Machine Learning Mastery
Making Developers Awesome at Machine Learning

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 October 3, 2019

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.

Discover how to develop deep learning models for text classification, translation, photo captioning and more in my new book, with 30 step-by-step tutorials and full source code.

Let's get started.

- **Update 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

How to Use Word Embedding Lay
Photo by thisguy, so

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 wor
learning model. This can be a slower approach, but ta

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 t .
- It can be used as part of a deep learning model w del itself.
- It can be used to load a pre-trained word embedd

The Embedding layer is defined as the first hidden lay

It must specify 3 arguments:

- **input_dim**: This is the size of the vocabulary in th encoded to values between 0-10, then the size of
- **output_dim**: This is the size of the vector space i ze 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 Emb

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 me
sequences of integers. We could experiment with othe
like counts or TF-IDF.

Keras provides the one_hot() function that creates a h
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_
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, 46, 16, 34
```

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 i … t of the Embedding layer is a 4×8 matrix and this is squas…

```
_________________________________________________
Layer (type)                 Output Shape
=================================================
embedding_1 (Embedding)      (None, 4, 8)
_________________________________________________
flatten_1 (Flatten)          (None, 32)
_________________________________________________
dense_1 (Dense)              (None, 1)
=================================================
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.

- 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) ne.
For example, below are the first line of the embedding

```
the -0.038194 -0.24487 0.72812 -0.39961 0.0831 0.28
```

As in the previous section, the first step is to define th e sequences to be the same length.

In this case, we need to be able to map words to integ

Keras provides a Tokenizer class that can be fit on the 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 embeddi 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

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

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

The key difference is that the embedding layer can be 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=False)
```

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='float3
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddi
# create a weight matrix for words in trainin
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(w
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vecto
# define model
model = Sequential()
e = Embedding(vocab_size, 100, weights=[embed
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))
```


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
=====================================================
```

```
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 y

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

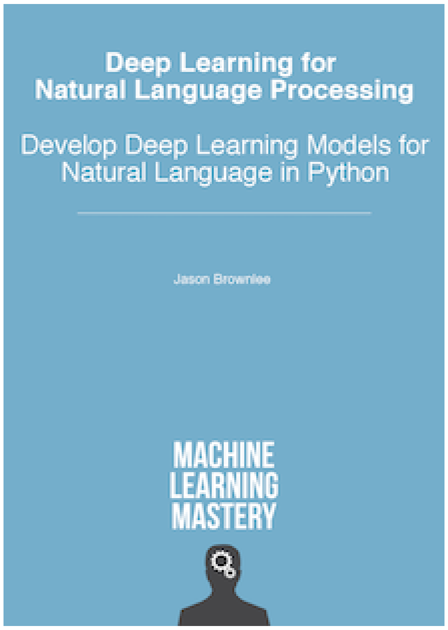
Summary

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.



About Jason Brownlee

Jason Brownlee, PhD is a machine learnin ... th
modern machine learning methods via han
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 ›

429 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.

sherry July 22, 2019 at 7:22 pm #

REPLY

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

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

REPLY

Try it and see.

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

I appreciate how well updated you keep th
start reading is the update date. thank you very mu

Jason Brownlee October 8, 2019 at 8:10

You're welcome.

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

Martin October 11, 2019 at 4:09 am #

REPLY

Hi, Jason:

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

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

REPLY

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

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 #

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

Sandy October 6, 2017 at 2:44 pm #

Thank you Jason. I always find things easier v
I have a question about the vector of each word after t
"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 trainin

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 #

Hi Jason,
any ideas on how to “filter the embedding for the uniqu
tutorial?

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

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

Sorry, what was your question?

If your question was if this is a good approach, my

Abiodun Modupe October 9, 2017 at 7:

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?

Rafael Sá June 17, 2019 at 2:47 am # REPLY

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 ?

Jason Brownlee June 17, 2019 at 8