Sentiment Classification

Generate Word Embeddings and retrieve outputs of each layer with Keras based o

Word embeddings are a type of word representation that allows words with similar meaning to have a

It is a distributed representation for text that is perhaps one of the key breakthroughs for the impressi methods on challenging natural language processing problems.

We willl use the imdb dataset to learn word embeddings as we train our dataset. This dataset contain labeled with sentiment (positive or negative).

Dataset

`from keras.datasets import imdb`

Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have review is encoded as a sequence of word indexes (integers). For convenience, the words are indexed meaning the for that has index 1 is the most frequent word. Use the first 20 words from each review t vocab size of 10,000.

As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown w

Aim

1. Import test and train data
2. Import the labels (train and test)
3. Get the word index and then Create key value pair for word and word_id. (12.5 points)
4. Build a Sequential Model using Keras for Sentiment Classification task. (10 points)
5. Report the Accuracy of the model. (5 points)
6. Retrive the output of each layer in keras for a given single test sample from the trained model you built. (2.5 p

Usage:

```
from keras.datasets import imdb

vocab_size = 20 #vocab size

import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
```

```
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old
```

```
Using TensorFlow backend.
The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.
We recommend you upgrade now or ensure your notebook will continue to use TensorFlow 1.x via the %tens
Downloading data from https://s3.amazonaws.com/text-datasets/imdb.npz
17465344/17464789 [==============================] - 1s 0us/step
17473536/17464789 [==============================] - 1s 0us/step
```

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
```

```
WARNING: Logging before flag parsing goes to stderr.
W0307 16:33:44.615564 140466024060800 deprecation.py:323] From /usr/local/lib/python2.7/
Instructions for updating:
non-resource variables are not supported in the long term
```

```
from keras.preprocessing.sequence import pad_sequences
vocab_size = 10000 #vocab size
maxlen = 300  #number of word used from each review


#load dataset as a list of ints
import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old

#make all sequences of the same length
train_data = pad_sequences(train_data, maxlen=maxlen)
test_data =  pad_sequences(test_data, maxlen=maxlen)


#Check Data Range
print('Min', np.min(np_load_old))
print('Max', np.max(np_load_old))
```

```
('Min', <function load at 0x7fc0be973bd0>)
('Max', <function load at 0x7fc0be973bd0>)
```

3. Get the word index and then Create a key-value pair for word and word_id (12.5 points)

```
import keras
NUM_WORDS=1000 # only use top 1000 words
INDEX_FROM=3   # word index offs

import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old


word_to_id = keras.datasets.imdb.get_word_index()
word_to_id = {k:(v+INDEX_FROM) for k,v in word_to_id.items()}
word_to_id["<PAD>"] = 0
word_to_id["<START>"] = 1
word_to_id["<UNK>"] = 2
word_to_id["<UNUSED>"] = 3

id_to_word = {value:key for key,value in word_to_id.items()}
print(' '.join(id_to_word[id] for id in train_data[0] ))
```

```
<START> this film was just brilliant casting location scenery story direction everyone's
```

Build Keras Embedding Layer Model

We can think of the Embedding layer as a dicionary that maps a index assigned to a word to a word v
can be used in a few ways:

- The embedding layer can be used at the start of a larger deep learning model.
- Also we could load pre-train word embeddings into the embedding layer when we create our mo
- Use the embedding layer to train our own word2vec models.

The keras embedding layer doesn't require us to onehot encode our words, instead we have to give ea
an id. For the imdb dataset we've loaded this has already been done, but if this wasn't the case we cou

```
import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=10000)

# restore np.load for future normal usage
np.load = np_load_old
```

```
# summarize size
print("Training data: ")
print(train_data.shape)
print(test_data.shape)
```

```
Training data:
(25000,)
(25000,)
```

```
# Summarize number of classes
print("Classes: ")
print(numpy.unique(test_data))
```

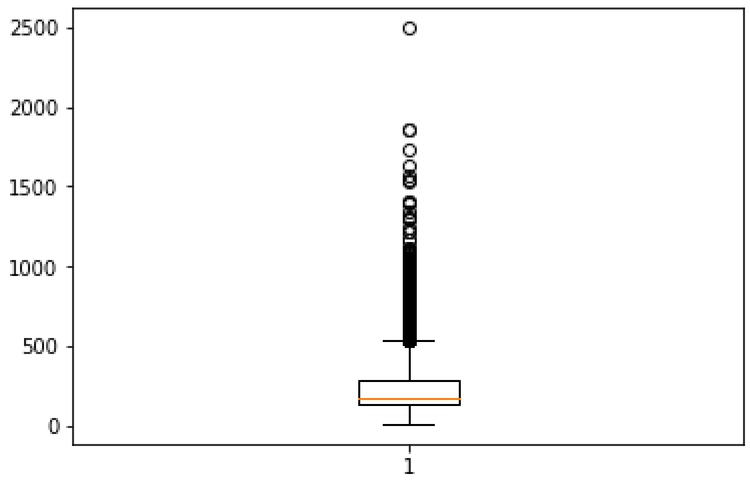
```
Classes:
[list([1, 2, 2, 2, 2, 4, 2, 393, 7, 6, 2, 1830, 39, 998, 2445, 1364, 656, 94, 58, 143, 4
 list([1, 2, 2, 2, 4, 31, 125, 1293, 29, 115, 93, 160, 22, 21, 1075, 6, 530, 3213, 7, 47
 list([1, 2, 2, 2, 7, 4, 3352, 5, 2226, 51, 2320, 5, 4063, 2473, 15, 71, 7681, 725, 178,
 ...
 list([1, 9868, 7140, 6178, 10, 10, 54, 1441, 2371, 46, 7, 4, 673, 8, 3538, 1720, 29, 30
 list([1, 9875, 27, 205, 667, 2, 18, 4, 268, 2133, 2979, 167, 6946, 9509, 3357, 14, 2996
 list([1, 9927, 2, 9, 6, 686, 19, 6, 117, 1056, 37, 1388, 4007, 5, 27, 1410, 33, 4, 1450
```

```
# Summarize number of words
print("Number of words: ")
print(len(numpy.unique(numpy.hstack(train_data))))
```

```
Number of words:
9998
```

```
# Summarize review length
print("Review length: ")
result = [len(x) for x in train_data]
print("Mean %.2f words (%f)" % (numpy.mean(result), numpy.std(result)))
# plot review length
pyplot.boxplot(result)
pyplot.show()
```

```
Review length:
Mean 238.71 words (176.493674)
```



```
import numpy as np
# save np.load
np_load_old = np.load

# modify the default parameters of np.load
np.load = lambda *a,**k: np_load_old(*a, allow_pickle=True, **k)

# call load_data with allow_pickle implicitly set to true
(train_data, train_labels), (test_data, test_labels) = imdb.load_data(num_words=5000)

# restore np.load for future normal usage
np.load = np_load_old
```

```
train_data = pad_sequences(train_data, maxlen=500)
test_data = pad_sequences(test_data, maxlen=500)
```

```
from keras.layers import Embedding
```

```
Embedding(5000, 32, input_length=500)
```

```
W0307 17:49:09.632725 140466024060800 module_wrapper.py:139] From /usr/local/lib/python2

<keras.layers.embeddings.Embedding at 0x7fc062434e50>
```

```
# MLP for the IMDB problem
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
```

```
max_words = 500
train_data = pad_sequences(train_data, maxlen=max_words)
test_data = pad_sequences(test_data, maxlen=max_words)
```

```
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

```
W0307 18:06:27.473428 140466024060800 module_wrapper.py:139] From /usr/local/lib/python2

W0307 18:06:27.487052 140466024060800 module_wrapper.py:139] From /usr/local/lib/python2

W0307 18:06:27.525612 140466024060800 module_wrapper.py:139] From /usr/local/lib/python2

W0307 18:06:27.541925 140466024060800 module_wrapper.py:139] From /usr/local/lib/python2

W0307 18:06:27.546897 140466024060800 deprecation.py:323] From /usr/local/lib/python2.7/
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
flatten_1 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               4000250
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
=================================================================
Total params: 4,160,501
Trainable params: 4,160,501
Non-trainable params: 0
_________________________________________________________________
None
```

```
np.load.__defaults__=(None, True, True, 'ASCII')
```

Double-click (or enter) to edit

```
# MLP for the IMDB problem
from keras.datasets import imdb
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
np.load = lambda *a,**k: old(*a,allow_pickle=True)

max_words = 500
train_data = sequence.pad_sequences(train_data, maxlen=max_words)
test_data = sequence.pad_sequences(test_data, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()
# Fit the model
model.fit(train_data, validation_data=(test_data), epochs=2, batch_size=128, verbose=2)
# Final evaluation of the model
scores = model.evaluate(test_data, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 500, 32)           160000
_________________________________________________________________
flatten_4 (Flatten)          (None, 16000)             0
_________________________________________________________________
dense_7 (Dense)              (None, 250)               4000250
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 251
=================================================================
Total params: 4,160,501
Trainable params: 4,160,501
Non-trainable params: 0
_________________________________________________________________

ValueErrorTraceback (most recent call last)
<ipython-input-94-0874b8c7279c> in <module>()
     21 model.summary()
     22 # Fit the model
---> 23 model.fit(train_data, validation_data=(test_data), epochs=2, batch_size=128, ver
     24 # Final evaluation of the model
     25 scores = model.evaluate(test_data, verbose=0)

/usr/local/lib/python2.7/dist-packages/keras/engine/training.pyc in fit(self, x, y, batc
    953         # Prepare validation data.
    954         do_validation = False
--> 955         if validation_data:
    956             do_validation = True
    957             if len(validation_data) == 2:

ValueError: The truth value of an array with more than one element is ambiguous. Use a.a
```

SEARCH STACK OVERFLOW

```
## Retrive the output of each layer in keras for a given single test sample from the trained
```

```
import keras.backend as K

def get_activations(model, model_inputs, print_shape_only=False, layer_name=None):
    print('----- activations -----')
    activations = []
    inp = model.input

    model_multi_inputs_cond = True
    if not isinstance(inp, list):
        # only one input! let's wrap it in a list.
        inp = [inp]
        model_multi_inputs_cond = False

    outputs = [layer.output for layer in model.layers if
               layer.name == layer_name or layer_name is None]  # all layer outputs

    funcs = [K.function(inp + [K.learning_phase()], [out]) for out in outputs]  # evaluation

    if model_multi_inputs_cond:
        list_inputs = []
        list_inputs.extend(model_inputs)
        list_inputs.append(0.)
    else:
        list_inputs = [model_inputs, 0.]

    # Learning phase. 0 = Test mode (no dropout or batch normalization)
    # layer_outputs = [func([model_inputs, 0.])[0] for func in funcs]
    layer_outputs = [func(list_inputs)[0] for func in funcs]
    for layer_activations in layer_outputs:
        activations.append(layer_activations)
        if print_shape_only:
            print(layer_activations.shape)
        else:
            print(layer_activations)
    return activations
```