually words.

E.g. you need ‘dolares’ which is not in Google’s Word2Vec. You want to have similar vectors as ‘money’. In this case, you add ‘dolares’ + 300 vectors from money. Very painful, I know. But it’s the only way I see to do “Transfer Learning with NLP”.

If you have a better solution, I’d love your input.

Cheerio, a big fan

Jason Brownlee May 17, 2018 at 6:30 am # REPLY

Not impossible, you can use an embedding trained on a other corpus and ignore the difference or fine tune the embedding while fitting your model.

You can also add missing words to the embedding and learn those.

Remember, we want a vector for each word that best captures their usage, some inconsistencies does not result in a useless model, it is not binary useful/useless case.

Franco May 17, 2018 at 4:42 pm # REPLY

Thank you very much for the detailed answer!

Ashar May 31, 2018 at 6:11 am # REPLY

The link for the Tokenizer API is this same webpage. Can you update it please?

Jason Brownlee May 31, 2018 at 6:26 am # REPLY

Fixed, thanks.

Andreas Papandreou May 31, 2018 at 10:55 am # REPLY

Hi Jason, great post!
I have successfully trained my model using the word embedding and Keras. I saved the trained model and the word tokens.Now in order to make some predictions, do i have to use the same tokenizer with one that i used in the training?

Jason Brownlee May 31, 2018 at 2:12 pm # REPLY

Correct.

Andreas Papandreou May 31, 2018 at 4:08 pm # REPLY

Thank you very much!

zedom June 2, 2018 at 5:15 pm # REPLY

Hi Jason, when i was looking for how to use pre-trained word embedding, I found your article along with this one:
https://jovianlin.io/embeddings-in-keras/
They have many similarities.

Jason Brownlee June 3, 2018 at 6:20 am # REPLY

Glad to hear it.

Jack June 13, 2018 at 10:58 pm #

REPLY

Hey jason,

I am trying to do this but sometime keras gives the same integer to different words. Would it be better to use scikit encoder that converts words to integers?

Jason Brownlee June 14, 2018 at 6:08 am #

REPLY

This might happen if you are using a hash encoder, as Keras does, but calls it a one hot encoder.

Perhaps try a different encoding scheme of words to integers

Meysam June 18, 2018 at 5:13 am #

REPLY

Hi Jason,
I have implemented the above tutorial and the code works fine with GloVe. I am so grateful abot the tutorial Jason.
but when I download GoogleNews-vectors-negative300.bin which is a pre-trained embedding word2vec it gave me this error:

File "/home/mary/anaconda3/envs/virenv/lib/python3.5/site-packages/gensim/models/keyedvectors.py", line 171, in __getitem__
return vstack([self.get_vector(entity) for entity in entities])

TypeError: 'int' object is not iterable.

I wrote the code as the same as your code which you wrote for loading glove but with a little change.

'model = gensim.models.KeyedVectors.load_word2vec_format('./GoogleNews-vectors-negative300.bin', binary=True)

for line in model:
values = line.split()
word = values[0]
coefs = asarray(values[1:], dtype='float32′)
embeddings_index[word] = coefs
model.close()
print('Loaded %s word vectors.' % len(embeddings_index))

embedding_matrix = zeros((vocab_dic_size, 300))
for word in vocab_dic.keys():
embedding_vector = embeddings_index.get(word)
if embedding_vector is not None:
embedding_matrix[vocab_dic[word]] = embedding_vector

I saw you wrote a tutorial about creating word2vec by yourself in this link "https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/",
but I have not seen a tutorial about aplying pre-trained word2vec like GloVe.
please guide me to solve the error and how to apply the GoogleNews-vectors-negative300.bin pretrained wor2vec?
I am so sorry to write a lot as I wanted to explain in detail to be clear.
any guidance will be appreciated.
Best
Meysam

Jason Brownlee June 18, 2018 at 6:46 am # REPLY

Perhaps try the text version as in the above tutorial?

Meysam June 19, 2018 at 12:28 am # REPLY

Hi Jason
thank you very much for replying, but as I am weak at the English language, the meaning of this sentence is not clear. what do you mean by "try the text version"??
in fact, GloVe contains txt files and I implement it correctly but when I wanna run a program by GoogleNews-vectors-negative300.bin which is a pre-trained embedding word2vec it gave me the error and also this file is a binary one and there is no pre-trained embedding word2vec file by txt prefix.
can you help me though I know you are busy?
Best
Meysam

Jason Brownlee June 19, 2018 at 6:36 am # REPLY

You are using the binary version of the glove file (.bin). Try downloading and using the text version instead.

You can get .txt versions here:
https://nlp.stanford.edu/projects/glove/

Kavit Gangar June 18, 2018 at 7:17 pm # REPLY

How can we use pre-trained word embedding on mobile?

Jason Brownlee June 19, 2018 at 6:29 am # REPLY

I don't see why not, other than disk/ram size issues.

NewToDeepNLP June 28, 2018 at 12:17 pm # REPLY

Great post! What changes are necessary if the labels are more than binary such as with 4 classes:
labels = array([2,1,1,1,2,0,-1,0,-1,0])
?
E.g. instead of 'binary_crossentropy' perhaps 'categorical_crossentropy'?
And how shold the Dense layer change?
If I use: model.add(Dense(4, activation='sigmoid')), I get an error:

ValueError: Error when checking target: expected dense_1 to have shape (None, 4) but got array with shape (10, 1)

thanks for your work!

Jason Brownlee June 28, 2018 at 2:12 pm # REPLY

I believe this will help:
https://machinelearningmastery.com/faq/single-faq/how-can-i-change-a-neural-network-from-regression-to-classification

NewToDeepNLP June 28, 2018 at 3:14 pm # REPLY

thanks! also using keras's to_categorical to discretize the labels was necessary.

NewToDeepNLP June 28, 2018 at 7:19 pm #

REPLY

one more question: is there a simply way to create the Tokenizer() instance, fit it, save it, and then extend it on new documents? Specifically, so that t.fit_on_texts( ) can be updated on new data.

Jason Brownlee June 29, 2018 at 5:53 am #

I'm not so sure that you can.

It might be easier to manage the encoding/mapping yourself so that you can extend it at will.

Jason Brownlee June 29, 2018 at 5:50 am #

REPLY

Nice.

James June 28, 2018 at 9:58 pm #

REPLY

Hi Jason,

For starters, thanks for this post. Ideal to get things going quickly. I have a couple of questions if you don't mind:

1) I don't think that one-hot encoding the string vectors is ideal. Even with the recommended vocab size (50), I still got collisions which defeats the purpose even in a toy example such as this. Even the documentation states that uniqueness is not guaranteed. Keras' Tokenizer(), which you used in the pre-trained example is a more reliable choice in that no two words will share the same integer mapping. How come you proposed one-hot encoding when Tokenizer() does the same job better?

2) Getting Tokenizer()'s word_index property, returns the full dictionary. I expected the vocab_size to be equal to len(t.word_index) but you increment that value by one. This is in fact necessary because otherwise fitting the model fails. But I cannot get the intuition of that. Why is the input dimension size equal to vocab_size + 1?

3) I created a model that expects a BoW vector representation of each "document". Naturally, the vectors were larger and sparser [(10,14)] which

means more parameters to learn, no. However, in your document you refer to this encoding or tf-idf as "more sophisticated". Why do you believe so? With that encoding don't you lose the word order which is important to learn word embeddings? For the record, this encoding worked well too but it's probably due to the nature of this little experiment.

Thank you in advance.

Jason Brownlee June 29, 2018 at 6:05 am # REPLY

The keras one hot encoding method really just takes a hash. It is better to use a true one hot encoding when needed.

I do prefer the Tokenizer class in practice.

The words are 1-offset, leaving room for 0 for "unknown" word.

tf-idf gives some idea of the frequency over the simple presence/absence of a word in BoW.

How that helps.

abbas July 12, 2018 at 3:27 am # REPLY

where can i find the file "../glove_data/glove.6B/glove.6B.100d.txt"??because i come up with the following error.
File "", line 36
f = open('../glove_data/glove.6B/glove.6B.100d.txt')
^
SyntaxError: invalid character in identifier

Jason Brownlee July 12, 2018 at 6:28 am # REPLY

You must download it and place it in your current working directory.

Perhaps re-read section 4.

abbas July 14, 2018 at 3:34 pm # REPLY

I have placed the code and dataset in same directory.what's wrong with the code??

f = open('glove.6B/glove.6B.100d.txt')
I am facing the following error.
File "", line 36
f = open('glove.6B/glove.6B.100d.txt')
^
SyntaxError: invalid character in identifier

Jason Brownlee July 15, 2018 at 6:08 am # REPLY

Perhaps this will help you when copying code from the tutorial:
https://machinelearningmastery.com/faq/single-faq/how-do-i-copy-code-from-a-tutorial

Harry July 17, 2018 at 1:02 pm # REPLY

Excellent work! This is quite helpful to novice.
And I wonder is this useful to other language apart from English? Since I am a Chinese, and I wonder whether I can apply this to Chinese language and vocabulary.
Thanks again for your time devoted!

Jason Brownlee July 17, 2018 at 2:31 pm # REPLY

I don't see why not.

sree harsha July 18, 2018 at 6:02 am # REPLY

Hi,

can you explain how can the word embeddings be given as hidden state input to LSTM?

thanks in advance

Jason Brownlee July 18, 2018 at 6:39 am # REPLY

Word embeddings don't have hidden state. They don't have any state.

sree harsha July 18, 2018 at 6:57 pm # REPLY

for example, I have a word and its 50 (dimensional) embeddings. How can I give these embeddings as hidden state input to an LSTM layer?

Jason Brownlee July 19, 2018 at 7:47 am # REPLY

Why would you want to give them as the hidden state instead of using them as input to the LSTM?

zbk July 20, 2018 at 8:39 am # REPLY

hi, what a practical post!
I have a question, I work in a sentiment analysis project with word2vec as an embedding model with keras. my problem is when I want to predict a new sentence as an input I face this error:

ValueError: Error when checking input: expected conv1d_1_input to have shape (15, 512) but got array with shape (3, 512)

consider that I want to enter a simple sentence like:"I'm really sad" with the length 3- and my input shape has the length of 15- I don't know how can I reshape it or doing what to get rid of this error.

and this is the related part of my code:

model = Sequential()
model.add(Conv1D(32, kernel_size=3, activation='elu', padding='same', input_shape=(15, 512)))
model.add(Conv1D(32, kernel_size=3, activation='elu', padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='elu', padding='same'))
model.add(Conv1D(32, kernel_size=3, activation='elu', padding='same'))
model.add(Dropout(0.25))

model.add(Conv1D(32, kernel_size=2, activation='elu', padding='same'))
model.add(Conv1D(32, kernel_size=2, activation='elu', padding='same'))
model.add(Conv1D(32, kernel_size=2, activation='elu', padding='same'))

model.add(Conv1D(32, kernel_size=2, activation='elu', padding='same'))
model.add(Dropout(0.25))

model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(2, activation='softmax'))

would you please help me to solve this problem?

Jason Brownlee July 21, 2018 at 6:27 am # REPLY

You must prepare the new sentence in exactly the same way as the training data, including length and integer encoding.

zbk July 20, 2018 at 6:09 pm # REPLY

At least would you mind sharing some suitable source for me to solve this problem please?
I hope you answer my question as what you done all the time. Thanks

Jason Brownlee July 21, 2018 at 6:32 am # REPLY

I'm eager to help and answer specific questions, but I don't have the capacity to write code for you sorry.

Alalehrz July 28, 2018 at 5:13 am # REPLY

Hi Jason,

I have two questions. First, for new instances if the length is greater than this model input shall we truncate the sentence?
Also, since the embedding input is just for the seen training set words, what happens to the predictions-to-word process? I assume it just returns some words similar to the training words not considering all the dictionary words. Of course I am talking about a language model with Glove.

Jason Brownlee July 28, 2018 at 6:40 am # REPLY

Yes, truncate.

Unseen words are mapped to nothing or 0.

The training dataset MUST be representative of the problem you are solving.

az July 31, 2018 at 3:30 am # REPLY

From what i understood from this comment,it is about prediction on test data. Lets assume that there are 50 words in vocabulary which means sentences will have unique integers uptil 50. Now since test data must be tokenized with same instance of tokenizer and if it has some new words, it would have integers with 51 ,52 oand so on..In this case,would the model automatically use 0 for word embeddings or can it raise out of bound type exception?Thanks

Jason Brownlee July 31, 2018 at 6:11 am # REPLY

You would encode unseen words as 0.

Jaskaran July 29, 2018 at 11:21 pm # REPLY

can this be described as transfer learning?

Jason Brownlee July 30, 2018 at 5:50 am # REPLY

Perhaps.

eden July 31, 2018 at 1:07 am # REPLY

Hi,
i have trained and tested my own network. During my work,when i integerized the sentences and created a corresponding word embedding matrix ,it included embeddings for train,validation and test data as well.
Now if i want to reload my model to test for some other similar data, i am

confused that how the words from this new data would relate to embedding matrix?
You should have embeddings for test data as well right? or when you create embedding matrix you exclude test data?Thanks

Jason Brownlee July 31, 2018 at 6:09 am # REPLY

The embedding is created from the training dataset.

It should be sufficiently rich/representative enough to cover all data you expect to in the future.

New data must have the same integer encoding as the training data prior to being mapped onto the embedding when making a prediction.

Does that help?

eden July 31, 2018 at 8:04 am # REPLY

yes,i understand that i should be using the same tokenizer object for encoding both train and test data, but i am not sure how the embedding layer would behave for the word or index which isnt part of embedding matrix. Obviously test data would have similar words but there must be some words that are bound to be new. Would you say this approach is right to include test data too while creating embedding matrix for model? If you want to predict using some pre trained model,how can i deal with this issue? A small example can be really helpful. Thanks alot for all the help and time!

Jason Brownlee July 31, 2018 at 2:55 pm # REPLY

It won’t. The encoding will set unknown words to 0.

It really depends on the goal of what you want to evaluate.

Jaskaran August 1, 2018 at 5:04 pm # REPLY

i want to train my model to predict the target word given to a 5 word sequence . how can i represent my target word ?

Jason Brownlee August 2, 2018 at 5:56 am # REPLY

Probably using a one hot encoding.

Bharath August 3, 2018 at 12:48 pm # REPLY

Hello Jason,

This is regarding the output shape of the first embedding layer : (None,4,8).
Am I correct in understanding that the 4 represents the input size which is 4 words and the 8 is the number of features it has generated using those words?

Jason Brownlee August 3, 2018 at 2:23 pm # REPLY

I believe so.

Sreenivasa August 10, 2018 at 7:33 pm # REPLY

Hi Jason,
Thanks for sharing your knowledge.
My task is to classify set of documents into different categories.(I have a training set of 100 documents and say 10 categories).
The idea is to extract top M words (say 20) from the first few lines of each doc, convert words to word embeddings and use it as feature vectors for the neural network.

Question : Since i take top M words from the document, it may not be in the "right" order each time, meaning the there can be different words at a given position in the input layer (unlike bag of words model). Wont this approach impact the Neural network from converging?

Regards,
Srini

Jason Brownlee August 11, 2018 at 6:08 am # REPLY

The key is to assign the same integer value to each word prior to feeding the data into the embedding.

You must use the same text tokenizer for all documents.

Fatemeh August 17, 2018 at 7:15 am # REPLY

Hi Jason,
Thank you for your great explanation. I have used the pre-trained google embedding matrix in my seqtoseq project by using encoder-decoder. but in my test, I have a problem. I don't know how to make a reverse for my embedding matrix. Do you have a sample project? My solution is: when my decoder predicts a vector, I should search for that in my pre-trained embedding matrix, and then find its index and then understand its related word. Am I right?

Jason Brownlee August 17, 2018 at 7:40 am # REPLY

Why would you need to reverse it?

Vivien September 29, 2018 at 1:45 pm # REPLY

Hi Jason

Thanks for an excellent tutorial. Using your methods, I've converted text into word index and applied word embeddings.

Like Fatemeh, I'm wondering if it's possible to reverse the process, and convert embedding vectors back into text? This could be useful for applications such as text summarising.

Thank you.

Jason Brownlee September 30, 2018 at 6:00 am # REPLY

Yes, each vector has an int value known by the embedding and each int has a mapping to a word via the tokenizer.

Random vectors in the space do not, you will have to find the closest vector in the embedding using euclidean distance.

fatma August 18, 2018 at 10:09 am # REPLY

Dear Dr. Jason,

Accuracy: 89.999998 on my Laptop, result different from computer to other?

Jason Brownlee August 19, 2018 at 6:15 am # REPLY

Well done!

Results are different each run, learn more here: https://machinelearningmastery.com/randomness-in-machine-learning/

Ivan September 1, 2018 at 2:00 pm # REPLY

Hi,

So many thanks for this tutorial!

I've been trying to train a network that consists of an Embedding layer, an LSTM and a softmax as the output layer. However, it seems that the loss and accuracy get stuck at some point.

Do you have any advice?

Thanks in advance.

Jason Brownlee September 2, 2018 at 5:29 am # REPLY

Yes, I have a ton of advice right here:
http://machinelearningmastery.com/improve-deep-learning-performance/

Aramis September 28, 2018 at 10:28 am # REPLY

Thank you so much,
It helped me alot in learning how to use pre trained embbeding in neural nets

Jason Brownlee September 28, 2018 at 3:00 pm # REPLY

I'm happy to hear that.

Rafael sa October 24, 2018 at 5:28 am # REPLY

Hi Jason, thank uou for the great materiAl.

I have one doubt, want to make the embedding of a list of 1200 documents to use it as input to a classification model to predict moviebox office based on the moviescript text...
My question is... if i want to train the embedding with the vocabulary of the real dataset, how can i after classify the rest of the dataset that was not trained ? Can a use the embeddings learned on the training as input to the classification model ?

Jason Brownlee October 24, 2018 at 6:33 am # REPLY

Good question.

You must ensure that the training dataset is representative of the broader problem. Any words unseen during training will be marked zero (unknown) by the tokenizer unless you update your model.

Rafael SA November 1, 2018 at 9:06 pm # REPLY

Thank You Jason. As soon as I get the results I'll try to share it here. I'd like to thank you too about your great platform, it is being very helpful to me.

Jason Brownlee November 2, 2018 at 5:49 am # REPLY

You're welcome.

Dimitris October 26, 2018 at 2:06 am # REPLY

Nice post once again! It seems that in each batch all embedding are updated which I think it should not happen. You got any idea how to update only the one that are passed each time? That is for computational reasons or others problem definitions related reasons.

Jason Brownlee October 26, 2018 at 5:38 am # REPLY

I'm not sure what you mean exactly, can you elaborate?

Mahdi November 18, 2018 at 1:57 am # REPLY

Hello Jason, i would like to think you for this post, it's really interresting and understandable.

I've reused the script but instead of using "docs" and "labels" lists, i used the IMDB movie reviews dataset. The problem is that i can't reach more than 50% accuracy and the loss is stable in all epochs to value 0.6932.

What do you think about that ?

Jason Brownlee November 18, 2018 at 6:43 am # REPLY

I have some suggestions here:
http://machinelearningmastery.com/improve-deep-learning-performance/

Mahdi November 18, 2018 at 9:37 am # REPLY

Okay I'll check it out, thank you Jason

Mehran November 18, 2018 at 12:55 pm # REPLY

Thanks for the article. Could you also provide an example of how to train a model with only one Embedding layer? I'm trying to do the same with Keras but the problem is that the fit method asks for labels which I don't have. I mean I only have a bunch of text files that I'm trying to come up with the mapping for.

Jason Brownlee November 19, 2018 at 6:42 am # REPLY

Models typically only have one embedding layer. What do you mean exactly?

Vishal November 22, 2018 at 4:07 pm # REPLY

Hello,

Thank you for the excellent explanation!

I have a few questions related to unknown words.

Some pretrained word embeddings like the GoogleNews embeddings have an embedding vector for a token called 'UNKNOWN' as well.

1. Can I use this vector to represent words that are not present in the training set instead of the vector of all zeros? If so, how should I go about loading this vector into the Keras Embedding layer? Should it be loaded at the 0th index in the embedding matrix?

2. Also, can I use the Tokenizer API to help me convert all unknown words (words not in the training set) to 'UNKNOWN'?

Thank you.

Jason Brownlee November 23, 2018 at 7:43 am # REPLY

Yes, find the integer value for the unknown word and use that to assign to all words not in the vocab.

Saurabh November 27, 2018 at 4:46 pm # REPLY

Hi,
If word embedding doesn't contain a word we input to a model , How to address this issue?
1) Is it possible to load additional words (besides those in our vocabulary) in embedding matrix.
Or may be any other elegant way you would like to suggest?

Jason Brownlee November 28, 2018 at 7:38 am # REPLY

It is marked as "unknown".

mohammad November 28, 2018 at 11:15 pm # REPLY

Hi .thanks a lot for your post . i'm new in python and deep learning !

i have 240,000 tweet train set "50 % male and 50% female" class . and 120,000 tweet test set " 50 % male and 50% female". i want use lstm in python bud i have following error at " fit " method :

ValueError: Error when checking input: expected lstm_16_input to have 3 dimensions, but got array with shape (120000, 400)

can you help me?

Jason Brownlee November 29, 2018 at 7:40 am # REPLY

It looks like a mismatch between your data and the model, you can change the data or change the model.

Mohamed December 5, 2018 at 3:46 am # REPLY

Hi Jason, Thanks for this article.

I am getting this error

TypeError: 'OneHotEncoder' object is not callable

How oto overcome?

Thanks

Jason Brownlee December 5, 2018 at 6:21 am # REPLY

I have some suggestions here:
https://machinelearningmastery.com/faq/single-faq/why-does-the-code-in-the-tutorial-not-work-for-me

Rushi December 6, 2018 at 5:40 pm # REPLY

Hi , i have 2 models with this embedding layers , how do i merge those model ?

Thanks

Jason Brownlee December 7, 2018 at 5:17 am # REPLY

What do you mean exactly? An ensemble model?

Utkarsh Rai December 7, 2018 at 12:29 am # REPLY

hi Jason, greate tutorial, i am very new to all this. I have a query, u r using glove for the embedding layer but during fitting u are directly using padded_docs. The vectors in padded_docs have no co-relation to glove. I am sure that i am missing something plz enlighten.

Jason Brownlee December 7, 2018 at 5:23 am # REPLY

The padding just adds '0' to ensure the sequences are the same length. It does not effect the encoding.

Eduardo Andrade December 7, 2018 at 3:48 pm # REPLY

Hi, Jason. Considering the "3. Example of Learning an Embedding", I'm adding "model.add(LSTM(32, return_sequences=True))" after the embedding layer and I would like to understand what happens. The number of parameters returned for this LSTM layer is "5248" and I don't know how to calculate it. Thank you.

Jason Brownlee December 8, 2018 at 6:58 am # REPLY

Each unit in the LSTM will take the entire embedding as input, therefore must have one weight for each dimension in the embedding.

Vic` December 18, 2018 at 10:12 am # REPLY

Hi Jason,

Do you have any example showing how we can use a bi-directional LSTM on text (i.e., using word embeddings)?

Jason Brownlee December 18, 2018 at 2:34 pm # REPLY

It is the same as using the LSTM except, the LSTM is wrapped in a Bidirectional Layer wrapper.

Here's an example of how to use the Bidirectional wrapper:
https://machinelearningmastery.com/develop-bidirectional-lstm-sequence-classification-python-keras/

Matt December 24, 2018 at 9:57 pm # REPLY

I am interested in using the predict function to predict new docs. For instance, 'Struck out!' My understanding is that if one or more words in the doc you want to predict weren't involved in training, then the model can't predict it. Is the solution to simply train on enough docs to make sure the vocabulary is extensive enough to make new predictions in this way?

Jason Brownlee December 25, 2018 at 7:21 am # REPLY

Yes, or mark new words as "unknown" a predict time.

Ajay December 26, 2018 at 11:53 pm # REPLY

Hello Jason, Is there any reason that the output of the Embedding layer is a 4×8 matrix?

Jason Brownlee December 27, 2018 at 5:43 am # REPLY

No, it is just an example to demonstrate the concept.

Zeyu December 30, 2018 at 7:09 am # REPLY

Hi, Jason. Thanks a lot for this excellent tutorial. I have a quick question about the Keras Embedding layer.

vocab_size = len(t.word_index) + 1

t.word_index starts from 1 and ends with 15. Therefore, there are totally 15 words in the vocabulary. Then why do we need to add 1 here please?

Thanks a lot for the help!

Jason Brownlee December 31, 2018 at 6:02 am # REPLY

The words are 1-offset and index 0 is left for "unknown" words.

Anna January 3, 2019 at 4:13 pm # REPLY

Hi Jason,

If I have three columns of (string type) multivariate data, one column is categorical, the other two columns are not. Is it ok if I integer encode them using LabelEncoding(), and then scale the encoded data using feature scaling method like MinMax, StandardScaler etc. before feed into anomaly detection model? Even though the ROC shows an impressive result. But is it valid to pre-process text data like that?

Thank you.

Jason Brownlee January 4, 2019 at 6:26 am # REPLY

Perhaps try it and compare performance.

Anna January 4, 2019 at 2:49 pm # REPLY

I have tried it and it shows nearly 100% ROC. What I mean is that, is it accurate to pre-process the text data like that? Because when I checked your post regarding pre-processing text data, there is no feature scaling (MinMax, StandardScaler etc) on text data after encode them to integer. I'm afraid if the way I pre-process data is not accurate.

Jason Brownlee January 5, 2019 at 6:47 am # REPLY

Generally text is encoded as integers then either one hot encoded or mapped to a word embedding. Other data preparation (e.g. scaling) is not required.

Kafeel Basha January 6, 2019 at 7:46 pm # REPLY

Hello

I was trying to do multi class classification of text data using Keras in R and Python.

In Python I was able to get predicted labels using inverse_transform() method from encoded class values. But when I try to do the same in R using CatEncoders library, getting some of the labels as NAs. Any reason for that.

Jason Brownlee January 7, 2019 at 6:28 am # REPLY

No need to transform the prediction, the model can make a class or probability prediction directly.

Li Xiaohong January 7, 2019 at 12:49 am # REPLY

Hi Jason,

Thanks for your sharing! I have a question on word embedding. Correct me if I am wrong: noticed the word embedding created here only contains words in the training/test set. I would think a word embedding including all vocab in GloVE file will be better? For example, if in production, we encounter a new word than in training/test set, but it is part of the GloVE vocab, in this case, we can capture the meaning of the production words although we don't see it in training/test set. I think this will benefit sentiment classification problems with smaller training set?

Thanks!
Regards
Xiaohong

Jason Brownlee January 7, 2019 at 6:36 am # REPLY

Generally, you carefully choose your vocab. If you want to maintain a larger vocab than is required of the project "just in case", go for it.

Leave a Reply

Name (required)

Email (will not be published) (required)

Website

SUBMIT COMMENT

Welcome to Machine Learning Mastery!

Hi, I'm Jason Brownlee, PhD
I write tutorials to help developers (*like you*) get results with machine learning.

Read More

POPULAR

How to Develop a Neural Machine Translation System from Scratch
JANUARY 10, 2018

So, You are Working on a Machine Learning Problem...
APRIL 4, 2018

How to Develop an N-gram Multichannel Convolutional Neural Network for Sentiment Analysis
JANUARY 12, 2018

How to Make Predictions with Keras
APRIL 9, 2018

11 Classical Time Series Forecasting Methods in Python (Cheat Sheet)
AUGUST 6, 2018

How to Develop LSTM Models for Multi-Step Time Series Forecasting of Household Power Consumption
OCTOBER 10, 2018

You're Doing it Wrong. Why Machine Learning Does Not Have to Be So Hard
MARCH 28, 2018

You might also like...

- How to Install Python for Machine Learning
- Your First Machine Learning Project in Python
- Your First Neural Network in Python
- Your First Classifier in Weka
- Your First Time Series Forecasting Project