Introduction

Welcome to the Summarization Notebook.

In this notebook, we are going to train a neural network to summarize news articles. Our neural network is going to learn from example, as we provide you with (article, summary) pairs. We play with a **toy dataset** made of only articles about police related news. Usual datasets can be 20x larger in size, but we have reduced it for computational purposes.

We will do this using a Transformer network, from the [Attention is all you need (http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf)](http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf) paper. In this notebook you will:

- Process text into sub-word tokens, to avoid fixed vocabulary sizes, and UNK tokens.
- Use a Transformer to read a news article, and produce a summary.
- Perform operations on learned word-vectors to examine what the model has learned.

Before you start

You should read the Attention is all you need paper.

All dataset files should be placed in the `dataset/` folder of this assignment.

Library imports

In [1]:
```
%load_ext autoreload
%autoreload 2
```

In [2]:
```
from transformer import Transformer
import sentencepiece as spm
import tensorflow as tf
import numpy as np
import json
import capita

root_folder = ""
```

In [3]:
```
# Load the word piece model that will be used to tokenize the texts into
# word pieces with a vocabulary size of 10000

sp = spm.SentencePieceProcessor()
sp.Load(root_folder+"dataset/wp_vocab10000.model")

vocab = [line.split('\t')[0] for line in open(root_folder+"dataset/wp_vocab10000.vocab", "r")]
pad_index = vocab.index('#')

def pad_sequence(numerized, pad_index, to_length):
    pad = numerized[:to_length]
    padded = pad + [pad_index] * (to_length - len(pad))
    mask = [w != pad_index for w in padded]
    return padded, mask
```

Building blocks of a Transformer

The Transformer is split into 3 files: transformer_attention.py, transformer_layers.py and transformer.py

Our Transformer is built as a Keras model

(1) Implementing the Query-Key-Value Attention (AttentionQKV)

This part is located in AttentionQKV in transformer_attention.py.

In [ ]:
```

```

(2) Implementing Multi-head attention

This part is located in the class MultiHeadProjection in transformer_attention.py.

In [ ]:
```

```

(3) Position Embedding

We implement the PositionEmbedding class in transformer.py.

In [ ]:

```

```

(4) Transformer Encoder / Transformer Decoder

We make 2 classes in the transformer.py file: TransformerEncoderBlock, TransformerDecoderBlock.

The code below will verify the accuracy of each block

In [ ]:

```

```

In [ ]:

```

```

(5) Transformer

This is the final high-level function that pieces it all together.

We implement the call function of the Transformer class in the `transformer.py` file.

In [ ]:

```

```

Creating a Transformer

Now that all the blocks of the Transformer are implemented, we can create a full model with placeholders and a loss.

In [102]:
```
class TransformerTrainer():

    def __init__(self, vocab_size, d_model, input_length, output_length, n_layers, d_filter, learning_rate=2e-3):

        self.source_sequence = tf.placeholder(tf.int32,shape=(None,input_length), name="source_sequence")
        self.target_sequence = tf.placeholder(tf.int32, shape=(None,output_length),name="target_sequence")
        self.encoder_mask = tf.placeholder(tf.bool,shape=(None,input_length),name="encoder_mask")
        self.decoder_mask = tf.placeholder(tf.bool, shape=(None,output_length),name="decoder_mask")

        self.model = Transformer(vocab_size=vocab_size, d_model=d_model, n_layers=n_layers, d_filter=d_filter)

        self.decoded_logits = self.model(self.source_sequence, self.target_sequence, encoder_mask=self.encoder_mask, decoder_mask=self.decoder_mask)
        self.global_step = tf.train.get_or_create_global_step()

        # Summarization loss
        self.loss = tf.losses.sparse_softmax_cross_entropy(self.target_sequence, self.decoded_logits, tf.cast(self.decoder_mask, tf.float32))
        sched=tf.train.exponential_decay(learning_rate,self.global_step, 500,.7)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=sched)
        self.train_op = self.optimizer.minimize(self.loss, global_step=self.global_step)
        self.saver = tf.train.Saver()
```

We now instantiate the Transformer with our sets of hyperparameters specific to the task of summarization. In summarization, we are going to go from documents with up to 400 words, to documents with up to 100 words. We are using WordPieces (http://aclweb.org/anthology/P18-1007).

In [103]:
```
# Dataset related parameters
vocab_size = len(vocab)
ilength = 400 # Length of the article
olength  = 100 # Length of the summaries

n_layers = 6
d_model  = 104
d_filter = 416

model = TransformerTrainer(vocab_size, d_model, ilength, olength, n_layers, d_filter)
```

Training the model

In [12]:
```
with open(root_folder+"dataset/summarization_dataset_preprocessed.json",
"r") as f:

    dataset = json.load(f)


d_train = [d for d in dataset if d['cut'] == 'training']
d_valid = [d for d in dataset if d['cut'] == 'evaluation']

len(d_train), len(d_valid)
```

Out[12]:
```
(61055, 1558)
```

In [13]:
```
# An example (article, summary) pair in the training data:

print(d_train[145]['story'])
print("=======================\n=======================")
print(d_train[145]['summary'])
```

```
Tbilisi, Georgia (CNN)Police have shot and killed a white tiger that ki
lled a man Wednesday in Tbilisi, Georgia, a Ministry of Internal Affair
s representative said, after severe flooding allowed hundreds of wild a
nimals to escape the city zoo.
The tiger attack happened at a warehouse in the city center. The animal
had been unaccounted for since the weekend floods destroyed the zoo pre
mises.
The man killed, who was 43, worked in a company based in the warehouse,
the Ministry of Internal Affairs said. Doctors said he was attacked in
the throat and died before reaching the hospital.
Experts are still searching the warehouse, the ministry said, adding th
at earlier reports that the tiger had injured a second man were unfound
ed.
The zoo administration said Wednesday that another tiger was still miss
ing. It was unable to confirm if the creature was dead or had escaped a
live.
Georgian Prime Minister Irakli Garibashvili apologized to the public, s
aying he had been misinformed by the zoo's management when he'd previou
sly said there were no more dangerous animals on the run.
City residents were urged to stay indoors for their own safety in the i
mmediate aftermath of the floods. Volunteers have since been helping ci
ty workers with the cleanup operation.
At least 19 people died in the flooding, according to Civil Georgia, a
news website run by the nongovernmental organization United Nations Ass
ociation of Georgia. Six more remained missing, it said Tuesday, citing
the State Security and Crisis Management Council.
Meanwhile, the zoo lost about half of its 600 animals, including lions,
tigers, bears and wolves, in the natural disaster.
Some animals have since been recaptured, Civil Georgia reported. Others
died in the floods or have been killed by police as they scour the stre
ets for escapees.
Russian state news outlet RT.com  that an African penguin had made it 6
0 kilometers (37 miles) downriver from Tbilisi before being caught aliv
e in a dragnet on the border with Azerbaijan.
Video from the city showed a large crocodile being restrained by rescue
rs, as well as a hippopotamus standing in floodwaters, looking confuse
d.
The latter was eventually cornered in a city square before being tranqu
ilized and recaptured.
One terrified bear escaped the flood by perching on a window ledge.
Video footage also showed devastation across swaths of the Georgian cap
ital, where flash floods swept away roads, at least one house and many
trees. The corpses of dead animals could be seen amid the wreckage.
The problems began before midnight Saturday when heavy rainfall turned
the Vere River, usually little more than a stream through the center of
Tbilisi, into a raging torrent, according to Civil Georgia.
Images on Tbilisi City Hall's Facebook page showed roads washed out, hi
llsides collapsed and vehicles tossed about like toys. Rescue workers c
arried people on their shoulders through waist-high water.
Garibashvili extended his condolences Tuesday to the families of those
killed in the flooding.
He also proposed the creation of a park in the zoo premises to honor th
ose lost. "It will be a park of solidarity, a symbol of our unity, self
lessness, and mutual support," he said in a statement on his website.
President Georgi Margvelashvili earlier said the capital's mayoral offi
ce would help those who had lost out financially as a result of the flo
ods.
```

```
"The situation is difficult, but it can be handled except for the fact
that we cannot bring back those who died," he said.
According to the World Wildlife Fund, as few as 3,200 tigers exist in t
he wild today.

Journalist Eka Kadagishvili reported from Tbilisi, and Laura Smith-Spar
k wrote from London. CNN's Kimberly Hutcherson contributed to this repo
rt.
=======================
=======================
Police have shot dead a tiger that killed a man in Tbilisi, Georgia, a
government official says, after zoo animals escaped in weekend floodin
g.
```

Similarly to the previous notebook, we create a function to get a random batch to train on, given a dataset.

In [14]:
```python
def build_batch(dataset, batch_size):
    indices = list(np.random.randint(0, len(dataset), size=batch_size))

    batch = [dataset[i] for i in indices]
    batch_input = np.array([a['input'] for a in batch])
    batch_input_mask = np.array([a['input_mask'] for a in batch])
    batch_output = np.array([a['output'] for a in batch])
    batch_output_mask = np.array([a['output_mask'] for a in batch])

    return batch_input, batch_input_mask, batch_output, batch_output_mas
k
```

In [135]:
```
batch_size=3

with tf.Session() as sess:
    # This is how you randomly initialize the Transformer weights.
    sess.run(tf.global_variables_initializer())
    #model.saver.restore(sess, root_folder+"models/final_transformer_sum
marization")

    for e in range(5000):
    # Create a random mini-batch from the training dataset
        batch_input, batch_input_mask, batch_output, batch_output_mask =
build_batch(d_train, batch_size)
        # Build the feed-dict connecting placeholders and mini-batch
        feed = {model.source_sequence: batch_input, model.target_sequenc
e: batch_output,
                                              model.encoder_mask: batch_input_mask,
model.decoder_mask: batch_output_mask,}

        # Obtain the loss. Be careful when you use the train_op and not,
as previously.
        train_loss, _, step = sess.run([model.loss, model.train_op, mode
l.global_step], feed_dict=feed)
        if e%500==0:
            print(step)
            batch_size+=1
            print(train_loss)
    # This is how you save model weights into a file
    model.saver.save(sess, root_folder+"models/final_transformer_summari
zation")
    # This is how you restore a model previously saved
    #model.saver.restore(sess, root_folder+"models/transformer_summarize
r")
```

```
1
9.834685
501
6.0821543
1001
5.7184105
1501
5.3763294
2001
5.069806
2501
5.3976316
3001
5.4380155
3501
5.2949867
4001
5.35953
4501
5.325645
```

Using the Summarization model

Now that you have trained a Transformer to perform Summarization, we will use the model on news articles from the wild.

The three subsections below explore what the model has learned.

In [136]:
```
# Put the file path to your best performing model in the string below.
model_file = root_folder+"models/final_transformer_summarization"
#model_file = root_folder+"models/transformer_unicorn_summarizer"
#model_file = root_folder+"models/transformer_summarizer"
```

The validation loss

Measure the validation loss of your model. This part could be used, as in our previous notebook, in deciding what is a likely, vs. unlikely summary for an article.

In [ ]:
```

```

Generating an article's summary

This model we have built is meant to be used to generate summaries for new articles we do not have summaries for. We got a news article (https://www.chicagotribune.com/news/local/breaking/ct-met-officer-shot-20190309-story.html) from the Chicago Tribune about a police shooting, and want to use our model to produce a summary.

As you will see, our model is still limited in its ability, and will most likely not produce a perfect summary, however, with more data and training, this model would be able to produce good summaries.

In [113]:
```
article_text = "A 34-year-old Chicago police officer has been shot in th
e shoulder during the execution of a search warrant in the Humboldt Park
neighborhood, police say. The alleged shooter, a 19-year-old woman, was
 in custody. The shooting happened about 7:20 p.m. in the 2700 block of
 West Potomac Avenue, police said. The officer, part of the Grand Centra
l District tactical unit, was taken to Stroger Hospital. While officers
 were serving a \"typical\" search warrant for \"narcotics and illegal w
eapons\" and were attempting to reach a rear door, \"a shot was fired,\"
striking the tactical officer in the shoulder, said Chicago police Super
intendent Eddie Johnson during a news briefing outside the hospital. He
 said the officer, who has about four or five years on the job, was \"st
able\" but in critical condition. \"His family is here,\" Johnson said.
\"He's talking a lot and just wants the ordeal to be over.\" He said thi
s incident serves as just another reminder of how dangerous a police off
icer's job is. At the scene of the shooting, crime tape closed Potomac f
rom Washtenaw Avenue to California Avenue and encompassed the alley west
of the brick apartment building, south of Potomac. Dozens of officers st
ood in the alley, while even more walked up and down the street. Neighbo
rs gathered at the edge of the yellow tape on the sidewalk along Califor
nia and watched them work. Standing next to a man, a woman talked to pol
ice in the crime scene, across the street. \"We're not under arrest? We
 can go?\" the woman checked with officers. They told her she could go,
 and she and the man walked underneath the yellow tape and out of the cr
ime scene."
input_length = 400
output_length = 100

# Process the capitalization with the preprocess_capitalization of the c
apita package.
article_text = capita.preprocess_capitalization(article_text)

# Numerize the tokens of the processed text using the loaded sentencepie
ce model.
numerized = sp.EncodeAsIds(article_text)
# Pad the sequence and keep the mask of the input
padded, mask = pad_sequence(numerized, len(article_text), input_length)

# Making the news article into a batch of size one, to be fed to the neu
ral network.
encoder_input = np.array([padded])
encoder_mask = np.array([mask])

with tf.Session() as sess:
    model.saver.restore(sess, model_file)

    decoded_so_far = [0]

    for j in range(output_length):
        padded_decoder_input, decoder_mask = pad_sequence(decoded_so_far
, pad_index, output_length)
        padded_decoder_input = [padded_decoder_input]
        decoder_mask = [decoder_mask]
        #print("=======================")
        #print(padded_decoder_input)
        # Use the model to find the distrbution over the vocabulary for
 the next word
```

```
        feed = {model.source_sequence: e_input, model.target_sequence: e
_output,
                                      model.encoder_mask: e_input_mask,
model.decoder_mask: e_output_mask}
        logits = sess.run(model.decoded_logits, feed_dict=feed)

        chosen_words =np.argmax(logits[0,j])
        print(chosen_words)# Take the argmax, getting the most likely ne
xt word
        decoded_so_far.append(int(chosen_words)) # We add it to the summ
ary so far


print("The final summary:")
print("".join([vocab[i] for i in decoded_so_far]).replace("_", " "))
```

```
INFO:tensorflow:Restoring parameters from models/final_transformer_summ
arization
3
4
5
3
127
16
130
130
7
61
863
12
10
31
10
3
6
3
4
3
166
7
3
107
107
7
19
19
6
107
9
107
6
3
6
6
3
4
10
28
3
127
19
9
194
5
3
20
5
55
3
6
5
6
```

```
6
19
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-113-628029f2d21d> in <module>()
     29         feed = {model.source_sequence: e_input, model.target_sequence: e_output,
     30                                         model.encoder_mask: e_input_mask, model.decoder_mask: e_output_mask}
---> 31         logits = sess.run(model.decoded_logits, feed_dict=feed)
     32 
     33         chosen_words =np.argmax(logits[0,j])

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/tensorflow/python/client/session.py in run(self, fetches, feed_dict, options, run_metadata)
    927     try:
    928       result = self._run(None, fetches, feed_dict, options_ptr,
--> 929                          run_metadata_ptr)
    930       if run_metadata:
    931         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/tensorflow/python/client/session.py in _run(self, handle, fetches, feed_dict, options, run_metadata)
   1150     if final_fetches or final_targets or (handle and feed_dict_tensor):
   1151       results = self._do_run(handle, final_targets, final_fetches,
-> 1152                              feed_dict_tensor, options, run_metadata)
   1153     else:
   1154       results = []

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/tensorflow/python/client/session.py in _do_run(self, handle, target_list, fetch_list, feed_dict, options, run_metadata)
   1326     if handle is None:
   1327       return self._do_call(_run_fn, feeds, fetches, targets, options,
-> 1328                            run_metadata)
   1329     else:
   1330       return self._do_call(_prun_fn, handle, feeds, fetches)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/tensorflow/python/client/session.py in _do_call(self, fn, *args)
   1332   def _do_call(self, fn, *args):
   1333     try:
-> 1334       return fn(*args)
   1335     except errors.OpError as e:
   1336       message = compat.as_text(e.message)

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/tensorflow/python/client/session.py in _run_fn(feed_dict, fetch_list, target_list, options, run_metadata)
   1317       self._extend_graph()
   1318       return self._call_tf_sessionrun(
```

```
-> 1319            options, feed_dict, fetch_list, target_list, run_meta
data)
   1320
   1321       def _prun_fn(handle, feed_dict, fetch_list):

/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
ckages/tensorflow/python/client/session.py in _call_tf_sessionrun(self,
options, feed_dict, fetch_list, target_list, run_metadata)
   1405       return tf_session.TF_SessionRun_wrapper(
   1406           self._session, options, feed_dict, fetch_list, target_l
ist,
-> 1407           run_metadata)
   1408
   1409     def _call_tf_sessionprun(self, handle, feed_dict, fetch_list)
:

KeyboardInterrupt:
```

Word vectors

The model we train learns word representations for each word in our vocabulary. A word represention is a vector of **dim** size.

It is common in NLP to inspect the word vectors, as some properties of language often appear in the embedding structure.

We are going to load the word embeddings learned by our model, and inspect it. Because our network was not trained for long, we are going for the simplest patterns, but if we let the network train longer, it learns more complex, semantic patterns.

In [114]:

```
# We help you load the matrix, as it is hidden within the Transformer st
ructure.

with tf.Session() as sess:
    model.saver.restore(sess, model_file)
    E = sess.run(model.model.encoder.embedding_layer.embedding.embedding
s)

print("The embedding matrix has shape:", E.shape)
print("The vocabulary has length:", len(vocab))
```

```
INFO:tensorflow:Restoring parameters from models/final_transformer_summ
arization
The embedding matrix has shape: (10000, 104)
The vocabulary has length: 10000
```

In [134]:
```
def cosine_sim(v1, v2):
    a = tf.placeholder(tf.float32, shape=[None], name="input_placeholder
_a")
    b = tf.placeholder(tf.float32, shape=[None], name="input_placeholder
_b")
    normalize_a = tf.nn.l2_normalize(a,0)
    normalize_b = tf.nn.l2_normalize(b,0)
    cos_similarity=tf.reduce_sum(tf.multiply(normalize_a,normalize_b))
    sess=tf.Session()
    cos_sim=sess.run(cos_similarity,feed_dict={a:v1,b:v2})
    return cos_sim

for w1, w2 in [("she", "he"), ("more", "less"), ("she", "ball"), ("more"
, "gorilla")]:
    w1_index = [vocab.index('_'+w1)] # The index of the first  word in o
ur vocabulary
    w2_index = [vocab.index('_'+w2)] # The index of the second word in o
ur vocabulary
    w1_vec=E[w1_index][0]

    w2_vec=E[w2_index][0]
     # Get the embedding vector of the second word

    print(w1," vs. ", w2, "similarity:",cosine_sim(w1_vec, w2_vec))
```

```
she  vs.  he similarity: 0.9314363
more  vs.  less similarity: 0.76642716
she  vs.  ball similarity: 0.26654655
more  vs.  gorilla similarity: -0.08681743
```

In [ ]: