Deep Learning for NLP - Project

RULES:

- Do not create any additional cell
- Fill in the blanks
- All cells should be runnable (modulo trivial compatibility bugs that we'd fix)
- 4 / 20 points will be allocated to the clarity of your code
- Efficient code will have a bonus

DELIVERABLE:

- this notebook
- the predictions of the SST test set

DO NOT INCLUDE THE DATASETS IN THE DELIVERABLE..

In [1]:
```python
import io
import os
import numpy as np
import scipy
```

In [2]:
```python
PATH_TO_DATA = "/Users/adilrhoulam/Downloads/data-2"
```

1) Monolingual (English) word embeddings

In [3]:
```python
class Word2vec():
    def __init__(self, fname, nmax=100000):
        self.load_wordvec(fname, nmax)
        self.word2id = dict.fromkeys(self.word2vec.keys())
        self.id2word = {v: k for k, v in self.word2id.items()}
        self.embeddings = np.array(self.word2vec.values())

    def load_wordvec(self, fname, nmax):
        self.word2vec = {}
        with io.open(fname, encoding='utf-8') as f:
            next(f)
            for i, line in enumerate(f):
                word, vec = line.split(' ', 1)
                self.word2vec[word] = np.fromstring(vec, sep=' ')
                if i == (nmax - 1):
                    break
        print('Loaded %s pretrained word vectors' % (len(self.word2vec)))

    def most_similar(self, w, K=5):
        # K most similar words: self.score  -  np.argsort
        keys = list(self.word2vec.keys())
        values = list(self.word2vec.values())
        scores = list(map(lambda x: self.score(x,w) if x!=w else -1,keys))
        arg = np.argsort(scores)
        return [keys[i] for i in np.flipud(arg[-K:])]

    def score(self, w1, w2):
        # cosine similarity: np.dot  -  np.linalg.norm
        vec1 = self.word2vec[w1]
        vec2 = self.word2vec[w2]
        return np.dot(vec1,vec2)/(np.linalg.norm(vec1)*np.linalg.norm(vec2))
```

In [4]:
```python
w2v = Word2vec(os.path.join(PATH_TO_DATA, 'crawl-300d-200k.vec'), nmax=100000)

# You will be evaluated on the output of the following:
for w1, w2 in zip(('cat', 'dog', 'dogs', 'paris', 'germany'), ('dog', 'pet', 'cats', 'france', 'berlin')):
    print(w1, w2, w2v.score(w1, w2))
for w1 in ['cat', 'dog', 'dogs', 'paris', 'germany']:
    print(w2v.most_similar(w1))
```

```
Loaded 100000 pretrained word vectors
cat dog 0.6716836662792491
dog pet 0.6842064029669219
dogs cats 0.7074389328052404
paris france 0.7775108541288561
germany berlin 0.7420295235998392
['cats', 'kitty', 'kitten', 'feline', 'kitties']
['dogs', 'puppy', 'Dog', 'doggie', 'canine']
['dog', 'pooches', 'Dogs', 'doggies', 'canines']
['france', 'Paris', 'london', 'berlin', 'tokyo']
['austria', 'europe', 'german', 'berlin', 'poland']
```

In [5]:
```python
class BoV():
    def __init__(self, w2v):
        self.w2v = w2v

    def encode(self, sentences, idf=False):
        # takes a list of sentences, outputs a numpy array of sentence embeddings
        # see TP1 for help
        sentemb = []
        for sent in sentences:
            if idf is False:
                # mean of word vectors
                keys = list(self.w2v.word2vec.keys())
                sentemb.append(np.mean([self.w2v.word2vec[w] for w in sent if w in keys]\
                                       , axis=0))
                #assert False, 'TODO: fill in the blank'
            else:
                # idf-weighted mean of word vectors
                keys = list(self.w2v.word2vec.keys())
                vec = np.sum([idf[w]*self.w2v.word2vec[w] for w in sent if w in keys]\
                                       , axis=0)/np.sum([idf[w] for w in sent if w in keys])
                sentemb.append(vec)
                #assert False, 'TODO: fill in the blank'
        return np.vstack(sentemb)

    def most_similar(self, s, sentences, idf=False, K=5):
        # get most similar sentences and **print** them
        #keys = self.encode(sentences, idf)
        #query = self.encode([s], idf)
        scores = list(map(lambda x: self.score(x,s) if x!=s else -1,sentences))
        arg = np.argsort(scores)
        most_sim = [sentences[i] for i in np.flipud(arg[-K:])]
        print('The most similar',K, 'sentences to: "',' '.join(s),'" are:\n')
        for i in range(len(most_sim)):
            print('Rank ',i+1 , ': ', ' '.join(most_sim[i]))


    def score(self, s1, s2, idf=False):
        # cosine similarity: use   np.dot  and  np.linalg.norm
        s1_encode = self.encode([s1], idf)
        s2_encode = self.encode([s2], idf)
        return float(np.dot(s1_encode,s2_encode.T)/(np.linalg.norm(s1_encode)*np.linalg.norm(s2_encode)))

    def build_idf(self, sentences):
        # build the idf dictionary: associate each word to its idf value
        idf = {}
        for sent in sentences:
            for w in set(sent): # Using set to avoid word repetition in a sentence
                idf[w] = idf.get(w, 0) + 1
        for w in list(idf.keys()):
            idf[w] = max(1, np.log10(len(sentences) / (idf[w])))
        return idf
```

In [687]:
```python
L = train_sent+dev_sent+test_sent
```

In [6]:
```python
w2v = Word2vec(os.path.join(PATH_TO_DATA, 'crawl-300d-200k.vec'), nmax=100000)
s2v = BoV(w2v)

# Load sentences in "PATH_TO_DATA/sentences.txt"
sentences = []
with open('/Users/adilrhoulam/Downloads/data-2/sentences.txt') as f:
    for i, line in enumerate(f):
        sent = line.rstrip().split()
        sentences.append(sent)

# Build idf scores for each word
idf = s2v.build_idf(sentences)

# You will be evaluated on the output of the following:
s2v.most_similar('' if not sentences else sentences[10], sentences)  # BoV-mean
score = s2v.score('' if not sentences else sentences[7], '' if not sentences else sentences[13])
print('\n- Cosinus score using BoV-mean between "',' '.join(sentences[7])\
      ,'" and "',' '.join(sentences[13]),'" is :', score,'\n')



s2v.most_similar('' if not sentences else sentences[10], sentences, idf)  # BoV-idf
score = s2v.score('' if not sentences else sentences[7], '' if not sentences else sentences[13], idf)
print('\n- Cosinus score using BoV-idf between "',' '.join(sentences[7])\
      ,'" and "',' '.join(sentences[13]),'" is :', score)
```

```
Loaded 100000 pretrained word vectors
The most similar 5 sentences to: " 1 smiling african american boy . " are:

Rank  1 :  an african american man smiling .
Rank  2 :  a little african american boy and girl looking up .
Rank  3 :  an afican american woman standing behind two small african american children .
Rank  4 :  an african american man is sitting .
Rank  5 :  a girl in black hat holding an african american baby .

- Cosinus score using BoV-mean between " 1 man singing and 1 man playing a saxophone in a concert . " and " 10 people venture out to go crosscountry skiing . " is : 0.5726258859719606

The most similar 5 sentences to: " 1 smiling african american boy . " are:

Rank  1 :  an african american man smiling .
Rank  2 :  a little african american boy and girl looking up .
Rank  3 :  an afican american woman standing behind two small african american children .
Rank  4 :  an african american man is sitting .
Rank  5 :  a girl in black hat holding an african american baby .

- Cosinus score using BoV-idf between " 1 man singing and 1 man playing a saxophone in a concert . " and " 10 people venture out to go crosscountry skiing . " is : 0.4751450875368781
```

2) Multilingual (English-French) word embeddings

Let's consider a bilingual dictionary of size V_a (e.g French-English).

Let's define **X** and **Y** the **French** and **English** matrices.

They contain the embeddings associated to the words in the bilingual dictionary.

We want to find a **mapping W** that will project the source word space (e.g French) to the target word space (e.g English).

Procrustes : **W* = argmin || W.X - Y || s.t W^T.W = Id** has a closed form solution: **W = U.V^T where U.Sig.V^T = SVD(Y.X^T)**

In what follows, you are asked to:

In [7]:
```python
# 1 - Download and load 50k first vectors of
#     https://s3-us-west-1.amazonaws.com/fasttext-vectors/wiki.en.vec
#     https://s3-us-west-1.amazonaws.com/fasttext-vectors/wiki.fr.vec

# TYPE CODE HERE
nmax = 50000
engvec = {}
with io.open('/Users/adilrhoulam/Downloads/wiki.en.vec', encoding='utf-8') as f:
    for i, line in enumerate(f):
        word, vec = line.split(' ', 1)
        engvec[word] = np.fromstring(vec, sep=' ')
        if i == (nmax - 1):
            break
frvec = {}
with io.open('/Users/adilrhoulam/Downloads/wiki.fr.vec', encoding='utf-8') as f:
    for i, line in enumerate(f):
        word, vec = line.split(' ', 1)
        frvec[word] = np.fromstring(vec, sep=' ')
        if i == (nmax - 1):
            break
```

In [8]:
```python
# 2 - Get words that appear in both vocabs (= identical character strings)
#     Use it to create the matrix X and Y (of aligned embeddings for these words)

# TYPE CODE HERE
identical_words = set()
keys_eng = set(engvec.keys())
keys_fr = set(frvec.keys())
for w in keys_eng:
    if w in keys_fr:
        identical_words.add(w)
X = []
Y = []
for w in identical_words:
    X.append(frvec[w])
    Y.append(engvec[w])
X,Y = np.array(X),np.array(Y)
```

In [9]:
```python
# 3 - Solve the Procrustes using the scipy package and: scipy.linalg.svd() and get the optimal W
#     Now W*French_vector is in the same space as English_vector

# TYPE CODE HERE
R = np.dot(Y,X.T)
U, S, V = np.linalg.svd(R) #Python3 svd has become in the numpy.linalg package
W = np.dot(U,V)
```

In [10]:
```python
# 4 - After alignment with W, give examples of English nearest neighbors of some French words (and vice versa)
#     You will be evaluated on that part and the code above

# TYPE CODE HERE
words = list(identical_words)
# French words nearest neighbors ******************************************************
print('**************************** French words nearest neighbors **********************************')
A = np.dot(W,X)
def nearest_neighbors_fr(A,Y,index,words,K=5):
    vec = A[index]
    scores = list(map(lambda x: float(np.dot(x,vec)/(np.linalg.norm(x)*np.linalg.norm(vec)))\
                      ,list(Y)))
    arg = np.argsort(scores)
    return [words[i] for i in np.flipud(arg[-K:])]

index = words.index('place')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "place" are:\n '\
      ,nearest)

index = words.index('costumes')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "costumes" are:\n '\
      ,nearest)

index = words.index('bizarre')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "bizarre" are:\n '\
      ,nearest)

index = words.index('programme')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "programme" are:\n '\
      ,nearest)

index = words.index('populations')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "populations" are:\n '\
      ,nearest)

index = words.index('automobiles')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "automobiles" are:\n '\
      ,nearest)

index = words.index('liaison')
nearest = nearest_neighbors_fr(A,Y,index,words)
print('\nThe nearest 5 english words neighbors to the french word "liaison" are:\n '\
      ,nearest)

# English words nearest neighbors ******************************************************
print('\n\n**************************** English words nearest neighbors **********************************')
def nearest_neighbors_eng(A,Y,index,words,K=5):
    vec = Y[index]
    scores = list(map(lambda x: float(np.dot(x,vec)/(np.linalg.norm(x)*np.linalg.norm(vec)))\
                      ,list(A)))
    arg = np.argsort(scores)
    return [words[i] for i in np.flipud(arg[-K:])]

index = words.index('motel')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "motel" are:\n '\
      ,nearest)

index = words.index('liberation')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "liberation" are:\n '\
      ,nearest)

index = words.index('fast')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "fast" are:\n '\
      ,nearest)

index = words.index('weapons')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "weapons" are:\n '\
      ,nearest)

index = words.index('cultures')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "cultures" are:\n '\
      ,nearest)

index = words.index('food')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "food" are:\n '\
      ,nearest)

index = words.index('freedom')
nearest = nearest_neighbors_eng(A,Y,index,words)
print('\nThe nearest 5 french words neighbors to the english word "freedom" are:\n '\
      ,nearest)
```

```
**************************** French words nearest neighbors **********************************

The nearest 5 english words neighbors to the french word "place" are:
  ['place', 'places', 'spot', 'takes', 'taking']

The nearest 5 english words neighbors to the french word "costumes" are:
  ['costumes', 'costume', 'attire', 'dress', 'shirts']

The nearest 5 english words neighbors to the french word "bizarre" are:
  ['bizarre', 'strange', 'weird', 'horrible', 'grotesque']

The nearest 5 english words neighbors to the french word "programme" are:
  ['programme', 'programmes', 'program', 'bbc', 'itv']

The nearest 5 english words neighbors to the french word "populations" are:
  ['populations', 'population', 'migrations', 'migrants', 'areas']

The nearest 5 english words neighbors to the french word "automobiles" are:
  ['automobiles', 'automobile', 'cars', 'automotive', 'trucks']

The nearest 5 english words neighbors to the french word "liaison" are:
  ['liaison', 'liaisons', 'coordination', 'staff', 'aide']


**************************** English words nearest neighbors **********************************

The nearest 5 french words neighbors to the english word "motel" are:
  ['motel', 'hotel', 'hotels', 'diner', 'restaurant']

The nearest 5 french words neighbors to the english word "liberation" are:
  ['liberation', 'libération', 'revolutionary', 'independence', 'front']

The nearest 5 french words neighbors to the english word "fast" are:
  ['fast', 'slow', 'pace', 'speed', 'agile']

The nearest 5 french words neighbors to the english word "weapons" are:
  ['weapons', 'weapon', 'munitions', 'projectiles', 'missiles']

The nearest 5 french words neighbors to the english word "cultures" are:
  ['cultures', 'culture', 'traditions', 'peoples', 'civilization']

The nearest 5 french words neighbors to the english word "food" are:
  ['food', 'foods', 'nutrition', 'meat', 'desserts']

The nearest 5 french words neighbors to the english word "freedom" are:
  ['freedom', 'liberty', 'conscience', 'democracy', 'rights']
```

If you want to dive deeper on this subject: https://github.com/facebookresearch/MUSE

3) Sentence classification with BoV and scikit-learn

In [692]:
```python
# 1 - Load train/dev/test of Stanford Sentiment TreeBank (SST)
#     (https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf)

# TYPE CODE HERE
train_sent = []
train_labels = []
path = '/Users/adilrhoulam/Downloads/nlp_project-3/data/SST/stsa.fine.train'
with io.open(path, encoding='utf-8') as f:
    for line in f:
        label, sent = line.split(' ', 1)
        train_sent.append(sent.rstrip().split())
        train_labels.append(int(label))

dev_sent = []
dev_labels = []
path = '/Users/adilrhoulam/Downloads/nlp_project-3/data/SST/stsa.fine.dev'
with io.open(path, encoding='utf-8') as f:
    for line in f:
        label, sent = line.split(' ', 1)
        dev_sent.append(sent.rstrip().split())
        dev_labels.append(int(label))

test_sent = []
path = '/Users/adilrhoulam/Downloads/nlp_project-3/data/SST/stsa.fine.test.X'
with io.open(path, encoding='utf-8') as f:
    for line in f:
        test_sent.append(line.rstrip().split())
```

In [699]:
```python
# 2 - Encode sentences with the BoV model above

# TYPE CODE HERE
PATH_TO_DATA = "/Users/adilrhoulam/Downloads/data-2"
w2v = Word2vec(os.path.join(PATH_TO_DATA, 'crawl-300d-200k.vec'), nmax=100000)
s2v = BoV(w2v)

train_sent_vec_mean = s2v.encode(train_sent)
dev_sent_vec_mean = s2v.encode(dev_sent)
test_sent_vec_mean = s2v.encode(test_sent)

idf = s2v.build_idf(train_sent+dev_sent+test_sent)
train_sent_vec_idf = s2v.encode(train_sent,idf)
dev_sent_vec_idf = s2v.encode(dev_sent,idf)
test_sent_vec_idf = s2v.encode(test_sent,idf)
```

```
Loaded 100000 pretrained word vectors
```

In [940]:
```python
# 3 - Learn Logistic Regression on top of sentence embeddings using scikit-learn
#     (consider tuning the L2 regularization on the dev set)

# TYPE CODE HERE
from sklearn.linear_model import LogisticRegression
C = [0.001,0.01,0.05,0.1,0.5,1,3,10,20,30,40,50,60,70,80,90,100,200,500,1000]
scores = []
for c in C:
    logReg = LogisticRegression(penalty='l2',C = c)
    logReg.fit(train_sent_vec_mean,train_labels)
    score = logReg.score(dev_sent_vec_mean,dev_labels)
    scores.append(score)
c1 = C[np.argmax(scores)]
print('The best score on dev set when using mean BoV is : ',round(np.max(scores)*100,2),'% \
and is given by given by a regularization parameter C = ',c1)

scores = []
for c in C:
    logReg = LogisticRegression(penalty='l2',C = c)
    logReg.fit(train_sent_vec_idf,train_labels)
    score = logReg.score(dev_sent_vec_idf,dev_labels)
    scores.append(score)
c2 = C[np.argmax(scores)]
print('The best score on dev set when using idf BoV is : ',round(np.max(scores)*100,2),'% \
and is given by given by a regularization parameter C = ',c2)
logReg = LogisticRegression(penalty='l2',C = c1)
logReg.fit(train_sent_vec_mean,train_labels)
score = logReg.score(train_sent_vec_mean,train_labels)
print('The score on train data using the tuned parameter on mean BoV is :',\
      round(score*100,2))
logReg = LogisticRegression(penalty='l2',C = c2)
logReg.fit(train_sent_vec_idf,train_labels)
score = logReg.score(train_sent_vec_idf,train_labels)
print('The score on train data using the tuned parameter on idf BoV is :',\
      round(score*100,2))
```

```
The best score on dev set when using mean BoV is :  43.69 % and is given by given by a regularization parameter C =  0.5
The best score on dev set when using idf BoV is :  42.51 % and is given by given by a regularization parameter C =  0.5
The score on train data using the tuned parameter on mean BoV is : 47.94
The score on train data using the tuned parameter on idf BoV is : 48.68
```

In [777]:
```python
# 4 - Produce 2210 predictions for the test set (in the same order). One line = one prediction (=0,1,2,3,4).
#     Attach the output file "logreg_bov_y_test_sst.txt" to your deliverable.
#     You will be evaluated on the results of the test set.

# TYPE CODE HERE
path = '/Users/adilrhoulam/Downloads/nlp_project-3/logreg_bov_y_test_sst.txt'
thefile = open(path, 'w')
logReg = LogisticRegression(penalty='l2',C = 20)
logReg.fit(train_sent_vec_mean,train_labels)
predictions = logReg.predict(test_sent_vec_mean)
for i in range(len(test_sent_vec_mean)):
    thefile.write("%s\n" % predictions[i])
```

In [942]:
```python
# BONUS!
# 5 - Try to improve performance with another classifier
#     Attach the output file "XXX_bov_y_test_sst.txt" to your deliverable (where XXX = the name of the classifier)

# TYPE CODE HERE
#Neural Network
from sklearn.svm import SVC
clf = SVC(C=200, cache_size=250, class_weight=None, coef0=0.0,\
    decision_function_shape='sigmoid', degree=4, gamma='auto', kernel='rbf',\
    max_iter=-1, probability=False, random_state=None, shrinking=True,\
    tol=0.1, verbose=False)
clf.fit(train_sent_vec_mean, train_labels)
clf.score(dev_sent_vec_mean,dev_labels)
```

Out[942]:
```
0.43869209809264303
```

In [974]:
```python
path = '/Users/adilrhoulam/Downloads/nlp_project-3/SVM_bov_y_test_sst.txt'
thefile = open(path, 'w')
predictions = clf.predict(test_sent_vec_mean)
for i in range(len(test_sent_vec_mean)):
    thefile.write("%s\n" % predictions[i])
```

4) Sentence classification with LSTMs in Keras

4.1 - Preprocessing

In [860]:
```python
import keras
```

In [861]:
```python
# 1 - Load train/dev/test sets of SST
PATH_TO_DATA = '/Users/adilrhoulam/Downloads/nlp_project-3/data/'

# TYPE CODE HERE
train_sent = []
train_labels = []
path = PATH_TO_DATA+'SST/stsa.fine.train'
with io.open(path, encoding='utf-8') as f:
    for line in f:
        label, sent = line.split(' ', 1)
        train_sent.append(sent)
        train_labels.append(int(label))

dev_sent = []
dev_labels = []
path = PATH_TO_DATA+'SST/stsa.fine.dev'
with io.open(path, encoding='utf-8') as f:
    for line in f:
        label, sent = line.split(' ', 1)
        dev_sent.append(sent)
        dev_labels.append(int(label))

test_sent = []
path = PATH_TO_DATA+'SST/stsa.fine.test.X'
with io.open(path, encoding='utf-8') as f:
    for line in f:
        test_sent.append(line)
```

In [862]:
```python
# 2 - Transform text to integers using keras.preprocessing.text.one_hot function
#     https://keras.io/preprocessing/text/

# TYPE CODE HERE
from keras.preprocessing import text
train_sent = list(map(lambda x: text.one_hot(x,len(x)),train_sent))
dev_sent = list(map(lambda x: text.one_hot(x,len(x)),dev_sent))
test_sent = list(map(lambda x: text.one_hot(x,len(x)),test_sent))
```

Padding input data

Models in Keras (and elsewhere) take batches of sentences of the same length as input. It is because Deep Learning framework have been designed to handle well Tensors, which are particularly suited for fast computation on the GPU.

Since sentences have different sizes, we "pad" them. That is, we add dummy "padding" tokens so that they all have the same length.

The input to a Keras model thus has this size : (batchsize, maxseqlen) where maxseqlen is the maximum length of a sentence in the batch.

In [864]:
```python
# 3 - Pad your sequences using keras.preprocessing.sequence.pad_sequences
#     https://keras.io/preprocessing/sequence/

# TYPE CODE HERE
from keras.preprocessing import sequence
vec_len = np.vectorize(len)
maxseqlen = max(max(vec_len(train_sent)),max(vec_len(dev_sent)),max(vec_len(test_sent)))
x_train = sequence.pad_sequences(train_sent,maxlen=maxseqlen)
x_dev = sequence.pad_sequences(dev_sent,maxlen=maxseqlen)
x_test = sequence.pad_sequences(test_sent,maxlen=maxseqlen)
```

4.2 - Design and train your model

In [954]:
```python
# 4 - Design your encoder + classifier using keras.layers
#     In Keras, Torch and other deep learning framework, we create a "container" which is the Sequential() module.
#     Then we add components to this contained : the lookuptable, the LSTM, the classifier etc.
#     All of these components are contained in the Sequential() and are trained together.


# ADAPT CODE BELOW


from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Activation
max_vec = np.vectorize(max)
embed_dim  = 32  # word embedding dimension
nhid       = 64  # number of hidden units in the LSTM
vocab_size = max(max(max_vec(train_sent)),max(max_vec(dev_sent)),max(max_vec(test_sent))) + 1
# size of the vocabulary
n_classes  = 5

model = Sequential()
model.add(Embedding(vocab_size, embed_dim))
model.add(LSTM(nhid, dropout=0.1, recurrent_dropout=0.1))
model.add(Dense(n_classes, activation='sigmoid'))
```

In [955]:
```python
# 5 - Define your loss/optimizer/metrics

# MODIFY CODE BELOW

loss_classif     =  'categorical_crossentropy' # find the right loss for multi-class classification
optimizer        =  'adam' # find the right optimizer
metrics_classif  =  ['accuracy']

# Observe how easy (but blackboxed) this is in Keras
model.compile(loss=loss_classif,
              optimizer=optimizer,
              metrics=metrics_classif)
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_42 (Embedding)     (None, None, 32)          8992
_________________________________________________________________
lstm_34 (LSTM)               (None, 64)                24832
_________________________________________________________________
dense_44 (Dense)             (None, 5)                 325
=================================================================
Total params: 34,149
Trainable params: 34,149
Non-trainable params: 0
_________________________________________________________________
None
```

In [956]:
```python
# 6 - Train your model and find the best hyperparameters for your dev set
#     you will be evaluated on the quality of your predictions on the test set

# ADAPT CODE BELOW
from keras.utils import np_utils
bs = 64
n_epochs = 20
y_train = np_utils.to_categorical(train_labels)
y_dev = np_utils.to_categorical(dev_labels)
history = model.fit(x_train, y_train, batch_size=bs, epochs=n_epochs, validation_data=(x_dev, y_dev))
```

```
Train on 8544 samples, validate on 1101 samples
Epoch 1/20
8544/8544 [==============================] - 13s 1ms/step - loss: 1.5745 - acc: 0.2656 - val_loss: 1.5713 - val_acc: 0.2534
Epoch 2/20
8544/8544 [==============================] - 7s 865us/step - loss: 1.5668 - acc: 0.2741 - val_loss: 1.5718 - val_acc: 0.2652
Epoch 3/20
8544/8544 [==============================] - 7s 829us/step - loss: 1.5613 - acc: 0.2779 - val_loss: 1.5689 - val_acc: 0.2507
Epoch 4/20
8544/8544 [==============================] - 7s 865us/step - loss: 1.5544 - acc: 0.2828 - val_loss: 1.5709 - val_acc: 0.2534
Epoch 5/20
8544/8544 [==============================] - 7s 876us/step - loss: 1.5468 - acc: 0.2904 - val_loss: 1.5750 - val_acc: 0.2389
Epoch 6/20
8544/8544 [==============================] - 7s 872us/step - loss: 1.5388 - acc: 0.3016 - val_loss: 1.5857 - val_acc: 0.2352
Epoch 7/20
8544/8544 [==============================] - 8s 911us/step - loss: 1.5311 - acc: 0.3096 - val_loss: 1.5929 - val_acc: 0.2398
Epoch 8/20
8544/8544 [==============================] - 7s 813us/step - loss: 1.5258 - acc: 0.3083 - val_loss: 1.5964 - val_acc: 0.2452
Epoch 9/20
8544/8544 [==============================] - 7s 842us/step - loss: 1.5211 - acc: 0.3117 - val_loss: 1.6253 - val_acc: 0.2416
Epoch 10/20
8544/8544 [==============================] - 7s 822us/step - loss: 1.5169 - acc: 0.3110 - val_loss: 1.6208 - val_acc: 0.2434
Epoch 11/20
8544/8544 [==============================] - 7s 814us/step - loss: 1.5129 - acc: 0.3207 - val_loss: 1.6082 - val_acc: 0.2452
Epoch 12/20
8544/8544 [==============================] - 7s 857us/step - loss: 1.5084 - acc: 0.3164 - val_loss: 1.6187 - val_acc: 0.2480
Epoch 13/20
8544/8544 [==============================] - 7s 822us/step - loss: 1.5036 - acc: 0.3221 - val_loss: 1.6319 - val_acc: 0.2425
Epoch 14/20
8544/8544 [==============================] - 7s 815us/step - loss: 1.5004 - acc: 0.3216 - val_loss: 1.6393 - val_acc: 0.2407
Epoch 15/20
8544/8544 [==============================] - 8s 993us/step - loss: 1.4952 - acc: 0.3246 - val_loss: 1.6345 - val_acc: 0.2371
Epoch 16/20
8544/8544 [==============================] - 7s 855us/step - loss: 1.4901 - acc: 0.3267 - val_loss: 1.6501 - val_acc: 0.2507
Epoch 17/20
8544/8544 [==============================] - 7s 822us/step - loss: 1.4874 - acc: 0.3247 - val_loss: 1.6485 - val_acc: 0.2498
Epoch 18/20
8544/8544 [==============================] - 7s 811us/step - loss: 1.4805 - acc: 0.3262 - val_loss: 1.6583 - val_acc: 0.2589
Epoch 19/20
8544/8544 [==============================] - 7s 809us/step - loss: 1.4769 - acc: 0.3297 - val_loss: 1.6720 - val_acc: 0.2407
Epoch 20/20
8544/8544 [==============================] - 7s 815us/step - loss: 1.4728 - acc: 0.3319 - val_loss: 1.6653 - val_acc: 0.2470
```

In [961]:
```python
# 7 - Generate your predictions on the test set using model.predict(x_test)
#     https://keras.io/models/model/
#     Log your predictions in a file (one line = one integer: 0,1,2,3,4)
#     Attach the output file "logreg_lstm_y_test_sst.txt" to your deliverable.

# TYPE CODE HERE
path = '/Users/adilrhoulam/Downloads/nlp_project-3/logreg_lstm_y_test_sst.txt'
thefile = open(path, 'w')
predictions = model.predict_classes(x_test)
for i in range(len(test_sent_vec_mean)):
    thefile.write("%s\n" % predictions[i])
```

4.3 -- innovate !

In [969]:
```python
# 8 - Open question: find a model that is better on your dev set
#     (e.g: use a 1D ConvNet, use a better classifier, pretrain your lookup tables ..)
#     you will get point if the results on the test set are better: be careful of not overfitting your dev set too much..
#     Attach the output file "XXX_XXX_y_test_sst.txt" to your deliverable.

# TYPE CODE HERE

from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense, Activation, Conv1D, Dropout, GlobalMaxPooling1D
max_vec = np.vectorize(max)
embed_dim  = 32  # word embedding dimension
nhid       = 64  # number of hidden units in the LSTM
vocab_size = max(max(max_vec(train_sent)),max(max_vec(dev_sent)),max(max_vec(test_sent))) + 1
# size of the vocabulary
n_classes  = 5

model = Sequential()

model.add(Embedding(vocab_size,embed_dim))
model.add(Dropout(0.2))
model.add(Conv1D(64,3,padding='valid',activation='relu',strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(64))
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(Dense(n_classes))
model.add(Activation('softmax'))
print(model.summary())
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_46 (Embedding)     (None, None, 32)          8992
_________________________________________________________________
dropout_17 (Dropout)         (None, None, 32)          0
_________________________________________________________________
conv1d_9 (Conv1D)            (None, None, 64)          6208
_________________________________________________________________
global_max_pooling1d_8 (Glob (None, 64)                0
_________________________________________________________________
dense_51 (Dense)             (None, 64)                4160
_________________________________________________________________
dropout_18 (Dropout)         (None, 64)                0
_________________________________________________________________
activation_17 (Activation)   (None, 64)                0
_________________________________________________________________
dense_52 (Dense)             (None, 5)                 325
_________________________________________________________________
activation_18 (Activation)   (None, 5)                 0
=================================================================
Total params: 19,685
Trainable params: 19,685
Non-trainable params: 0
_________________________________________________________________
None
```

In [970]:
```python
from keras.utils import np_utils
bs = 64
n_epochs = 15
y_train = np_utils.to_categorical(train_labels)
y_dev = np_utils.to_categorical(dev_labels)
history = model.fit(x_train, y_train, batch_size=bs, epochs=n_epochs, validation_data=(x_dev, y_dev))
```

```
Train on 8544 samples, validate on 1101 samples
Epoch 1/15
8544/8544 [==============================] - 6s 712us/step - loss: 1.5760 - acc: 0.2577 - val_loss: 1.5730 - val_acc: 0.2534
Epoch 2/15
8544/8544 [==============================] - 1s 151us/step - loss: 1.5694 - acc: 0.2727 - val_loss: 1.5736 - val_acc: 0.2534
Epoch 3/15
8544/8544 [==============================] - 1s 163us/step - loss: 1.5661 - acc: 0.2717 - val_loss: 1.5712 - val_acc: 0.2607
Epoch 4/15
8544/8544 [==============================] - 1s 170us/step - loss: 1.5628 - acc: 0.2774 - val_loss: 1.5726 - val_acc: 0.2625
Epoch 5/15
8544/8544 [==============================] - 2s 182us/step - loss: 1.5570 - acc: 0.2896 - val_loss: 1.5723 - val_acc: 0.2598
Epoch 6/15
8544/8544 [==============================] - 1s 169us/step - loss: 1.5493 - acc: 0.2974 - val_loss: 1.5741 - val_acc: 0.2543
Epoch 7/15
8544/8544 [==============================] - 1s 173us/step - loss: 1.5379 - acc: 0.3113 - val_loss: 1.5868 - val_acc: 0.2598
Epoch 8/15
8544/8544 [==============================] - 1s 169us/step - loss: 1.5231 - acc: 0.3272 - val_loss: 1.5848 - val_acc: 0.2589
Epoch 9/15
8544/8544 [==============================] - 1s 169us/step - loss: 1.5084 - acc: 0.3397 - val_loss: 1.6009 - val_acc: 0.2470
Epoch 10/15
8544/8544 [==============================] - 1s 174us/step - loss: 1.4899 - acc: 0.3522 - val_loss: 1.6043 - val_acc: 0.2489
Epoch 11/15
8544/8544 [==============================] - 1s 170us/step - loss: 1.4684 - acc: 0.3638 - val_loss: 1.6236 - val_acc: 0.2461
Epoch 12/15
8544/8544 [==============================] - 1s 154us/step - loss: 1.4481 - acc: 0.3824 - val_loss: 1.6395 - val_acc: 0.2480
```