Jonathan Pedoeem HW5 ECE 471

Professor Curro

October 9th, 2018

In [0]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tqdm import tqdm
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
```

In [0]:
```
LONGEST_ARTICLE = 190
NUM_CLASSES = 4
lr = 1e-4
```

In [0]:
```
#preprocess the data
def pre_data(filename):
    train_sentences = []
    train_y = []
    with open(filename, "r") as data:
        line = data.readline()
        while line:
            y,title,body = line.split("\",\"")
            y = int(y[1:])-1
            text ="{} {}".format(title,body).split(" ")
            text = [stemmer.stem(t.strip()) for t in text]
            train_sentences.append(text)
            train_y.append(y)
            line = data.readline()
    return train_sentences, train_y

#know longest article is 197 words and that there are 141,206
words
```

In [0]:
```
train_sentences, y = pre_data("train.csv")
```

In [0]:
```
train_sentencesNP = np.array(train_sentences)
yC = keras.utils.to_categorical(y)
randos = np.random.choice(len(train_sentencesNP),len(train_se
ntencesNP),replace=False)
validation_randos = randos[:20000]
train_randos = randos[20000:]
train_x = train_sentencesNP[train_randos]
train_y = yC[train_randos]
val_x = train_sentencesNP[validation_randos]
val_y = yC[validation_randos]
```

In [0]:
```
train_x = np.append(train_x, "ENDOFSENTENCETOKEN")
```

In [0]:
```
t = keras.preprocessing.text.Tokenizer()
t.fit_on_texts(train_x)
```

In [0]:
```
train_sequences = t.texts_to_sequences(train_x)
validation_sequences = t.texts_to_sequences(val_x)
```

In [0]:
```
EOS_TOKEN = train_sequences.pop()[0]
```

In [0]:
```
def add_padding(sequences, EOS_TOKEN, MAX_LENGTH):
    padded_sequences = np.zeros((len(sequences),MAX_LENGTH))
    for seq in tqdm(range(len(sequences))):
        len_seq = len(sequences[seq])
        if len_seq>MAX_LENGTH:
            padded_sequences[seq] =  sequences[seq][:MAX_LENG
TH]
        else:
            padded_sequences[seq] =  sequences[seq] + (MAX_LE
NGTH - len_seq )*[EOS_TOKEN]
    return padded_sequences
```

In [47]:
```
padded_sequences = add_padding(train_sequences,EOS_TOKEN,LONG
EST_ARTICLE)
```

```
100%|██████████| 100000/100000 [00:01<00:00, 53531.04it/s]
```

In [48]:
```
padded_val_sequences = add_padding(validation_sequences, EOS_
TOKEN,LONGEST_ARTICLE)
```

```
100%|██████████| 20000/20000 [00:00<00:00, 51193.41it/s]
```

In [0]:
```
##THIS IS A LIST OF OTHER MODELS THAT DID JUST A TAD BETTER T
HAN THE SMALL LIST, BUT WITH MANY MORE PARAMETERS
#with out any convolutions get some good results, after 6 epo
chs get 91.6% on val set.
# model = keras.Sequential()
# model.add(keras.layers.Embedding(len(t.word_index)+1,512, i
nput_length=LONGEST_ARTICLE))
# model.add(keras.layers.Flatten())
# model.add(keras.layers.Dense(NUM_CLASSES,activation="softma
x"))
# model.compile(optimizer=keras.optimizers.Adam(lr),
#               loss='categorical_crossentropy',
#               metrics=['accuracy'])
```

In [0]:
```
#This is also pretty good get, after 6 epochs get 91%
# model = keras.Sequential()
# model.add(keras.layers.Embedding(len(t.word_index)+1,256, i
nput_length=LONGEST_ARTICLE))
# model.add(keras.layers.Conv1D(filters=32,kernel_size=4, dil
ation_rate=8, padding='valid',activation='elu'))
# model.add(keras.layers.Flatten())
# model.add(keras.layers.Dense(NUM_CLASSES,activation="softma
x"))
# model.compile(optimizer=keras.optimizers.Adam(lr),
#               loss='categorical_crossentropy',
#               metrics=['accuracy'])
```

In [0]:
```
#this is an attempt to make a really small model, after 32 ep
ochs get 90.8% on val.
model = keras.Sequential()
model.add(keras.layers.Embedding(len(t.word_index)+1,4, input
_length=LONGEST_ARTICLE))
model.add(keras.layers.Conv1D(filters=32,kernel_size=2, dilat
ion_rate=2, padding='valid',activation='relu'))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(NUM_CLASSES,activation="softmax"
))
model.compile(optimizer=keras.optimizers.Adam(lr),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

In [50]:
```
model.summary()
model.fit(padded_sequences,train_y, epochs = 32, batch_size =
512, validation_data=(padded_val_sequences,val_y),verbose=1)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 190, 4)            510492
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 188, 32)           288
_________________________________________________________________
flatten_9 (Flatten)          (None, 6016)              0
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 24068
=================================================================
Total params: 534,848
Trainable params: 534,848
Non-trainable params: 0
_________________________________________________________________

/usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/gradients_impl.py:108: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

```
Train on 100000 samples, validate on 20000 samples
Epoch 1/32
100000/100000 [==============================] - 3s 30us/step
- loss: 1.3823 - acc: 0.3040 - val_loss: 1.3698 - val_acc: 0.3
694
Epoch 2/32
100000/100000 [==============================] - 2s 23us/step
- loss: 1.2981 - acc: 0.5543 - val_loss: 1.1759 - val_acc: 0.6
733
Epoch 3/32
100000/100000 [==============================] - 2s 24us/step
- loss: 1.0067 - acc: 0.7320 - val_loss: 0.8458 - val_acc: 0.7
753
Epoch 4/32
100000/100000 [==============================] - 2s 23us/step
- loss: 0.7247 - acc: 0.8032 - val_loss: 0.6360 - val_acc: 0.8
208
Epoch 5/32
100000/100000 [==============================] - 2s 25us/step
- loss: 0.5610 - acc: 0.8422 - val_loss: 0.5186 - val_acc: 0.8
460
Epoch 6/32
100000/100000 [==============================] - 2s 24us/step
- loss: 0.4620 - acc: 0.8644 - val_loss: 0.4445 - val_acc: 0.8
639
...
Epoch 31/32
100000/100000 [==============================] - 2s 23us/step
- loss: 0.1064 - acc: 0.9676 - val_loss: 0.2573 - val_acc: 0.9
139
Epoch 32/32
100000/100000 [==============================] - 2s 23us/step
- loss: 0.1019 - acc: 0.9690 - val_loss: 0.2583 - val_acc: 0.9
152
```

Out[50]: <tensorflow.python.keras.callbacks.History at 0x7f1543d126a0>

In [0]:
```
def pre_data_test(filename):
    test_sentences = []
    test_y = []
    with open(filename, "r") as data:
        line = data.readline()
        while line:
            y,title,body = line.split("\",\"")
            y = int(y[1:])-1
            text ="{} {}".format(title,body).split(" ")
            text = [stemmer.stem(t.strip()) for t in text]
            test_sentences.append(text)
            test_y.append(y)
            line = data.readline()
    return test_sentences, test_y
```

In [54]:
```
test_x,test_y = pre_data_test("test.csv")
test_sequences = t.texts_to_sequences(test_x)
test_yC = keras.utils.to_categorical(test_y)
padded_test = add_padding(test_sequences,EOS_TOKEN,LONGEST_AR
TICLE)
```

```
100%|██████████| 7600/7600 [00:00<00:00, 52505.23it/s]
```

In [55]:
```
model.evaluate(padded_test, test_yC)
```

```
7600/7600 [==============================] - 1s 92us/step
```

Out[55]:
```
[0.25492076643987704, 0.9153947368421053]
```

Test Accuracy of Small Model : 91.5%