english_german_translation_with_keras

April 18, 2019

1 German to English Translation

First, we must load the data in a way that preserves the Unicode German characters. The function below called `load doc()` will load the file as a blob of text.

```
In [8]: import string
        import re
        from pickle import dump
        from unicodedata import normalize
        from numpy import array
```

```
In [19]: # load doc into memory
         def load_doc(filename):
             # open the file as read only
             file = open(filename, mode='rt', encoding='utf-8')
             # read all text
             text = file.read()
             # close the file
             file.close()
             return text
```

Each line contains a single pair of phrases, first English and then German, separated by a tab character. We must split the loaded text by line and then by phrase. The function to `pairs()` below will split the loaded text.

```
In [20]: # split a loaded document into sentences
         def to_pairs(doc):
           lines = doc.strip().split('\n')
           pairs = [line.split('\t') for line in lines]
           return pairs
```

We are now ready to clean each sentence. We will perform the following specific cleaning operations: Remove all non-printable characters. Remove all punctuation characters. Normalize all Unicode characters to ASCII (e.g. Latin characters). Normalize the case to lowercase. Remove any remaining tokens that are not alphabetic. We perform these operations on each phrase for each pair in the loaded dataset. Function `clean_pairs()` implements these operations.

```
In [21]: # clean a list of lines
         def clean_pairs(lines):
                 cleaned = list()
                 # prepare regex for char filtering
                 re_punc = re.compile('[%s]' % re.escape(string.punctuation))
                 re_print = re.compile('[^%s]' % re.escape(string.printable))
                 for pair in lines:
                         clean_pair = list()
                         for line in pair:
                                 # normalize unicode characters
                                 line = normalize('NFD', line).encode('ascii', 'ignore')
                                 line = line.decode('UTF-8')
                                 # tokenize on white space
                                 line = line.split()
                                 # convert to lowercase
                                 line = [word.lower() for word in line]
                                 # remove punctuation from each token
                                 line = [re_punc.sub('', w) for w in line]
                                 # remove non-printable chars form each token
                                 line = [re_print.sub('', w) for w in line]
                                 # remove tokens with numbers in them
                                 line = [word for word in line if word.isalpha()]
                                 # store as string
                                 clean_pair.append(' '.join(line))
                         cleaned.append(clean_pair)
                 return array(cleaned)
```

Cleaned data will be saved to a file using picle API, Python's compression mechanism.

```
In [22]: # save a list of clean sentences to file
         def save_clean_data(sentences, filename):
                 dump(sentences, open(filename, 'wb'))
                 print('Saved: %s' % filename)
```

We are no ready to process the data

```
In [13]: !pwd
```

```
/e/CLASSES/DeepLearning_cscie89/code/codedl11
```

```
In [23]: # load dataset
         filename = 'deu.txt'
         doc = load_doc(filename)
         type(doc)
```

```
Out[23]: str
```

```
In [24]: # Split into english-german pairs
         pairs = to_pairs(doc)
         # clean sentences
         clean_pairs = clean_pairs(pairs)
         # save clean pairs to file
         save_clean_data(clean_pairs, 'english-german.pkl')
```

```
Saved: english-german.pkl
```

```
In [ ]: # spot check
```

```
In [25]: for i in range(100):
                 print('[%s] => [%s]' % (clean_pairs[i,0], clean_pairs[i,1]))
```

```
[hi] => [hallo]
[hi] => [gru gott]
[run] => [lauf]
[wow] => [potzdonner]
[wow] => [donnerwetter]
[fire] => [feuer]
[help] => [hilfe]
[help] => [zu hulf]
[stop] => [stopp]
[wait] => [warte]
[go on] => [mach weiter]
[hello] => [hallo]
[i ran] => [ich rannte]
[i see] => [ich verstehe]
[i see] => [aha]
[i try] => [ich probiere es]
[i won] => [ich hab gewonnen]
[i won] => [ich habe gewonnen]
[smile] => [lacheln]
[cheers] => [zum wohl]
[freeze] => [keine bewegung]
[freeze] => [stehenbleiben]
[got it] => [verstanden]
[got it] => [einverstanden]
[he ran] => [er rannte]
[he ran] => [er lief]
[hop in] => [mach mit]
[hug me] => [druck mich]
[hug me] => [nimm mich in den arm]
[hug me] => [umarme mich]
[i fell] => [ich fiel]
[i fell] => [ich fiel hin]
[i fell] => [ich sturzte]
[i fell] => [ich bin hingefallen]
```

[i fell] => [ich bin gesturzt]
[i know] => [ich wei]
[i lied] => [ich habe gelogen]
[i lost] => [ich habe verloren]
[im] => [ich bin jahre alt]
[im] => [ich bin]
[im ok] => [mir gehts gut]
[im ok] => [es geht mir gut]
[im up] => [ich bin wach]
[im up] => [ich bin auf]
[no way] => [unmoglich]
[no way] => [das gibts doch nicht]
[no way] => [ausgeschlossen]
[no way] => [in keinster weise]
[really] => [wirklich]
[really] => [echt]
[really] => [im ernst]
[thanks] => [danke]
[try it] => [versuchs]
[why me] => [warum ich]
[ask tom] => [frag tom]
[ask tom] => [fragen sie tom]
[ask tom] => [fragt tom]
[be cool] => [entspann dich]
[be fair] => [sei nicht ungerecht]
[be fair] => [sei fair]
[be nice] => [sei nett]
[be nice] => [seien sie nett]
[beat it] => [geh weg]
[beat it] => [hau ab]
[beat it] => [verschwinde]
[beat it] => [verdufte]
[beat it] => [mach dich fort]
[beat it] => [zieh leine]
[beat it] => [mach dich vom acker]
[beat it] => [verzieh dich]
[beat it] => [verkrumele dich]
[beat it] => [troll dich]
[beat it] => [zisch ab]
[beat it] => [pack dich]
[beat it] => [mach ne fliege]
[beat it] => [schwirr ab]
[beat it] => [mach die sause]
[beat it] => [scher dich weg]
[beat it] => [scher dich fort]
[call me] => [ruf mich an]
[come in] => [komm herein]
[come in] => [herein]

```
[come on] => [komm]
[come on] => [kommt]
[come on] => [mach schon]
[come on] => [macht schon]
[come on] => [komm schon]
[get tom] => [hol tom]
[get out] => [raus]
[get out] => [geh raus]
[go away] => [geh weg]
[go away] => [hau ab]
[go away] => [verschwinde]
[go away] => [verdufte]
[go away] => [mach dich fort]
[go away] => [zieh leine]
[go away] => [mach dich vom acker]
[go away] => [verzieh dich]
[go away] => [verkrumele dich]
[go away] => [troll dich]
```

We need to perform a few more preparations. Notably, creat training and testing datasets

```
In [26]: from pickle import load
         from pickle import dump
         from numpy.random import shuffle

In [27]: # load a clean dataset
         def load_clean_sentences(filename):
                 return load(open(filename, 'rb'))

         # save a list of clean sentences to file
         def save_clean_data(sentences, filename):
                 dump(sentences, open(filename, 'wb'))
                 print('Saved: %s' % filename)

In [28]: # load dataset
         raw_dataset = load_clean_sentences('english-german.pkl')
         raw_dataset.size

Out[28]: 338394
```

To make our work faser we will reduce the datase to 10,000 samples

```
In [31]: # reduce dataset size
         n_sentences = 10000
         dataset = raw_dataset[:n_sentences, :]
         dataset.size

Out[31]: 20000
```

```
In [32]: # random shuffle
         shuffle(dataset)
         # split into train/test
         train, test = dataset[:9000], dataset[9000:]

In [33]: print(train.size, train.shape)

18000 (9000, 2)


In [34]: print(test.size, test.shape)

2000 (1000, 2)


In [35]: # save
         save_clean_data(dataset, 'english-german-both.pkl')
         save_clean_data(train, 'english-german-train.pkl')
         save_clean_data(test, 'english-german-test.pkl')

Saved: english-german-both.pkl
Saved: english-german-train.pkl
Saved: english-german-test.pkl
```

2 Prepare and Train the Model

Now we are finally ready to analyze those sentence pairs.

```
In [36]: from pickle import load
         from numpy import array
         from keras.preprocessing.text import Tokenizer
         from keras.preprocessing.sequence import pad_sequences
         from keras.utils import to_categorical
         # from keras.utils.vis_utils import plot_model
         from keras.models import Sequential
         from keras.layers import LSTM
         from keras.layers import Dense
         from keras.layers import Embedding
         from keras.layers import RepeatVector
         from keras.layers import TimeDistributed
         from keras.callbacks import ModelCheckpoint

c:\programs\python\python36\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [37]:
```
# load a clean dataset
def load_clean_sentences(filename):
        return load(open(filename, 'rb'))

# fit a tokenizer
def create_tokenizer(lines):
        tokenizer = Tokenizer()
        tokenizer.fit_on_texts(lines)
        return tokenizer

# max sentence length
def max_length(lines):
        return max(len(line.split()) for line in lines)

# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
        # integer encode sequences
        X = tokenizer.texts_to_sequences(lines)
        # pad sequences with 0 values
        X = pad_sequences(X, maxlen=length, padding='post')
        return X

# one hot encode target sequence
def encode_output(sequences, vocab_size):
        ylist = list()
        for sequence in sequences:
                encoded = to_categorical(sequence, num_classes=vocab_size)
                ylist.append(encoded)
        y = array(ylist)
        y = y.reshape(sequences.shape[0], sequences.shape[1], vocab_size)
        return y
```

3 Define NMT Model

In [38]:
```
# define NMT model
def define_model(src_vocab, tar_vocab, source_steps, target_steps, embedding_dim):
        model = Sequential()
        model.add(Embedding(src_vocab, embedding_dim, input_length=source_steps, mask_z
        model.add(LSTM(embedding_dim))
        model.add(RepeatVector(target_steps))
        model.add(LSTM(embedding_dim, return_sequences=True))
        model.add(TimeDistributed(Dense(tar_vocab, activation='softmax')))
        # compile model
        model.compile(optimizer='adam', loss='categorical_crossentropy')
        # summarize defined model
        model.summary()
        #plot_model(model, to_file='model.png', show_shapes=True)
```

```
    return model
```

```
In [39]: # load datasets
         dataset = load_clean_sentences('english-german-both.pkl')
         train = load_clean_sentences('english-german-train.pkl')
         test = load_clean_sentences('english-german-test.pkl')
         # prepare english tokenizer
         eng_tokenizer = create_tokenizer(dataset[:, 0])
         eng_vocab_size = len(eng_tokenizer.word_index) + 1
         eng_length = max_length(dataset[:, 0])
         print('English Vocabulary Size: %d' % eng_vocab_size)
         print('English Max Length: %d' % (eng_length))
```

```
English Vocabulary Size: 2317
English Max Length: 5
```

```
In [40]: # prepare german tokenizer
         ger_tokenizer = create_tokenizer(dataset[:, 1])
         ger_vocab_size = len(ger_tokenizer.word_index) + 1
         ger_length = max_length(dataset[:, 1])
         print('German Vocabulary Size: %d' % ger_vocab_size)
         print('German Max Length: %d' % (ger_length))
```

```
German Vocabulary Size: 3686
German Max Length: 10
```

```
In [41]: # prepare training data
         trainX = encode_sequences(ger_tokenizer, ger_length, train[:, 1])
         trainY = encode_sequences(eng_tokenizer, eng_length, train[:, 0])
         trainY = encode_output(trainY, eng_vocab_size)
         # prepare validation data
         testX = encode_sequences(ger_tokenizer, ger_length, test[:, 1])
         testY = encode_sequences(eng_tokenizer, eng_length, test[:, 0])
         testY = encode_output(testY, eng_vocab_size)
         # define model
         model = define_model(ger_vocab_size, eng_vocab_size, ger_length, eng_length, 256)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 10, 256)           943616
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               525312
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 5, 256)            0
_________________________________________________________________
lstm_2 (LSTM)                (None, 5, 256)            525312
```

```
_________________________________________________________________
time_distributed_1 (TimeDist (None, 5, 2317)           595469
=================================================================
Total params: 2,589,709
Trainable params: 2,589,709
Non-trainable params: 0
_________________________________________________________________
```

4 Train the model

```
In [42]: # fit model
         checkpoint = ModelCheckpoint('model.h5', monitor='val_loss', verbose=1, save_best_only=
         model.fit(trainX, trainY, epochs=30, batch_size=64, validation_data=(testX, testY), cal

Train on 9000 samples, validate on 1000 samples
Epoch 1/30
 - 16s - loss: 4.3666 - val_loss: 3.5597

Epoch 00001: val_loss improved from inf to 3.55970, saving model to model.h5
Epoch 2/30
 - 10s - loss: 3.4330 - val_loss: 3.4158

Epoch 00002: val_loss improved from 3.55970 to 3.41580, saving model to model.h5
Epoch 3/30
 - 11s - loss: 3.2881 - val_loss: 3.3521

Epoch 00003: val_loss improved from 3.41580 to 3.35209, saving model to model.h5
Epoch 4/30
 - 11s - loss: 3.1792 - val_loss: 3.2426

Epoch 00004: val_loss improved from 3.35209 to 3.24257, saving model to model.h5
Epoch 5/30
 - 11s - loss: 3.0343 - val_loss: 3.1428

Epoch 00005: val_loss improved from 3.24257 to 3.14276, saving model to model.h5
Epoch 6/30
 - 11s - loss: 2.8835 - val_loss: 3.0081

Epoch 00006: val_loss improved from 3.14276 to 3.00810, saving model to model.h5
Epoch 7/30
 - 11s - loss: 2.7154 - val_loss: 2.8880

Epoch 00007: val_loss improved from 3.00810 to 2.88804, saving model to model.h5
Epoch 8/30
 - 11s - loss: 2.5474 - val_loss: 2.7684
```

```
Epoch 00008: val_loss improved from 2.88804 to 2.76843, saving model to model.h5
Epoch 9/30
 - 15s - loss: 2.3860 - val_loss: 2.6643

Epoch 00009: val_loss improved from 2.76843 to 2.66427, saving model to model.h5
Epoch 10/30
 - 15s - loss: 2.2293 - val_loss: 2.5695

Epoch 00010: val_loss improved from 2.66427 to 2.56952, saving model to model.h5
Epoch 11/30
 - 13s - loss: 2.0832 - val_loss: 2.4855

Epoch 00011: val_loss improved from 2.56952 to 2.48554, saving model to model.h5
Epoch 12/30
 - 17s - loss: 1.9470 - val_loss: 2.4230

Epoch 00012: val_loss improved from 2.48554 to 2.42302, saving model to model.h5
Epoch 13/30
 - 14s - loss: 1.8181 - val_loss: 2.3585

Epoch 00013: val_loss improved from 2.42302 to 2.35853, saving model to model.h5
Epoch 14/30
 - 13s - loss: 1.6959 - val_loss: 2.3122

Epoch 00014: val_loss improved from 2.35853 to 2.31218, saving model to model.h5
Epoch 15/30
 - 11s - loss: 1.5807 - val_loss: 2.2679

Epoch 00015: val_loss improved from 2.31218 to 2.26789, saving model to model.h5
Epoch 16/30
 - 11s - loss: 1.4708 - val_loss: 2.2227

Epoch 00016: val_loss improved from 2.26789 to 2.22267, saving model to model.h5
Epoch 17/30
 - 16s - loss: 1.3646 - val_loss: 2.1861

Epoch 00017: val_loss improved from 2.22267 to 2.18608, saving model to model.h5
Epoch 18/30
 - 12s - loss: 1.2653 - val_loss: 2.1491

Epoch 00018: val_loss improved from 2.18608 to 2.14908, saving model to model.h5
Epoch 19/30
 - 11s - loss: 1.1677 - val_loss: 2.1184

Epoch 00019: val_loss improved from 2.14908 to 2.11838, saving model to model.h5
Epoch 20/30
 - 11s - loss: 1.0743 - val_loss: 2.0839
```

```
Epoch 00020: val_loss improved from 2.11838 to 2.08386, saving model to model.h5
Epoch 21/30
 - 11s - loss: 0.9894 - val_loss: 2.0613

Epoch 00021: val_loss improved from 2.08386 to 2.06132, saving model to model.h5
Epoch 22/30
 - 12s - loss: 0.9096 - val_loss: 2.0355

Epoch 00022: val_loss improved from 2.06132 to 2.03552, saving model to model.h5
Epoch 23/30
 - 10s - loss: 0.8329 - val_loss: 2.0312

Epoch 00023: val_loss improved from 2.03552 to 2.03116, saving model to model.h5
Epoch 24/30
 - 12s - loss: 0.7595 - val_loss: 1.9955

Epoch 00024: val_loss improved from 2.03116 to 1.99546, saving model to model.h5
Epoch 25/30
 - 12s - loss: 0.6908 - val_loss: 2.0000

Epoch 00025: val_loss did not improve
Epoch 26/30
 - 13s - loss: 0.6307 - val_loss: 1.9834

Epoch 00026: val_loss improved from 1.99546 to 1.98342, saving model to model.h5
Epoch 27/30
 - 11s - loss: 0.5728 - val_loss: 1.9715

Epoch 00027: val_loss improved from 1.98342 to 1.97146, saving model to model.h5
Epoch 28/30
 - 11s - loss: 0.5209 - val_loss: 1.9705

Epoch 00028: val_loss improved from 1.97146 to 1.97048, saving model to model.h5
Epoch 29/30
 - 11s - loss: 0.4765 - val_loss: 1.9640

Epoch 00029: val_loss improved from 1.97048 to 1.96400, saving model to model.h5
Epoch 30/30
 - 11s - loss: 0.4340 - val_loss: 1.9728

Epoch 00030: val_loss did not improve


Out[42]: <keras.callbacks.History at 0x1fd14195b38>

In [43]: from pickle import load
         from numpy import argmax
         from keras.preprocessing.text import Tokenizer
```

```
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu

# load a clean dataset
def load_clean_sentences(filename):
        return load(open(filename, 'rb'))

# fit a tokenizer
def create_tokenizer(lines):
        tokenizer = Tokenizer()
        tokenizer.fit_on_texts(lines)
        return tokenizer

# max sentence length
def max_length(lines):
        return max(len(line.split()) for line in lines)

# encode and pad sequences
def encode_sequences(tokenizer, length, lines):
        # integer encode sequences
        X = tokenizer.texts_to_sequences(lines)
        # pad sequences with 0 values
        X = pad_sequences(X, maxlen=length, padding='post')
        return X

# map an integer to a word
def word_for_id(integer, tokenizer):
        for word, index in tokenizer.word_index.items():
                if index == integer:
                        return word
        return None
```

In [44]:
```
# generate target given source sequence
def predict_sequence(model, tokenizer, source):
        prediction = model.predict(source, verbose=0)[0]
        integers = [argmax(vector) for vector in prediction]
        target = list()
        for i in integers:
                word = word_for_id(i, tokenizer)
                if word is None:
                        break
                target.append(word)
        return ' '.join(target)
```

In [45]:
```
# evaluate the skill of the model
def evaluate_model(model, sources, raw_dataset):
        actual, predicted = list(), list()
```

```
                for i, source in enumerate(sources):
                        # translate encoded source text
                        source = source.reshape((1, source.shape[0]))
                        translation = predict_sequence(model, eng_tokenizer, source)
                        raw_target, raw_src = raw_dataset[i]
                        if i < 10:
                                print('src=[%s], target=[%s], predicted=[%s]' % (raw_src, raw_t
                        actual.append(raw_target.split())
                        predicted.append(translation.split())
                # calculate BLEU score
                print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
                print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
                print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0))
                print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25,
```

```
In [46]: # load datasets
         dataset = load_clean_sentences('english-german-both.pkl')
         train = load_clean_sentences('english-german-train.pkl')
         test = load_clean_sentences('english-german-test.pkl')
         # prepare english tokenizer
         eng_tokenizer = create_tokenizer(dataset[:, 0])
         eng_vocab_size = len(eng_tokenizer.word_index) + 1
         eng_length = max_length(dataset[:, 0])
         # prepare german tokenizer
         ger_tokenizer = create_tokenizer(dataset[:, 1])
         ger_vocab_size = len(ger_tokenizer.word_index) + 1
         ger_length = max_length(dataset[:, 1])
         # prepare data
         trainX = encode_sequences(ger_tokenizer, ger_length, train[:, 1])
         testX = encode_sequences(ger_tokenizer, ger_length, test[:, 1])
         # load model

In [47]: model = load_model('model.h5')
         # test on some training sequences
         print('train')
         evaluate_model(model, trainX, train)
         # test on some test sequences
         print('test')
         evaluate_model(model, testX, test)
```

```
train
src=[du bist gefeuert], target=[youre fired], predicted=[youre fired]
src=[strengt euch mehr an], target=[try harder], predicted=[try harder]
src=[ich unterstutzte ihn], target=[i supported him], predicted=[i supported him]
src=[tom hat nachgegeben], target=[tom relented], predicted=[tom relented]
src=[es ist nicht da], target=[its not there], predicted=[its isnt easy]
src=[na und], target=[then what], predicted=[then what]
src=[ich bin geduldig], target=[im patient], predicted=[im patient]
```

```
src=[die tasse ist voll], target=[the cup is full], predicted=[the cup is full]
src=[es war groartig], target=[it was superb], predicted=[it was superb]
src=[lest das zuerst], target=[read this first], predicted=[read this first]


c:\programs\python\python36\lib\site-packages\nltk\translate\bleu_score.py:523: UserWarning:
The hypothesis contains 0 counts of 2-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
c:\programs\python\python36\lib\site-packages\nltk\translate\bleu_score.py:523: UserWarning:
The hypothesis contains 0 counts of 3-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)
c:\programs\python\python36\lib\site-packages\nltk\translate\bleu_score.py:523: UserWarning:
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)


BLEU-1: 0.074982
BLEU-2: 0.000000
BLEU-3: 0.000000
BLEU-4: 0.000000
test
src=[sind sie fertig], target=[are you ready], predicted=[are you ready]
src=[wir versuchten es], target=[we tried], predicted=[we will]
src=[tom ist da], target=[tom is here], predicted=[tom is there]
src=[ich fuhlte mich nackt], target=[i felt naked], predicted=[i felt isolated]
src=[hort auf zu schieen], target=[stop shooting], predicted=[stop shooting]
src=[wer wird das essen zubereiten], target=[wholl cook], predicted=[lets take eat]
src=[ich habe kopfweh], target=[my head aches], predicted=[im head aches]
src=[er hatte gluck], target=[he was lucky], predicted=[he was stupid]
src=[versuchen sies doch mal], target=[give it a shot], predicted=[give and see it]
src=[tom war ein spieer], target=[tom was square], predicted=[tom was a]
BLEU-1: 0.063465
BLEU-2: 0.000000
BLEU-3: 0.000000
BLEU-4: 0.000000
```