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**

Last Updated on February 2, 2021

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.

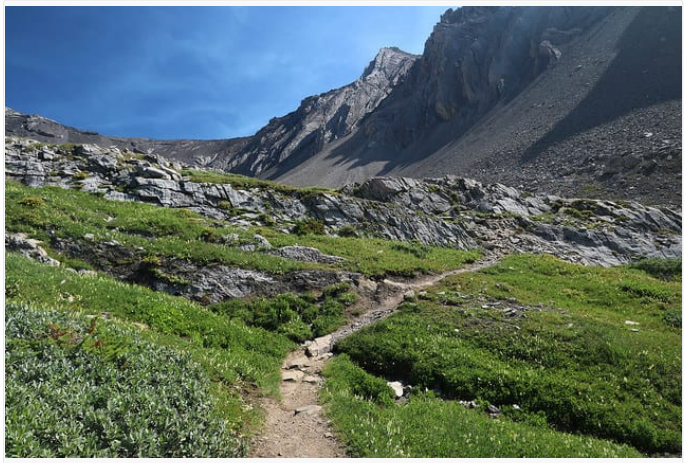
After completing this tutorial, you will know:

- 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.

Kick-start your project with my new book Deep Learning for Natural Language Processing, including *step-by-step tutorials* and the *Python source code* files for all examples.

Let's get started.

- **Updated Feb/2018**: Fixed a bug due to a change in the underlying APIs.
- **Updated Oct/2019**: Updated for Keras 2.3 and TensorFlow 2.0.



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=['accuracy'])
# 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=['accuracy'])
# 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], [49,
```

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
_________________________________________________________________
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

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 0.0
```

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, trainable=F
```

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, trainable=
model.add(e)
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# 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))
```

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

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, 14]]

[[ 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.

More On This Topic

How to use the UpSampling2D and Conv2DTranspose…

How to Develop Word-Based Neural Language Models in…

How to Develop Word Embeddings in Python with Gensim

How to Develop a Word-Level Neural Language Model…

How Do Convolutional Layers Work in Deep Learning…

What Are Word Embeddings for Text?

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 ›

632 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.

REPLY

Ajit March 15, 2020 at 12:40 am #

Thanks man, It was really helpful.

REPLY

Jason Brownlee March 15, 2020 at 6:14 am #

You're welcome.

REPLY

sherry July 22, 2019 at 7:22 pm #

after embedding,have to have a "Flatten()" layer? In my project, I used a dense layer directly after embedding. is it ok?

REPLY

Jason Brownlee July 23, 2019 at 7:59 am #

Try it and see.

REPLY

Peter Nduru October 8, 2019 at 6:06 am #

I appreciate how well updated you keep these tutorials. the first thing I always look at, when I start reading is the update date. thank you very much.

REPLY

Jason Brownlee October 8, 2019 at 8:10 am #

You're welcome.

I require all of the code to work and keep working!

REPLY

Martin October 11, 2019 at 4:09 am #

Hi, Jason:

when one_hot encoding is used, why is padding necessary? Doesn't one_hot encoding already create an input of equal length?

REPLY

Jason Brownlee October 11, 2019 at 6:25 am #

The one hot encoding is for one variable at one time step, e.g. features.

Padding is needed to make all sequences have the same number of time steps.

See this:
https://machinelearningmastery.com/faq/single-faq/what-is-the-difference-between-samples-timesteps-and-features-for-lstm-input

REPLY

shiv October 5, 2017 at 10:07 am #

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%.

REPLY

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

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

REPLY

trulia October 6, 2017 at 12:47 pm #

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

REPLY

Sandy October 6, 2017 at 2:44 pm #

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?

REPLY

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

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?

REPLY

Sandy October 7, 2017 at 5:37 pm #

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.

REPLY

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

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.

REPLY

Chiedu October 7, 2017 at 5:36 pm #

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

REPLY

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

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

REPLY

mahna April 28, 2018 at 2:31 am #

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.

REPLY

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

No, it is an attribute on the model.

REPLY

Abbey October 8, 2017 at 2:19 am #

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

REPLY

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

Sorry, what was your question?

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

REPLY

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

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

REPLY

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

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

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

REPLY

Rafael Sá June 17, 2019 at 2:47 am #

Hi Jason,

I have a set of documents(1200 text of movie Scripts) and i want to use pretrained embeddings. But i want to update the vocabulary and train again adding the words of my corpus. Is that possible ?

REPLY

Jason Brownlee June 17, 2019 at 8:24 am #

Sure.

Load the pre-trained vectors. Add new random vectors for the new words in the vocab and train the whole lot together.

REPLY

Vinu October 9, 2017 at 5:54 pm #

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

REPLY

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

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

REPLY

Hao October 12, 2017 at 5:49 pm #

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!

REPLY

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

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

REPLY

ammara May 10, 2018 at 2:47 am #

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

REPLY

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

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

REPLY

Michael October 13, 2017 at 10:22 am #

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.

REPLY

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

Thanks, I'm glad it helped.

REPLY

Azim October 17, 2017 at 5:47 pm #

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?

REPLY

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

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/

REPLY

Azim October 21, 2017 at 9:34 pm #

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

REPLY

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

I don't know Azim.

REPLY

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

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..

REPLY

Willie October 21, 2017 at 5:55 pm #

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.

REPLY

cam October 28, 2017 at 5:34 am #

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.

REPLY

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

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

REPLY

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

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.

REPLY

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

Nice!

REPLY

Ravil November 3, 2017 at 5:43 am #

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.

REPLY

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

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

REPLY

Ken April 2, 2018 at 8:37 am #

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?"

REPLY

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

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.

REPLY

chengcheng November 9, 2017 at 2:56 am #

the chinese word how to vector sequence

REPLY

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

Sorry?

REPLY

lstmbot December 16, 2017 at 10:30 pm #

me bot trying interact comments born with lstm

REPLY

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

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.

REPLY

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

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

What was the problem exactly?

REPLY

Tony July 11, 2018 at 8:00 am #

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?

REPLY

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

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.

REPLY

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

Hello Jason!

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

REPLY

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

Yes.

REPLY

Alex November 21, 2017 at 11:15 pm #

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.

REPLY

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

Thanks for the tip Alex.

REPLY

Liliana November 26, 2017 at 12:00 pm #

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!

REPLY

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

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/

REPLY

Wassim November 28, 2017 at 1:06 am #

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

REPLY

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

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

REPLY

Wassim November 28, 2017 at 10:50 pm #

Thank you Jason

REPLY

ashish December 2, 2017 at 8:29 pm #

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

REPLY

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

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

REPLY

Stuart December 7, 2017 at 12:11 am #

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

REPLY

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

I'm glad the post helped Stuart!

REPLY

Stuart December 11, 2017 at 3:30 pm #

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"?

REPLY

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

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

REPLY

Stuart December 11, 2017 at 9:34 pm #

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

REPLY

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

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

REPLY

Stuart December 12, 2017 at 7:53 am #

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

REPLY

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

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

REPLY

Nadav December 25, 2017 at 8:28 am #

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).

REPLY

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

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.

REPLY

Hitkul January 9, 2018 at 9:05 pm #

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?

REPLY

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

It depends on what you’re trying to model.

REPLY

Paul January 12, 2018 at 6:40 pm #

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

REPLY

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

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.

Elizabeth October 27, 2019 at 9:49 am #

REPLY

get_weights() for what exactly? does it need a loop?

Jason Brownlee October 28, 2019 at 6:01 am #

REPLY

get_weights is a function that will return the weights for a model or layers, depending on what you call it on exactly:
https://keras.io/layers/about-keras-layers/

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.

REPLY

Vladimir January 22, 2018 at 11:54 pm #

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?

REPLY

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

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.

REPLY

joseph January 31, 2018 at 9:20 pm #

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

REPLY

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

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

REPLY

Anna February 4, 2018 at 9:40 pm #

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

REPLY

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

The dimensionality is fixed and specified.

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

REPLY

Anna February 5, 2018 at 2:17 pm #

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

REPLY

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

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.

REPLY

Miroslav February 5, 2018 at 7:09 am #

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

REPLY

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

I don't know, sorry.

REPLY

Gabriel February 6, 2018 at 4:08 am #

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!

REPLY

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

Great question!

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

REPLY

Gabriel February 6, 2018 at 4:22 pm #