DATA20001 Deep Learning - Exercise 4

Due Friday December 1, before 12:00 PM (noon)

In this fourth computer exercise we are going to work with text and methodologies you can freely choose from Keras (https://keras.io/). Due to absence of our teacher, this exercise has not been formulated as completely as the previous exercises. Because of this and the impending project work, PLEASE REFRAIN FROM SPENDING **TOO MUCH** TIME ON THIS EXERCISE. Once the project is announced, focus more or on that. We will take this into account when scoring this exercise. Use the discussion board on Moodle to discuss issues in the exercise.

Exercise 4. Classifying text (6pts)

Your task is to solve the classification problem for the dataset we present below. You should solve this exercise in similar fashion as the previous exercise (and you can copy-paste any useful tools from there to use them here, as you see fit):

- Figure out the model which is suitable for the given data, complete from the hidden layers to the representation of X and the suitable loss function.
- Validate and evaluate the model properly.
- Document briefly what you did.

The necessary imports:

In [1]:

```
%matplotlib inline

import keras
from keras.datasets import reuters
import glob
from keras.utils import np_utils
import matplotlib.pyplot as plt
import numpy as np
from keras.utils.data_utils import get_file
```

```
Using TensorFlow backend.
```

Dataset

This time our dataset is a dataset of 11,228 newswires from Reuters, labeled over 46 topics. Each wire is encoded as a sequence of word indexes (same conventions).

In [2]:

```
(x_train, y_train), (x_test, y_test) = reuters.load_data(path="reuters.npz",
                                                         num_words=None,
                                                         skip_top=0,
                                                         maxlen=None,
                                                         test_split=0.2,
                                                         seed=113,
                                                         start_char=1,
                                                         oov_char=2,
                                                         index_from=3)
```

Let's see how the data is formatted by printing the dimensionalities of the variables. Refer to the keras-documentation for further info: https://keras.io/datasets/ (https://keras.io/datasets/)

In [3]:

```
print("x_train size", x_train.shape)
print("y_train size", y_train.shape)
print("x_test size", x_test.shape)
print("y_test size", y_test.shape)
print("")
print("Number of classes:", np.unique(y_train).shape[0])
word_index = reuters.get_word_index(path="reuters_word_index.json")
print(word_index)
```

```
x_train size (8982,)
y_train size (8982,)
x_test size (2246,)
y_test size (2246,)

Number of classes: 46
{'mdbl': 10996, 'fawc': 16260, 'degussa': 12089, 'woods': 8803, 'hangin
g': 13796, 'localized': 20672, 'sation': 20673, 'chanthaburi': 20675,
'refunding': 10997, 'hermann': 8804, 'passsengers': 20676, 'stipulate':
20677, 'heublein': 8352, 'screaming': 20713, 'tcby': 16261, 'four': 18
5, 'grains': 1642, 'broiler': 20680, 'wooden': 12090, 'wednesday': 122
0, 'highveld': 13797, 'duffour': 7593, '0053': 20681, 'elections': 391
4, '270': 2563, '271': 3551, '272': 5113, '273': 3552, '274': 3400, 'ru
dman': 7975, '276': 3401, '277': 3478, '278': 3632, '279': 4309, 'dorma
ncy': 9381, 'errors': 7247, 'deferred': 3086, 'sptnd': 20683, 'cookin
g': 8805, 'stratabit': 20684, 'designing': 16262, 'metalurgicos': 2068
5, 'databank': 13798, '300er': 20686, 'shocks': 20687, 'nawg': 7972, 't
nta': 20688, 'perforations': 20689, 'affiliates': 2891, '27p': 20690,
'ching': 16263, 'china': 595, 'wagyu': 16264, 'affiliated': 3189, 'chin
```

Here we provide you with a ready embedding. This is a 100-dimensional GloVe -embedding, which works similarly as Word2Vec. You can use either this or the existing features of the dataset. Note that you have to map the above word_index to the words you can find in the embedding, string by string.

In [4]:

```
database_path = 'embedding/'

dl_file='glove.6B.zip'
dl_url='https://www.cs.helsinki.fi/u/jgpyykko/'
get_file(dl_file, dl_url+dl_file, cache_dir='./', cache_subdir=database_path, extract=True)
```

```
Downloading data from https://www.cs.helsinki.fi/u/jgpyykko/glove.6B.zip (ht
tps://www.cs.helsinki.fi/u/jgpyykko/glove.6B.zip)
133275648/134300573 [============================>.] - ETA: 0s
```

Out[4]:

```
'./embedding/glove.6B.zip'
```

In [5]:

```
embedding_filename = glob.glob(database_path + '*.txt')[0]
#embedding_data = ""
embedding_data = {}
print(embedding_filename)
with open(embedding_filename, 'r') as fp:
    for line in fp:
        #embedding_data = embedding_data + str(line)
        data = line.split(" ")
        embedding_data[data[0]] = [float(x) for x in data[1:]]
print(len(embedding_data["the"]))
```

```
embedding/glove.6B.100d.txt
100
```

Exchange key value pair of the word index

In [6]:

```
word_dict = dict((v+3,k) for k,v in word_index.items())
word_dict[1] = "start_char"
word_dict[2] = "oov_char"
print(" ".join([word_dict[word] for word in x_train[0]]))
```

```
start_char mcgrath rentcorp said as a result of its december acquisition of
space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per shar
e up from 70 cts in 1986 the company said pretax net should rise to nine to
10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to
22 mln dlrs from 12 5 mln dlrs it said cash flow per share this year should
be 2 50 to three dlrs reuter 3
```

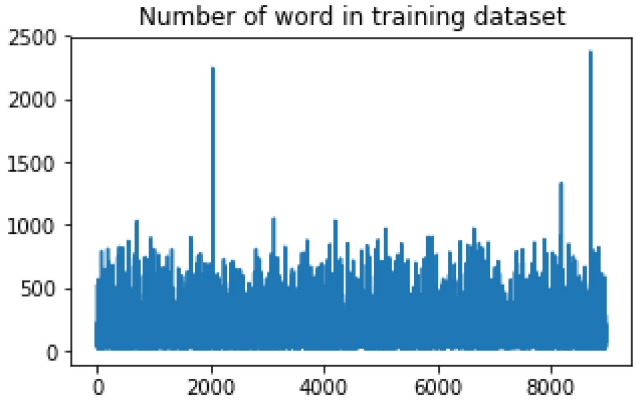
Visualize the distribution of the number of words for each wires

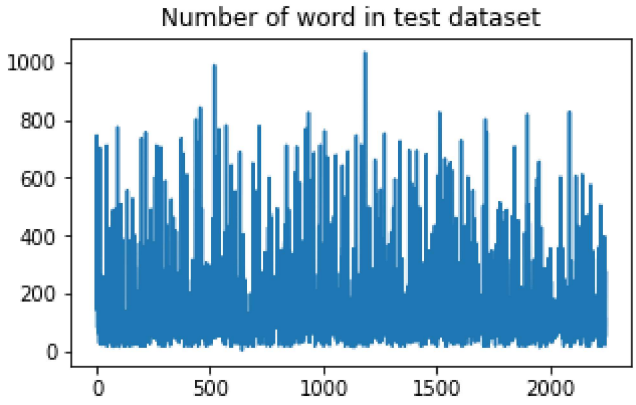
In [7]:

```
num_words_train = np.array([len(wire) for wire in x_train])
num_words_test = np.array([len(wire) for wire in x_test])

plt.figure(figsize=(5,3))
plt.plot(num_words_train)
plt.title('Number of word in training dataset')

plt.figure(figsize=(5,3))
plt.plot(num_words_test)
plt.title('Number of word in test dataset')
```

Out[7]:

```
<matplotlib.text.Text at 0x7f39a2e2b278>
```





In [8]:

```
n = num_words_train.max() #maximal number of words for each wires
k = 100 #100-dimensional embedding vector
```

CNN model

In [9]:

```
from keras.models import Sequential,Model
from keras.layers import *
from keras.optimizers import *
from keras.preprocessing import sequence

x_train_1 = sequence.pad_sequences(x_train, maxlen=n, padding="post", truncating="post",val
x_test_1 = sequence.pad_sequences(x_test, maxlen=n, padding="post", truncating="post",value

num_classes = np.unique(y_train).shape[0]
y_train_cat = np_utils.to_categorical(y_train, num_classes)
y_test_cat = np_utils.to_categorical(y_test, num_classes)

print(x_train_1.shape,x_train_1.shape)
print(y_train_cat.shape,y_test_cat.shape)
```

```
(8982, 2376) (8982, 2376)
(8982, 46) (2246, 46)
```

Construct embedding matrix

In [10]:

```
embedding_matrix = np.random.rand(len(word_dict)+1,k)

# For words which are not included in the GloVe, they are randomly initialized.
mapfunc = lambda word: word in embedding_data.keys() and\
            embedding_data[word] or np.random.rand(k)

for word,i in word_index.items():
    embedding_matrix[i] = mapfunc(word)

print(embedding_matrix.shape)
```

```
(30982, 100)
```

Non-trainable GloVe-embedding

In [15]:

```
model = Sequential()
model.add(Embedding(embedding_matrix.shape[0],embedding_matrix.shape[1],
                            weights=[embedding_matrix],
                            input_length=n, trainable=False))
model.add(Conv1D(10, 3))
model.add(MaxPooling1D(3))
model.add(Conv1D(10, 3, activation='relu'))
model.add(MaxPooling1D(3))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1024,activation="relu"))
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dense(num_classes,activation="softmax"))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 2376, 100)         3098200
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 2374, 10)          3010
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 791, 10)           0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 789, 10)           310
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 263, 10)           0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2630)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2630)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1024)              2694144
_________________________________________________________________
dense_5 (Dense)              (None, 128)               131200
_________________________________________________________________
batch_normalization_2 (Batch (None, 128)               512
_________________________________________________________________
activation_2 (Activation)    (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 46)                5934
=================================================================
Total params: 5,933,310
Trainable params: 2,834,854
Non-trainable params: 3,098,456
_________________________________________________________________
None
```

In [16]:

```
%%time

# Training
epochs = 100
history = model.fit(x_train_1,
                    y_train_cat,
                    epochs=epochs,
                    batch_size=128,
                    verbose=1)
```

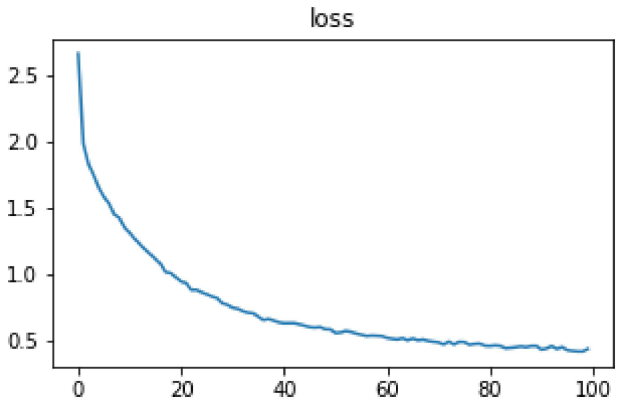
```
Epoch 1/100
8982/8982 [==============================] - 33s - loss: 2.6629 - acc:
0.3694
Epoch 2/100
8982/8982 [==============================] - 29s - loss: 1.9807 - acc:
0.4894
Epoch 3/100
8982/8982 [==============================] - 29s - loss: 1.8285 - acc:
0.5294
Epoch 4/100
8982/8982 [==============================] - 30s - loss: 1.7451 - acc:
0.5434
Epoch 5/100
8982/8982 [==============================] - 30s - loss: 1.6501 - acc:
0.5676
Epoch 6/100
8982/8982 [==============================] - 30s - loss: 1.5802 - acc:
0.5819
Epoch 7/100
```

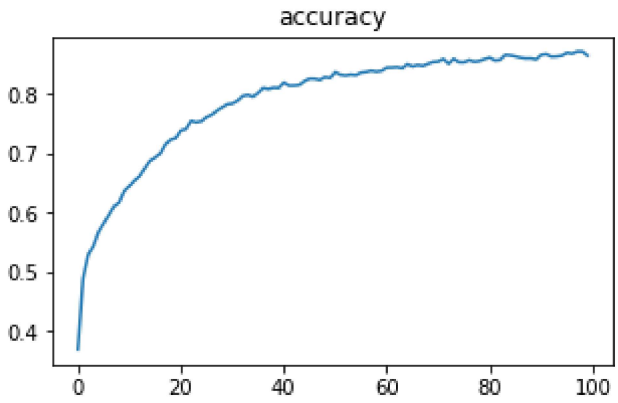
In [18]:

```
# Plot loss and accuracy in training

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'])
plt.title('accuracy')
```

Out[18]:

```
<matplotlib.text.Text at 0x7f38d56109e8>
```





In [17]:

```
# Evaluate on test set
scores = model.evaluate(x_test_1, y_test_cat, verbose=2)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
acc: 58.33%
```

Trainable GloVe Embedding

In [19]:

```
model = Sequential()
model.add(Embedding(embedding_matrix.shape[0],embedding_matrix.shape[1],
                            input_length=n,trainable=True))
model.add(Conv1D(10, 3))
model.add(MaxPooling1D(3))
model.add(Conv1D(10, 3, activation='relu'))
model.add(MaxPooling1D(3))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1024,activation="relu"))
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dense(num_classes,activation="softmax"))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 2376, 100)         3098200
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 2374, 10)          3010
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 791, 10)           0
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 789, 10)           310
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 263, 10)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2630)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2630)              0
_________________________________________________________________
dense_7 (Dense)              (None, 1024)              2694144
_________________________________________________________________
dense_8 (Dense)              (None, 128)               131200
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512
_________________________________________________________________
activation_3 (Activation)    (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 46)                5934
=================================================================
Total params: 5,933,310
Trainable params: 5,933,054
Non-trainable params: 256
_________________________________________________________________
None
```

In [20]:

```
%%time

# Training
epochs = 100
history = model.fit(x_train_1,
                    y_train_cat,
                    epochs=epochs,
                    batch_size=128,
                    verbose=1)
```

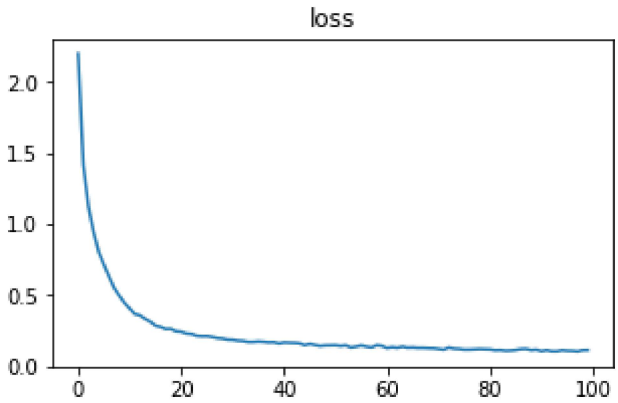
```
Epoch 1/100
8982/8982 [==============================] - 33s - loss: 2.2027 - acc:
0.4721
Epoch 2/100
8982/8982 [==============================] - 32s - loss: 1.4188 - acc:
0.6517
Epoch 3/100
8982/8982 [==============================] - 32s - loss: 1.1230 - acc:
0.7214
Epoch 4/100
8982/8982 [==============================] - 31s - loss: 0.9433 - acc:
0.7691
Epoch 5/100
8982/8982 [==============================] - 32s - loss: 0.8044 - acc:
0.7988
Epoch 6/100
8982/8982 [==============================] - 32s - loss: 0.7141 - acc:
0.8210
Epoch 7/100
```

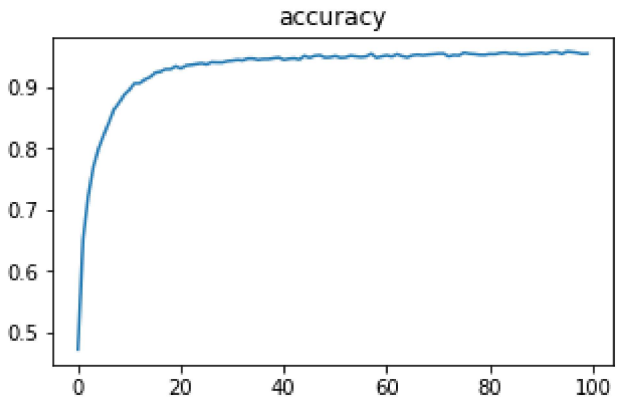
In [21]:

```
# Plot loss and accuracy in training

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'])
plt.title('accuracy')
```

Out[21]:

```
<matplotlib.text.Text at 0x7f38d063a4e0>
```





In [22]:

```
# Evaluate on test set
scores = model.evaluate(x_test_1, y_test_cat, verbose=2)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
acc: 71.77%
```

Random

In [23]:

```
model = Sequential()
model.add(Embedding(embedding_matrix.shape[0],embedding_matrix.shape[1],
                            input_length=n,trainable=True))
model.add(Conv1D(10, 3))
model.add(MaxPooling1D(3))
model.add(Conv1D(10, 3, activation='relu'))
model.add(MaxPooling1D(3))
model.add(Flatten())
model.add(Dropout(0.5))
model.add(Dense(1024,activation="relu"))
model.add(Dense(128))
model.add(BatchNormalization())
model.add(Activation("relu"))
model.add(Dense(num_classes,activation="softmax"))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 2376, 100)         3098200
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 2374, 10)          3010
_________________________________________________________________
max_pooling1d_7 (MaxPooling1 (None, 791, 10)           0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 789, 10)           310
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 263, 10)           0
_________________________________________________________________
flatten_4 (Flatten)          (None, 2630)              0
_________________________________________________________________
dropout_4 (Dropout)          (None, 2630)              0
_________________________________________________________________
dense_10 (Dense)             (None, 1024)              2694144
_________________________________________________________________
dense_11 (Dense)             (None, 128)               131200
_________________________________________________________________
batch_normalization_4 (Batch (None, 128)               512
_________________________________________________________________
activation_4 (Activation)    (None, 128)               0
_________________________________________________________________
dense_12 (Dense)             (None, 46)                5934
=================================================================
Total params: 5,933,310
Trainable params: 5,933,054
Non-trainable params: 256
_________________________________________________________________
None
```

In [24]:

```
%%time

# Training
epochs = 100
history = model.fit(x_train_1,
                    y_train_cat,
                    epochs=epochs,
                    batch_size=128,
                    verbose=1)
```

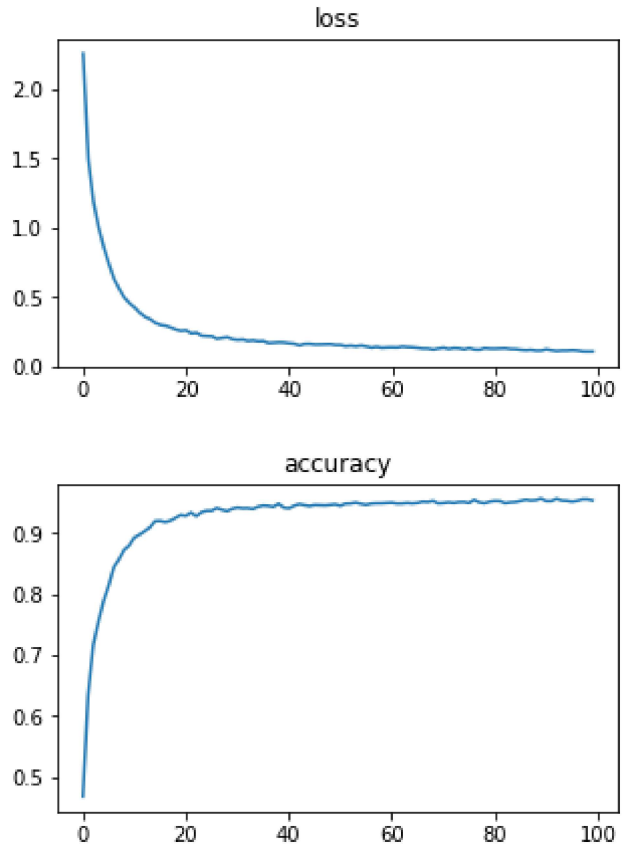
```
Epoch 1/100
8982/8982 [==============================] - 33s - loss: 2.2518 - acc:
0.4680
Epoch 2/100
8982/8982 [==============================] - 32s - loss: 1.4903 - acc:
0.6340
Epoch 3/100
8982/8982 [==============================] - 32s - loss: 1.1859 - acc:
0.7159
Epoch 4/100
8982/8982 [==============================] - 32s - loss: 1.0024 - acc:
0.7540
Epoch 5/100
8982/8982 [==============================] - 33s - loss: 0.8597 - acc:
0.7879
Epoch 6/100
8982/8982 [==============================] - 31s - loss: 0.7388 - acc:
0.8134
Epoch 7/100
```

In [25]:

```
# Plot loss and accuracy in training

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['loss'])
plt.title('loss')

plt.figure(figsize=(5,3))
plt.plot(history.epoch,history.history['acc'])
plt.title('accuracy')
```

Out[25]:

```
<matplotlib.text.Text at 0x7f3886691a20>
```



In [26]:

```
# Evaluate on test set
scores = model.evaluate(x_test_1, y_test_cat, verbose=2)
print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
acc: 71.73%
```

Document and Discussion

In Exercise 4, I refer to some online materials, documents and codes to help me form the model. However, as what is showed above, they did not perform as good as the outcomes of models in the previous exercises. I suppose there may be some procedure during data pre-processing, the model construction and representation configuration. Nevertheless, I will document what I have done so far.

Since I would like to use GloVe embedding, I first construct key-value pairs of words, which is dict object in python, and embedding vectors instead of connecting them together as what is performed in the given code. Next, I re-organize the word index so that I can use them to construct embedding matrix with GloVe vectors. Basically, I did it by reversing the key-value pair of the original word-index. I noticed that the numbers of words in each wire are not unified. In addition, according to the given code and documentation of Keras, oov_char is the word which is used to fill up short sentence so that length of sentence can be unified. I, therefore, use 2 to pad all the X. The example of one newswire eventually output something meaningful. I then match the word and GloVe vector accordingly to form the embedding matrix so that it can be used in the following CNNs.

I create 3 variant of CNN, non-trainable(static), trainable(non-static) and random. For those three, I choose to use a simpler sequential CNN model structure after embedding layer. The main reason is that a simpler model will take less time to train. Another reason is that the accuracy of the simpler model is surprisingly better than some other complex models. Another reason is that the model I use showed an increasing trend after 20 epochs for all variants. I think with more epochs the result can be improved. I did not try the model which is used in the article "Convolutional Neural Networks for Sentence Classification". But the outcomes of some reference materials seem to be very promising.

Variant	Accuracy
Non-trainable	58.3%
Trainable	71.7%
Random	71.7%

The result of these three variants, from my point of view, is quite awful. The non-trainable variant has the lowest accuracy and the accuracies of the trainable and random variant are almost the same, which is exactly opposite to the observation of article "Convolutional Neural Networks for Sentence Classification". On the other hand, none of them obtain a higher accuracy than 80%, while the obervation of the article and some other experiments show that the accuracies are all above 80%. Although the data source may be completely different from what I use now, the result is still not what I expected.

I think could be several reasons for the above outcomes:

- One reason is, of course, my model is not sophisticated enough to handle the dataset. Some of my reference materials and codes suggest using a parallel model instead of a sequential model, which may be an approach to improve the accuracy.
- Another reason may be due to my misunderstanding on appling embedding matrix. There are several ways to do that either with the non-static or static approach. For example, the code of this link (https://github.com/alexander-rakhlin/CNN-for-Sentence-Classification-in-Keras) uses the same way as I did to apply non-static embedding matrix, while, for static, it reconstructs x_train with embedding vector in the order of words and feed this as input. Another material (https://github.com/fchollet/keras/issues/1515) suggests that by setting the argument of "trainable", we can decide whether the model is static or non-static. Both of them seems to make sense. But I choose to implement the model with the latter approach.
- The third reason, I suppose, is due to the small scale of our dataset. We only have around 11 thousand data. One-fifth of all data has to be the test set, which only left us with less than 9 thousand data and this may not be a big number for training. Despite a larger dataset will surely lead to a longer pre-processing time, it may get us a much better result.
- As I mentioned above, it is possible that there exists some other configuration or data processing mistake. Though I spent much time on trying to figure out the method of applying GloVe embedding and the correct form of representation of X, I still can not be sure everything is on the right track.