Deep Learning Course (980)

Assignment Three

Assignment Goals:

- Implementing RNN based language models.
- Implementing and applying a Recurrent Neural Network on text classification problem using TensorFlow.
- Implementing **many to one** and **many to many** RNN sequence processing.

In this assignment, you will implement RNN-based language models and compare extracted word representation from different models. You will also compare two different training methods for sequential data: Truncated Backpropagation Through Time **(TBTT)** and Backpropagation Through Time **(BTT)**. Also, you will be asked to apply Vanilla RNN to capture word representations and solve a text classification problem.

DataSets: You will use two datasets, an English Literature for language model task (part 1 to 4) and 20Newsgroups for text classification (part 5).

1. (30 points) Implement the RNN based language model described by Mikolov et al.[1], also called **Elman network** and train a language model on the English Literature dataset. This network contains input, hidden and output layer and is trained by standard backpropagation (TBTT with $\tau = 1$) using the cross-entropy loss.

   - The input represents the current word while using 1-of-N coding (thus its size is equal to the size of the vocabulary) and vector $s(t - 1)$ that represents output values in the hidden layer from the previous time step.
   - The hidden layer is a fully connected sigmoid layer with size 500.
   - Softmax Output Layer to capture a valid probability distribution.
   - The model is trained with truncated backpropagation through time (TBTT) with $\tau = 1$: the weights of the network are updated based on the error vector computed only for the current time step.

   Download the English Literature dataset and train the language model as described, report the model cross-entropy loss on the train set. Use nltk.word_tokenize to tokenize the documents. For initialization, $s(0)$ can be set to a vector of small values. Note that we are not interested in the *dynamic model* mentioned in the original paper. To make the implementation simpler you can use Keras to define neural net layers, including Keras.Embedding. (Keras.Embedding will create an additional mapping layer compared to the Elman architecture.)
2. (20 points) TBTT has less computational cost and memory needs in comparison with *backpropagation through time algorithm (BTT)*. These benefits come at the cost of losing long term dependencies [2]. Now let's try to investigate computational costs and performance of learning our language model with BTT. For training the Elman-type RNN with BTT, one option is to perform mini-batch gradient descent with exactly one sentence per mini-batch. (The input size will be [1, Sentence Length]).
   A. Split the document into sentences (you can use nltk.tokenize.sent_tokenize).
   B. For each sentence, perform one pass that computes the mean/sum loss for this sentence; then perform a gradient update for the whole sentence. (So the mini-batch size varies for the sentences with different lengths). You can truncate long sentences to fit the data in memory.
   C. Report the model cross-entropy loss.

3. (15 points) It does not seem that simple recurrent neural networks can capture truly exploit context information with long dependencies, because of the problem that gradients vanish and exploding. To solve this problem, gating mechanisms for recurrent neural networks were introduced. Try to learn your last model (Elman + BTT) with the SimpleRnn unit replaced with a Gated Recurrent Unit (GRU). Report the model cross-entropy loss. Compare your results in terms of cross-entropy loss with two other approach(part 1 and 2). Use each model to generate 10 synthetic sentences of 15 words each. Discuss the quality of the sentences generated - do they look like proper English? Do they match the training set? Text generation from a given language model can be done using the following iterative process:
   A. Set sequence = [first_word], chosen randomly.
   B. Select a new word based on the sequence so far, add this word to the sequence, and repeat. At each iteration, select the word with maximum probability given the sequence so far. The trained language model outputs this probability.
4. (15 points) The text describes how to extract a word representation from a trained RNN (Chapter 4). How we can evaluate the extracted word representation for your trained RNN? Compare the words representation extracted from each of the approaches using one of the existing methods.
5. (20 points) We are aiming to learn an RNN model that predicts document categories given its content (text classification). For this task, we will use the 20Newsgroupst dataset. The 20Newsgroupst contains messages from twenty newsgroups. We selected four major categories (comp, politics, rec, and religion) comprising around 13k documents altogether. Your model should learn word representations to support the classification task. For solving this problem modify the **Elman network** architecture such that the last layer is a softmax layer with just 4 output neurons (one for each category).
   A. Download the 20Newsgroups dataset, and use the implemented code from the notebook to read in the dataset.
   B. Split the data into a training set (90 percent) and validation set (10 percent). Train the model on 20Newsgroups.
   C. Report your accuracy results on the validation set.

NOTE: Please use Jupyter Notebook. The notebook should include the final code, results and your answers. You should submit your Notebook in (.pdf or .html) and .ipynb format. (penalty 10 points)

To reduce the parameters, you can merge all words that occur less often than a threshold into a special rare token (__unk__).

Instructions:

The university policy on academic dishonesty and plagiarism (cheating) will be taken very seriously in this course. Everything submitted should be your own writing or coding. You must not let other students copy your work. Spelling and grammar count.

Your assignments will be marked based on correctness, originality (the implementations and ideas are from yourself), clarification and test performance.

[1] Tom´ as Mikolov, Martin Kara ˇ fiat, Luk´ ´ as Burget, Jan ˇ Cernock´ ˇ y,Sanjeev Khudanpur: Recurrent neural network based language model, In: Proc. INTERSPEECH 2010

[2] Tallec, Corentin, and Yann Ollivier. "Unbiasing truncated backpropagation through time." arXiv preprint

In [1]:

```
"""This code is used to read all news and their labels"""
import os
import glob

def to_categories(name, cat=["politics","rec","comp","religion"]):
    for i in range(len(cat)):
        if str.find(name,cat[i])>-1:
            return(i)
    print("Unexpected folder: " + name) # print the folder name which does not include expected categories
    return("wth")

def data_loader(images_dir):
    categories = os.listdir(data_path)
    news = [] # news content
    groups = [] # category which it belong to

    for cat in categories:
        print("Category:"+cat)
        for the_new_path in glob.glob(data_path + '/' + cat + '/*'):
            news.append(open(the_new_path,encoding = "ISO-8859-1", mode ='r').read().lower())
            groups.append(cat)

    return news, list(map(to_categories, groups))



data_path = "datasets/20news_subsampled"
news, groups = data_loader(data_path)
```

```
Category:comp.graphics
Category:comp.os.ms-windows.misc
Category:comp.sys.ibm.pc.hardware
Category:comp.sys.mac.hardware
Category:comp.windows.x
Category:rec.autos
Category:rec.motorcycles
Category:rec.sport.baseball
Category:rec.sport.hockey
Category:soc.religion.christian
Category:talk.politics.guns
Category:talk.politics.mideast
Category:talk.politics.misc
Category:talk.religion.misc
```

In [2]:
```
'''Implementing RNN based language model Elman network. (part 1)
'''
from nltk import word_tokenize, download
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.python.client import device_lib
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.models import load_model
import numpy as np

# load the English Literature dataset
english_literature_path = './datasets/English Literature.txt'
with open(english_literature_path) as f:
    english_literature_text = f.read()
print(len(english_literature_text))

# tokenize the English Literature dataset
download('punkt')
english_literature_tokens = word_tokenize(english_literature_text)
print(len(english_literature_tokens))
```

```
C:\Users\songyih\AppData\Roaming\Python\Python37\site-packages\tensorflow\pyt
hon\framework\dtypes.py:526: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\songyih\AppData\Roaming\Python\Python37\site-packages\tensorflow\pyt
hon\framework\dtypes.py:527: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\songyih\AppData\Roaming\Python\Python37\site-packages\tensorflow\pyt
hon\framework\dtypes.py:528: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\songyih\AppData\Roaming\Python\Python37\site-packages\tensorflow\pyt
hon\framework\dtypes.py:529: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\songyih\AppData\Roaming\Python\Python37\site-packages\tensorflow\pyt
hon\framework\dtypes.py:530: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\songyih\AppData\Roaming\Python\Python37\site-packages\tensorflow\pyt
hon\framework\dtypes.py:535: FutureWarning: Passing (type, 1) or '1type' as a
synonym of type is deprecated; in a future version of numpy, it will be under
stood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
1115394
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\songyih\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
254533
```

In [23]:

```python
# build vocabulary
from collections import Counter

word2index = {}
index2word = []
english_literature_counter = Counter(english_literature_tokens)
for word, count in english_literature_counter.items():
    index2word.append(word)
    word2index[word] = len(word2index)

vocabulary_size = len(word2index)
print(vocabulary_size)
```

```
14309
```

In [24]:
```
# preprocess the dataset to get training data

max_input_len = 1
step = 1
x = []
y = []
for i in range(0, len(english_literature_tokens) - max_input_len, step):
    if i % 100 == 0:
        print("Progress: {0}%".format(round(i / len(english_literature_tokens)
* 100, 2)), end="\r")
    curr_words = english_literature_tokens[i:i + max_input_len]
    x.append([word2index.get(curr_word, 0) for curr_word in curr_words])
    next_word = english_literature_tokens[i + max_input_len]
    y.append(word2index.get(next_word, 0))
X = np.array(x)
Y = to_categorical(y, vocabulary_size)
print("")
print(X.shape, Y.shape)
```

```
Progress: 99.99%
(254532, 1) (254532, 14309)
```

In [24]:
```
model_elman = Sequential()
model_elman.add(Embedding(vocabulary_size, 500, input_length=max_input_len))
model_elman.add(SimpleRNN(units=500, activation='sigmoid'))
model_elman.add(Dense(vocabulary_size, activation='softmax'))

model_elman.summary()
model_elman.compile(optimizer='adam', loss='categorical_crossentropy', metrics
=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 1, 500)            7154500
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, 500)               500500
_________________________________________________________________
dense_9 (Dense)              (None, 14309)             7168809
=================================================================
Total params: 14,823,809
Trainable params: 14,823,809
Non-trainable params: 0
_________________________________________________________________
```

In [19]:
```
print(device_lib.list_local_devices())
filepath = "./model_elman.pth"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=0, save_best_on
ly=True, save_weights_only=False)
elman_training_history = model_elman.fit(
    X,
    Y,
    batch_size=128,
    epochs=20,
    verbose=1,
    shuffle=False,
    callbacks=[checkpoint]
)
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 10287761693967540431
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6700198133
locality {
  bus_id: 1
  links {
  }
}
incarnation: 12434136452038456957
physical_device_desc: "device: 0, name: GeForce GTX 1070, pci bus id: 0000:0
1:00.0, compute capability: 6.1"
]
Epoch 1/20
254532/254532 [==============================] - 53s 208us/sample - loss: 6.1
979 - acc: 0.1224
Epoch 2/20
254532/254532 [==============================] - 48s 189us/sample - loss: 5.5
174 - acc: 0.1526
Epoch 3/20
254532/254532 [==============================] - 48s 189us/sample - loss: 5.2
357 - acc: 0.1665
Epoch 4/20
254532/254532 [==============================] - 48s 189us/sample - loss: 5.0
299 - acc: 0.1753
Epoch 5/20
254532/254532 [==============================] - 48s 187us/sample - loss: 4.8
672 - acc: 0.1808
Epoch 6/20
254532/254532 [==============================] - 48s 188us/sample - loss: 4.7
328 - acc: 0.1837
Epoch 7/20
254532/254532 [==============================] - 48s 188us/sample - loss: 4.6
128 - acc: 0.1862
Epoch 8/20
254532/254532 [==============================] - 48s 189us/sample - loss: 4.5
197 - acc: 0.1876
Epoch 9/20
254532/254532 [==============================] - 48s 188us/sample - loss: 4.4
488 - acc: 0.1884
Epoch 10/20
254532/254532 [==============================] - 47s 186us/sample - loss: 4.3
873 - acc: 0.1892
Epoch 11/20
254532/254532 [==============================] - 48s 188us/sample - loss: 4.3
437 - acc: 0.1900
Epoch 12/20
254532/254532 [==============================] - 48s 187us/sample - loss: 4.3
097 - acc: 0.1906
Epoch 13/20
254532/254532 [==============================] - 47s 186us/sample - loss: 4.2
829 - acc: 0.1913
```

```
Epoch 14/20
254532/254532 [==============================] - 48s 187us/sample - loss: 4.2
612 - acc: 0.1923
Epoch 15/20
254532/254532 [==============================] - 48s 187us/sample - loss: 4.2
413 - acc: 0.1928
Epoch 16/20
254532/254532 [==============================] - 47s 186us/sample - loss: 4.2
252 - acc: 0.1934
Epoch 17/20
254532/254532 [==============================] - 48s 188us/sample - loss: 4.2
130 - acc: 0.1936
Epoch 18/20
254532/254532 [==============================] - 48s 187us/sample - loss: 4.2
054 - acc: 0.1942
Epoch 19/20
254532/254532 [==============================] - 48s 188us/sample - loss: 4.2
032 - acc: 0.1951
Epoch 20/20
254532/254532 [==============================] - 48s 187us/sample - loss: 4.1
998 - acc: 0.1959
```

Report the model cross-entropy loss.

The model cross-entropy loss for the elman + TBTT model (part 1) on the train set is 4.1998

In [25]:
```
''' Elman network + backpropagation through time algorithm (part 2)
'''
from nltk.tokenize import sent_tokenize

# prepare sentence sequences of the dataset
english_literature_sentences = sent_tokenize(english_literature_text)
english_literature_sentences_seq = []
english_literature_sentences_length = []
max_length = 40
for sentence in english_literature_sentences:
    tmp_tokens = word_tokenize(sentence)
    if len(tmp_tokens) > max_length:
        tmp_tokens = tmp_tokens[:max_length]
    for i in range(1, len(tmp_tokens)):
        # 1-of-N encoding
        tmp_seq = tmp_tokens[:i+1]
        tmp_seq_encoded = []
        for token in tmp_seq:
            tmp_seq_encoded.append(word2index[token])
        english_literature_sentences_seq.append(tmp_seq_encoded)
        english_literature_sentences_length.append(len(tmp_seq_encoded))
```

In [26]:
```
print('number of sequences', len(english_literature_sentences_seq))
print('mean sentence length', sum(english_literature_sentences_length) / len(e
nglish_literature_sentences_length))
max_sentence_length = max(english_literature_sentences_length)
print('max sentence length', max_sentence_length)
```

```
number of sequences 210783
mean sentence length 14.081591020148684
max sentence length 40
```

In [27]:
```
# prepare input and target data for training the model
from tensorflow.keras.preprocessing.sequence import pad_sequences

english_literature_sentences_seq = pad_sequences(english_literature_sentences_
seq, maxlen=max_sentence_length, padding='pre')
english_literature_sentences_seq = np.array(english_literature_sentences_seq)
x_BTT = english_literature_sentences_seq[:, :-1]

y_BTT = to_categorical(english_literature_sentences_seq[:, -1], vocabulary_siz
e)
```

In [28]:
```
# build the network with BTT
model_BTT = Sequential()
model_BTT.add(Embedding(vocabulary_size, 500, input_length=max_sentence_length
-1))
model_BTT.add(SimpleRNN(units=500, activation='sigmoid'))
model_BTT.add(Dense(vocabulary_size, activation='softmax'))

model_BTT.summary()
model_BTT.compile(optimizer='adam', loss='categorical_crossentropy', metrics=[
'accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 39, 500)           7154500   
_________________________________________________________________
simple_rnn (SimpleRNN)       (None, 500)               500500    
_________________________________________________________________
dense (Dense)                (None, 14309)             7168809   
=================================================================
Total params: 14,823,809
Trainable params: 14,823,809
Non-trainable params: 0
_________________________________________________________________
```

In [29]:
```
print(device_lib.list_local_devices())
filepath = "./model_BTT.pth"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=0, save_best_on
ly=True, save_weights_only=False)
BTT_training_history = model_BTT.fit(
    x_BTT,
    y_BTT,
    batch_size=128,
    epochs=60,
    verbose=1,
    shuffle=False,
    callbacks=[checkpoint]
)
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 4406640222819138814
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6700198133
locality {
  bus_id: 1
  links {
  }
}
incarnation: 13219286407285742176
physical_device_desc: "device: 0, name: GeForce GTX 1070, pci bus id: 0000:0
1:00.0, compute capability: 6.1"
]
Epoch 1/60
210783/210783 [==============================] - 75s 357us/sample - loss: 5.8
978 - acc: 0.1354 - loss: 5.9117 - acc: 0.134 -
Epoch 2/60
210783/210783 [==============================] - 71s 339us/sample - loss: 5.1
589 - acc: 0.1709
Epoch 3/60
210783/210783 [==============================] - 71s 337us/sample - loss: 4.7
923 - acc: 0.1912
Epoch 4/60
210783/210783 [==============================] - 71s 335us/sample - loss: 4.4
710 - acc: 0.2065
Epoch 5/60
210783/210783 [==============================] - 71s 337us/sample - loss: 4.1
532 - acc: 0.2208
Epoch 6/60
210783/210783 [==============================] - 72s 341us/sample - loss: 3.8
396 - acc: 0.2395
Epoch 7/60
210783/210783 [==============================] - 72s 341us/sample - loss: 3.5
509 - acc: 0.2708
Epoch 8/60
210783/210783 [==============================] - 72s 342us/sample - loss: 3.3
028 - acc: 0.3063
Epoch 9/60
210783/210783 [==============================] - 72s 341us/sample - loss: 3.0
983 - acc: 0.3385
Epoch 10/60
210783/210783 [==============================] - 71s 338us/sample - loss: 2.9
141 - acc: 0.3694
Epoch 11/60
210783/210783 [==============================] - 70s 330us/sample - loss: 2.7
600 - acc: 0.3963
Epoch 12/60
210783/210783 [==============================] - 69s 328us/sample - loss: 2.6
309 - acc: 0.4196
Epoch 13/60
210783/210783 [==============================] - 69s 328us/sample - loss: 2.4
981 - acc: 0.4445
```

```
Epoch 14/60
210783/210783 [==============================] - 69s 328us/sample - loss: 2.3
790 - acc: 0.4676
Epoch 15/60
210783/210783 [==============================] - 69s 328us/sample - loss: 2.2
696 - acc: 0.4892
Epoch 16/60
210783/210783 [==============================] - 69s 328us/sample - loss: 2.1
710 - acc: 0.5084
Epoch 17/60
210783/210783 [==============================] - 69s 329us/sample - loss: 2.0
684 - acc: 0.5301
Epoch 18/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.9
736 - acc: 0.5515
Epoch 19/60
210783/210783 [==============================] - 69s 328us/sample - loss: 1.8
821 - acc: 0.5707
Epoch 20/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.8
165 - acc: 0.5844
Epoch 21/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.7
425 - acc: 0.6010
Epoch 22/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.6
499 - acc: 0.6223
Epoch 23/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.5
417 - acc: 0.6470
Epoch 24/60
210783/210783 [==============================] - 69s 328us/sample - loss: 1.5
061 - acc: 0.6547
Epoch 25/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.4
078 - acc: 0.6788
Epoch 26/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.3
743 - acc: 0.6856
Epoch 27/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.3
115 - acc: 0.6992
Epoch 28/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.2
528 - acc: 0.7124
Epoch 29/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.1
599 - acc: 0.7359
Epoch 30/60
210783/210783 [==============================] - 69s 327us/sample - loss: 1.2
259 - acc: 0.7199
Epoch 31/60
210783/210783 [==============================] - 69s 328us/sample - loss: 1.1
592 - acc: 0.7339
Epoch 32/60
210783/210783 [==============================] - 69s 329us/sample - loss: 1.0
972 - acc: 0.7490
```

```
Epoch 33/60
210783/210783 [==============================] - 69s 328us/sample - loss: 1.0
754 - acc: 0.7542
Epoch 34/60
210783/210783 [==============================] - 69s 328us/sample - loss: 0.9
798 - acc: 0.7768
Epoch 35/60
210783/210783 [==============================] - 69s 328us/sample - loss: 0.9
058 - acc: 0.7950
Epoch 36/60
210783/210783 [==============================] - 69s 328us/sample - loss: 0.8
545 - acc: 0.8086
Epoch 37/60
210783/210783 [==============================] - 69s 328us/sample - loss: 0.8
126 - acc: 0.8184
Epoch 38/60
210783/210783 [==============================] - 69s 327us/sample - loss: 0.9
555 - acc: 0.7817
Epoch 39/60
210783/210783 [==============================] - 69s 327us/sample - loss: 0.9
690 - acc: 0.7769
Epoch 40/60
210783/210783 [==============================] - 69s 327us/sample - loss: 0.9
602 - acc: 0.7786
Epoch 41/60
210783/210783 [==============================] - 70s 333us/sample - loss: 0.7
996 - acc: 0.8163
Epoch 42/60
210783/210783 [==============================] - 71s 336us/sample - loss: 0.7
670 - acc: 0.8261
Epoch 43/60
210783/210783 [==============================] - 70s 334us/sample - loss: 0.8
035 - acc: 0.8145
Epoch 44/60
210783/210783 [==============================] - 71s 335us/sample - loss: 0.7
626 - acc: 0.8244
Epoch 45/60
210783/210783 [==============================] - 71s 334us/sample - loss: 0.7
135 - acc: 0.8378
Epoch 46/60
210783/210783 [==============================] - 71s 337us/sample - loss: 0.6
794 - acc: 0.8457
Epoch 47/60
210783/210783 [==============================] - 71s 338us/sample - loss: 0.6
857 - acc: 0.8434
Epoch 48/60
210783/210783 [==============================] - 71s 338us/sample - loss: 0.6
507 - acc: 0.8521
Epoch 49/60
210783/210783 [==============================] - 71s 335us/sample - loss: 0.7
741 - acc: 0.8192
Epoch 50/60
210783/210783 [==============================] - 71s 336us/sample - loss: 0.6
776 - acc: 0.8413
Epoch 51/60
210783/210783 [==============================] - 71s 335us/sample - loss: 0.7
101 - acc: 0.8332
```

```
Epoch 52/60
210783/210783 [==============================] - 70s 334us/sample - loss: 0.7
991 - acc: 0.8168
Epoch 53/60
210783/210783 [==============================] - 71s 336us/sample - loss: 0.6
600 - acc: 0.8491
Epoch 54/60
210783/210783 [==============================] - 71s 336us/sample - loss: 0.7
992 - acc: 0.8136
Epoch 55/60
210783/210783 [==============================] - 70s 334us/sample - loss: 0.7
041 - acc: 0.8347
Epoch 56/60
210783/210783 [==============================] - 71s 336us/sample - loss: 0.6
926 - acc: 0.8415
Epoch 57/60
210783/210783 [==============================] - 71s 339us/sample - loss: 0.5
841 - acc: 0.8640
Epoch 58/60
210783/210783 [==============================] - 71s 339us/sample - loss: 0.5
690 - acc: 0.8689
Epoch 59/60
210783/210783 [==============================] - 71s 335us/sample - loss: 0.5
758 - acc: 0.8671
Epoch 60/60
210783/210783 [==============================] - 71s 336us/sample - loss: 0.5
191 - acc: 0.8812
```

Report the model cross-entropy loss.

The model cross-entropy loss for the elman + BTT model (part 2) on the train set is 0.5191

In [8]:
```
''' Elman + BTT model with the SimpleRnn unit replaced with a Gated Recurrent
 Unit (part 3)
'''
from tensorflow.keras.layers import GRU

model_BTT_GRU = Sequential()
model_BTT_GRU.add(Embedding(vocabulary_size, 500, input_length=max_sentence_le
ngth-1))
model_BTT_GRU.add(GRU(units=500, activation='sigmoid'))
model_BTT_GRU.add(Dense(vocabulary_size, activation='softmax'))

model_BTT_GRU.summary()
model_BTT_GRU.compile(optimizer='adam', loss='categorical_crossentropy', metri
cs=['accuracy'])
```

```
WARNING:tensorflow:From C:\Users\songyih\AppData\Roaming\Python\Python37\site
-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with
(from tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding (Embedding)        (None, 39, 500)           7154500   
_________________________________________________________________
gru (GRU)                    (None, 500)               1501500   
_________________________________________________________________
dense (Dense)                (None, 14309)             7168809   
=================================================================
Total params: 15,824,809
Trainable params: 15,824,809
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
print(device_lib.list_local_devices())
filepath = "./model_BTT_GRU.pth"
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=0, save_best_only=True, save_weights_only=False)
BTT_GRU_training_history = model_BTT_GRU.fit(
    x_BTT,
    y_BTT,
    batch_size=128,
    epochs=100,
    verbose=1,
    shuffle=False,
    callbacks=[checkpoint]
)
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 6192848861112352318
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6700198133
locality {
  bus_id: 1
  links {
  }
}
incarnation: 16910545245568130575
physical_device_desc: "device: 0, name: GeForce GTX 1070, pci bus id: 0000:0
1:00.0, compute capability: 6.1"
]
WARNING:tensorflow:From C:\Users\songyih\AppData\Roaming\Python\Python37\site
-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.p
ython.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/100
210783/210783 [==============================] - 132s 629us/sample - loss: 5.
9056 - acc: 0.1354
Epoch 2/100
210783/210783 [==============================] - 128s 605us/sample - loss: 5.
1159 - acc: 0.1751
Epoch 3/100
210783/210783 [==============================] - 128s 606us/sample - loss: 4.
6989 - acc: 0.1985
Epoch 4/100
210783/210783 [==============================] - 127s 604us/sample - loss: 4.
3138 - acc: 0.2184
Epoch 5/100
210783/210783 [==============================] - 127s 604us/sample - loss: 3.
9169 - acc: 0.2385
Epoch 6/100
210783/210783 [==============================] - 126s 598us/sample - loss: 3.
5159 - acc: 0.2794
Epoch 7/100
210783/210783 [==============================] - 126s 598us/sample - loss: 3.
1664 - acc: 0.3358
Epoch 8/100
210783/210783 [==============================] - 126s 598us/sample - loss: 2.
8839 - acc: 0.3831
Epoch 9/100
210783/210783 [==============================] - 126s 599us/sample - loss: 2.
6343 - acc: 0.4268
Epoch 10/100
210783/210783 [==============================] - 126s 598us/sample - loss: 2.
4109 - acc: 0.4687
Epoch 11/100
210783/210783 [==============================] - 126s 599us/sample - loss: 2.
2013 - acc: 0.5114
Epoch 12/100
```

```
210783/210783 [==============================] - 126s 599us/sample - loss: 2.
0082 - acc: 0.5529
Epoch 13/100
210783/210783 [==============================] - 126s 598us/sample - loss: 1.
8329 - acc: 0.5919
Epoch 14/100
210783/210783 [==============================] - 126s 598us/sample - loss: 1.
6699 - acc: 0.6291
Epoch 15/100
210783/210783 [==============================] - 126s 599us/sample - loss: 1.
5212 - acc: 0.6641
Epoch 16/100
210783/210783 [==============================] - 126s 598us/sample - loss: 1.
3847 - acc: 0.6952
Epoch 17/100
210783/210783 [==============================] - 126s 599us/sample - loss: 1.
2613 - acc: 0.7244
Epoch 18/100
210783/210783 [==============================] - 126s 598us/sample - loss: 1.
1493 - acc: 0.7506
Epoch 19/100
210783/210783 [==============================] - 126s 599us/sample - loss: 1.
0496 - acc: 0.7736
Epoch 20/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
9609 - acc: 0.7945
Epoch 21/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
8822 - acc: 0.8132
Epoch 22/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
8150 - acc: 0.8281
Epoch 23/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
7473 - acc: 0.8437
Epoch 24/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
6904 - acc: 0.8568
Epoch 25/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
6392 - acc: 0.8681
Epoch 26/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
6048 - acc: 0.8755
Epoch 27/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
5711 - acc: 0.8820
Epoch 28/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
5375 - acc: 0.8895
Epoch 29/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
5074 - acc: 0.8948
Epoch 30/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
4871 - acc: 0.8989
Epoch 31/100
```

```
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
4622 - acc: 0.9042
Epoch 32/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
4536 - acc: 0.9043
Epoch 33/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
4629 - acc: 0.8998
Epoch 34/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
4242 - acc: 0.9095
Epoch 35/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
4215 - acc: 0.9098
Epoch 36/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
4084 - acc: 0.9120
Epoch 37/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3892 - acc: 0.9158
Epoch 38/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3882 - acc: 0.9153
Epoch 39/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3806 - acc: 0.9161
Epoch 40/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3771 - acc: 0.9162
Epoch 41/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3740 - acc: 0.9160
Epoch 42/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3653 - acc: 0.9177
Epoch 43/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3535 - acc: 0.9206
Epoch 44/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3568 - acc: 0.9188
Epoch 45/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3674 - acc: 0.9154
Epoch 46/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3564 - acc: 0.9176
Epoch 47/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3552 - acc: 0.9182
Epoch 48/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3422 - acc: 0.9208
Epoch 49/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3378 - acc: 0.9216
Epoch 50/100
```

```
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3356 - acc: 0.9222
Epoch 51/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3225 - acc: 0.9243
Epoch 52/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3415 - acc: 0.9194
Epoch 53/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3299 - acc: 0.9217
Epoch 54/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3559 - acc: 0.9151
Epoch 55/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3623 - acc: 0.9126
Epoch 56/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3311 - acc: 0.9214
Epoch 57/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3184 - acc: 0.9245
Epoch 58/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3347 - acc: 0.9201
Epoch 59/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3311 - acc: 0.9207
Epoch 60/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3830 - acc: 0.9072
Epoch 61/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3331 - acc: 0.9201
Epoch 62/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3170 - acc: 0.9241
Epoch 63/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3271 - acc: 0.9214
Epoch 64/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3275 - acc: 0.9203
Epoch 65/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3177 - acc: 0.9226
Epoch 66/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3126 - acc: 0.9242
Epoch 67/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3091 - acc: 0.9247
Epoch 68/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
3038 - acc: 0.9259
Epoch 69/100
```

```
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3096 - acc: 0.9236
Epoch 70/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3086 - acc: 0.9238
Epoch 71/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3105 - acc: 0.9237
Epoch 72/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3101 - acc: 0.9231
Epoch 73/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
2973 - acc: 0.9265
Epoch 74/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
3098 - acc: 0.9230
Epoch 75/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3003 - acc: 0.9252
Epoch 76/100
210783/210783 [==============================] - 126s 599us/sample - loss: 0.
2956 - acc: 0.9266
Epoch 77/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
2968 - acc: 0.9254
Epoch 78/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2978 - acc: 0.9252
Epoch 79/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3045 - acc: 0.9242
Epoch 80/100
210783/210783 [==============================] - 126s 596us/sample - loss: 0.
3028 - acc: 0.9237
Epoch 81/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2958 - acc: 0.9260
Epoch 82/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2972 - acc: 0.9256
Epoch 83/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
2921 - acc: 0.9267
Epoch 84/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
2906 - acc: 0.9269
Epoch 85/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2968 - acc: 0.9250
Epoch 86/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2919 - acc: 0.9258
Epoch 87/100
210783/210783 [==============================] - 126s 598us/sample - loss: 0.
2892 - acc: 0.9268
Epoch 88/100
```

```
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2914 - acc: 0.9260
Epoch 89/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3133 - acc: 0.9209
Epoch 90/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3056 - acc: 0.9219
Epoch 91/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2925 - acc: 0.9258
Epoch 92/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2989 - acc: 0.9238
Epoch 93/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2972 - acc: 0.9247
Epoch 94/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2988 - acc: 0.9239
Epoch 95/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3080 - acc: 0.9215
Epoch 96/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3053 - acc: 0.9218
Epoch 97/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
2982 - acc: 0.9237
Epoch 98/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3002 - acc: 0.9240
Epoch 99/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3087 - acc: 0.9216
Epoch 100/100
210783/210783 [==============================] - 126s 597us/sample - loss: 0.
3027 - acc: 0.9229
```

Report the model cross-entropy loss.

The model cross-entropy loss for the elman + BTT with GRU is 0.2892

Compare your results in terms of cross-entropy loss with two other approach (part 1 and 2)

- Elman + TBTT (part 1): cross-entropy loss 4.1998, acc 0.1959
- Elman + BTT (part 2): best cross-entropy loss 0.5191, acc 0.8812
- Elman + BTT with the SimpleRNN unit replaced with GRU (part 3): best cross-entropy loss 0.2892, acc 0.9268

The cross-entropy loss of Elman + BTT network with the SimpleRNN unit replaced with GRU is the best among these three model.

In [31]:
```
'''Use each model to generate 10 synthetic sentences of 15 words each
'''
word_num = 15
sentence_num = 10

# basic elman with TBTT model (part 1)
elman_TBTT = load_model('./model_elman.pth')
for i in range(sentence_num):
    # randomly choose init word
    init_encoded = np.random.randint(vocabulary_size)
    encoded_sequence = [init_encoded]
    # generate the predicted sequence
    for _ in range(word_num - 1):
        latest_word_encoded = [encoded_sequence[-1]]
        latest_word_encoded = np.array(latest_word_encoded)
        predicted_encoded = elman_TBTT.predict_classes(latest_word_encoded, ve
rbose=0)
        encoded_sequence.append(predicted_encoded[0])
    # decode the sequence
    decoded_sequence = []
    for encoded_word in encoded_sequence:
        decoded_sequence.append(index2word[encoded_word])
    print(' '.join(decoded_sequence))
```

```
studded all the world , I have no more than the world , I have
gaze this island . PROSPERO : I have no more than the world , I
chid'st me , I have no more than the world , I have no more
measuring his name , I have no more than the world , I have no
planched gate And , I have no more than the world , I have no
Senators : I have no more than the world , I have no more than
drift ; And , I have no more than the world , I have no
grazing , I have no more than the world , I have no more than
cap of the world , I have no more than the world , I have
corrupted foul play the world , I have no more than the world , I
```

In [32]:
```
# elman with BTT model (part 2)
elman_BTT = load_model('./model_BTT.pth')
for i in range(sentence_num):
    # randomly choose init word
    init_encoded = np.random.randint(vocabulary_size)
    encoded_sequence = [init_encoded]
    # generate the predicted sequence
    for _ in range(word_num - 1):
        input_sequence = pad_sequences([encoded_sequence], maxlen=max_sentence
_length-1, padding='pre')
        input_sequence = np.array(input_sequence)
        predicted_encoded = elman_BTT.predict_classes(input_sequence, verbose=
0)
        encoded_sequence.append(predicted_encoded[0])
    # decode the sequence
    decoded_sequence = []
    for encoded_word in encoded_sequence:
        decoded_sequence.append(index2word[encoded_word])
    print(' '.join(decoded_sequence))
```

```
dip'dst in view ; but first was struck with me than never will be ruled
bud of better ; he does offend my brother ? ' Lord , how have
noisemaker ! ' I hate thee by your side ; and see this night or
punto reverso ! ' I not ? -- No ; I will resist such entertainment
strokedst me and madest much of him ! ' I ' the plain way is
births : On whom God will never yet a word , we hear the minstrels
It is a hint That wrings mine eyes to't . ' n ' the air
awaked him , we 'll be put to woo . ' I ' the part
footing of the city ? ' song , the great subject well , she 'll
Fill me for that gird , good Tranio , for that thou likest it not
```

In [33]:
```
# elman with BTT model with the SimpleRnn unit replaced with GRU (part 3)
elman_BTT_GRU = load_model('./model_BTT_GRU.pth')
for i in range(sentence_num):
    # randomly choose init word
    init_encoded = np.random.randint(vocabulary_size)
    encoded_sequence = [init_encoded]
    # generate the predicted sequence
    for _ in range(word_num - 1):
        input_sequence = pad_sequences([encoded_sequence], maxlen=max_sentence
_length-1, padding='pre')
        input_sequence = np.array(input_sequence)
        predicted_encoded = elman_BTT_GRU.predict_classes(input_sequence, verb
ose=0)
        encoded_sequence.append(predicted_encoded[0])
    # decode the sequence
    decoded_sequence = []
    for encoded_word in encoded_sequence:
        decoded_sequence.append(index2word[encoded_word])
    print(' '.join(decoded_sequence))
```

```
Devised are thee ? ' to the Capitol ! -- I am : and here
pluck in the topsail . ' the air doth burn . ' the air doth
fan , this Claudio is mine only son . ' the other way In that
shield me not first ? ' to the gaol . ' the house , how
abusing Baptista is to a cause to sigh , Then he shall command his mind
ye are not so mad -- That thou hast cause to pry into this morning
commonly and the air And each more villain : if these thing it is ,
pieces : He hath not possible nor prayers ; and he be too noble for
couples : You seem to hear of this : you have such vantage in this
fixes renown 'd two in this field We fall in broil . ' the last
```

Discuss the quality of the sentences generated

The sentences generated by basic elman with TBTT model (part 1) doesn't look like English at all and seems to be overfitting as it is simply repeating some phrases, while sentences generated by the later 2 models look much better. Although the elman with BTT model cannot form a nice sentence, it produces some natural phrases. And the elman + BTT with SimpleRNN replaced with GRU performs the best among these three models. Though it still doesn't work perfectly, it is able to somehow generate some sentences that looks like English, and matches the training set.

In [5]:
```
''' Compare the words representation extracted from each of the approaches usi
ng one of the existing methods. (part 4)
Here I choose to use intrinsic evaluation method.
I evaluate the model by compare the similarity of my three models and gold sta
ndard similarity dataset
'''

# load wordsim_similarity_goldstandard as benchmark
goldstandard_word0 = []
goldstandard_word1 = []
goldstandard_similarity = []
vocabulary_keys = word2index.keys()
found_pairs = 0
total_pairs = 0
with open('./datasets/wordsim_similarity_goldstandard.txt') as f:
    lines = f.readlines()
    for line in lines:
        total_pairs += 1
        temp = line.strip().split('\t')
        # only save the pair of words that can be found in our vocabulary
        if temp[0] in vocabulary_keys and temp[1] in vocabulary_keys:
            found_pairs += 1
            goldstandard_word0.append(temp[0])
            goldstandard_word1.append(temp[1])
            goldstandard_similarity.append(float(temp[2]))
```

In [6]:
```
print('Found {0} pair of words in local vocabulary out of {1} pair of words in
wordsim_similarity_goldstandard'.format(found_pairs, total_pairs))
```

```
Found 66 pair of words in local vocabulary out of 203 pair of words in wordsi
m_similarity_goldstandard
```

In [7]:
```
# calculate similarity for elman + TBTT model (part 1) on the found pair of wo
rds

# load model
elman_TBTT = load_model('./model_elman.pth')
elman_TBTT_embedding = elman_TBTT.layers[0].get_weights()[0]
elman_TBTT_embedding = np.array(elman_TBTT_embedding)
```

```
WARNING:tensorflow:From C:\Users\songyih\AppData\Roaming\Python\Python37\site
-packages\tensorflow\python\ops\resource_variable_ops.py:435: colocate_with
(from tensorflow.python.framework.ops) is deprecated and will be removed in a
future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\songyih\AppData\Roaming\Python\Python37\site
-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.p
ython.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

In [37]:
```
from sklearn.metrics.pairwise import cosine_similarity

# do the calculation
elman_TBTT_similarity = []
for i in range(len(goldstandard_similarity)):
    index_word0 = word2index[goldstandard_word0[i]]
    index_word1 = word2index[goldstandard_word1[i]]
    elman_TBTT_similarity.append(cosine_similarity(elman_TBTT_embedding[index_
word0].reshape(1, -1), elman_TBTT_embedding[index_word1].reshape(1, -1))[0][0
])
```

In [38]:
```
# calculate the Spearman rank correlation on our similarity and goldstandard s
imilarity

from scipy import stats
elman_TBTT_correlation = stats.spearmanr(np.array(elman_TBTT_similarity), np.a
rray(goldstandard_similarity))
print('Spearman rank correlation between elman + TBTT (part 1) and gold starnd
ard similarity is ', elman_TBTT_correlation.correlation.round(4))
```

```
Spearman rank correlation between elman + TBTT (part 1) and gold starndard si
milarity is  0.1581
```

In [43]:
```
# calculate similarity for elman + BTT model (part 2) on the found pair of wor
ds

elman_BTT = load_model('./model_BTT_max40.pth')
elman_BTT_embedding = elman_BTT.layers[0].get_weights()[0]
elman_BTT_embedding = np.array(elman_BTT_embedding)
```

In [44]:
```
from sklearn.metrics.pairwise import cosine_similarity

# do the calculation
elman_BTT_similarity = []
for i in range(len(goldstandard_similarity)):
    index_word0 = word2index[goldstandard_word0[i]]
    index_word1 = word2index[goldstandard_word1[i]]
    elman_BTT_similarity.append(cosine_similarity(elman_BTT_embedding[index_wo
rd0].reshape(1, -1), elman_BTT_embedding[index_word1].reshape(1, -1))[0][0])
```

In [45]:
```
# calculate the Spearman rank correlation on our similarity and goldstandard s
imilarity

from scipy import stats
elman_BTT_correlation = stats.spearmanr(np.array(elman_BTT_similarity), np.arr
ay(goldstandard_similarity))
print('Spearman rank correlation between elman + BTT (part 2) and gold starnda
rd similarity is ', elman_BTT_correlation.correlation.round(4))
```

```
Spearman rank correlation between elman + BTT (part 2) and gold starndard sim
ilarity is  0.2478
```

In [46]:
```
# calculate similarity for elman + BTT model with GRU (part 3) on the found pa
ir of words

# load model
elman_BTT_GRU = load_model('./model_BTT_GRU.pth')
elman_BTT_GRU_embedding = elman_BTT_GRU.layers[0].get_weights()[0]
elman_BTT_GRU_embedding = np.array(elman_BTT_GRU_embedding)
```

In [47]:
```
from sklearn.metrics.pairwise import cosine_similarity

# do the calculation
elman_BTT_GRU_similarity = []
for i in range(len(goldstandard_similarity)):
    index_word0 = word2index[goldstandard_word0[i]]
    index_word1 = word2index[goldstandard_word1[i]]
    elman_BTT_GRU_similarity.append(cosine_similarity(elman_BTT_GRU_embedding[
index_word0].reshape(1, -1), elman_BTT_GRU_embedding[index_word1].reshape(1, -
1))[0][0])
```

In [48]:
```
# calculate the Spearman rank correlation on our similarity and goldstandard s
imilarity

from scipy import stats
elman_BTT_GRU_correlation = stats.spearmanr(np.array(elman_BTT_GRU_similarity
), np.array(goldstandard_similarity))
print('Spearman rank correlation between elman + BTT with GRU (part 3) and gol
d starndard similarity is ', elman_BTT_GRU_correlation.correlation.round(4))
```

```
Spearman rank correlation between elman + BTT with GRU (part 3) and gold star
ndard similarity is  0.2482
```

Compare the words representation extracted from each of the approaches using one of the existing methods.

I used intrinsic evaluation method to evaluate the extracted words representations

I chose the gold standard similarity dataset as benchmark, then found the word pairs that appears on both my local vocabulary and benchmark dataset.

After that I calculated the similarity for the found word pairs on the three models. Finally I calculated the Spearman rank correlation between the similarity of my models and the benchmark.

Here are the correlation results:

- elman + TBTT (part 1): 0.1581
- elman + BTT (part 2): 0.2478
- elman + BTT with GRU (part 3): 0.2482

The result is actually consistent with the text generation result above, that the elman + TBTT works the worst and the other two are much better.

In [3]:
```
''' Learn an RNN model that predicts document categories given its content (pa
rt 5)
'''

news_tokens = []
for news_item in news:
    news_tokens.append(word_tokenize(news_item))
```

In [4]:
```
from collections import Counter

# build vocabulary
word2index_news = {}
index2word_news = []
flattend_news_tokens = []
for sublist in news_tokens:
    for item in sublist:
        flattend_news_tokens.append(item)
news_counter = Counter(flattend_news_tokens)

for word, count in news_counter.items():
    index2word_news.append(word)
    word2index_news[word] = len(word2index_news)

news_vocabulary_size = len(word2index_news)
print(news_vocabulary_size)
```

```
207442
```

In [5]:
```
# encode the dataset
max_news_length = 400
news_encoded = []
news_encoded_length = []
for news_tokens_item in news_tokens:
    if len(news_tokens_item) > max_news_length:
        news_tokens_item = news_tokens_item[:max_news_length]
    tmp_encoded = []
    for news_token in news_tokens_item:
        tmp_encoded.append(word2index_news[news_token])
    news_encoded.append(tmp_encoded)
    news_encoded_length.append(len(tmp_encoded))
```

In [6]:
```
print('number of news', len(news_encoded))
print('mean news length', sum(news_encoded_length) / len(news_encoded_length))
max_news_length = max(news_encoded_length)
print('max news length', max_news_length)
```

```
number of news 13108
mean news length 238.1869087580104
max news length 400
```

In [7]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences

# prepare the input and target for the network
news_encoded_padded = pad_sequences(news_encoded, maxlen=max_news_length, padd
ing='pre')
x_news = np.array(news_encoded_padded)
y_news = to_categorical(groups, 4)
```

In [8]:
```
# split train test dataset
from sklearn.model_selection import train_test_split
x_news_train, x_news_test, y_news_train, y_news_test = train_test_split(x_news
, y_news, test_size=0.1, stratify=y_news)
```

In [13]:
```
# define the new network structure
model_news = Sequential()
model_news.add(Embedding(news_vocabulary_size, 400, input_length=max_news_leng
th))
model_news.add(SimpleRNN(units=500, activation='sigmoid'))
model_news.add(Dropout(0.5))
model_news.add(Dense(256, activation='sigmoid'))
model_news.add(Dropout(0.5))
model_news.add(Dense(4, activation='softmax'))

model_news.summary()
model_news.compile(optimizer='adam', loss='categorical_crossentropy', metrics=
['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_3 (Embedding)      (None, 400, 400)          82976800  
_________________________________________________________________
simple_rnn_3 (SimpleRNN)     (None, 500)               450500    
_________________________________________________________________
dropout_5 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_5 (Dense)              (None, 256)               128256    
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0         
_________________________________________________________________
dense_6 (Dense)              (None, 4)                 1028      
=================================================================
Total params: 83,556,584
Trainable params: 83,556,584
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
print(device_lib.list_local_devices())
filepath = "./model_news.pth"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=0, save_bes
t_only=True, save_weights_only=False)
news_training_history = model_news.fit(
    x_news_train,
    y_news_train,
    batch_size=128,
    epochs=20,
    verbose=1,
    shuffle=True,
    validation_data=(x_news_test, y_news_test),
    callbacks=[checkpoint]
)
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 7053203860371525633
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6700198133
locality {
  bus_id: 1
  links {
  }
}
incarnation: 12412288893670240266
physical_device_desc: "device: 0, name: GeForce GTX 1070, pci bus id: 0000:0
1:00.0, compute capability: 6.1"
]
Train on 11797 samples, validate on 1311 samples
Epoch 1/20
11797/11797 [==============================] - 29s 2ms/sample - loss: 1.3739
- acc: 0.3376 - val_loss: 1.3090 - val_acc: 0.3722
Epoch 2/20
11797/11797 [==============================] - 29s 2ms/sample - loss: 1.3144
- acc: 0.3673 - val_loss: 1.2973 - val_acc: 0.3722
Epoch 3/20
11797/11797 [==============================] - 28s 2ms/sample - loss: 1.1890
- acc: 0.4512 - val_loss: 1.0961 - val_acc: 0.5286
Epoch 4/20
11797/11797 [==============================] - 28s 2ms/sample - loss: 0.8521
- acc: 0.6516 - val_loss: 1.0089 - val_acc: 0.5881
Epoch 5/20
11797/11797 [==============================] - 29s 2ms/sample - loss: 0.5872
- acc: 0.7726 - val_loss: 0.9626 - val_acc: 0.6102
Epoch 6/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.3978
- acc: 0.8554 - val_loss: 0.9736 - val_acc: 0.6331
Epoch 7/20
11797/11797 [==============================] - 27s 2ms/sample - loss: 0.2928
- acc: 0.8979 - val_loss: 1.0004 - val_acc: 0.6812
Epoch 8/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.2244
- acc: 0.9242 - val_loss: 1.1662 - val_acc: 0.6377
Epoch 9/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1895
- acc: 0.9401 - val_loss: 1.1066 - val_acc: 0.6606
Epoch 10/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1383
- acc: 0.9551 - val_loss: 1.1644 - val_acc: 0.6873
Epoch 11/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1250
- acc: 0.9582 - val_loss: 1.2155 - val_acc: 0.6850
Epoch 12/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1293
- acc: 0.9569 - val_loss: 1.1849 - val_acc: 0.6926
Epoch 13/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1079
```

```
- acc: 0.9672 - val_loss: 1.2746 - val_acc: 0.6888
Epoch 14/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1028
- acc: 0.9663 - val_loss: 1.3048 - val_acc: 0.7079
Epoch 15/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1122
- acc: 0.9624 - val_loss: 1.3464 - val_acc: 0.6773
Epoch 16/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.0910
- acc: 0.9702 - val_loss: 1.3871 - val_acc: 0.6850
Epoch 17/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.2187
- acc: 0.9375 - val_loss: 1.9257 - val_acc: 0.3257
Epoch 18/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.5127
- acc: 0.8009 - val_loss: 1.3939 - val_acc: 0.6323
Epoch 19/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1708
- acc: 0.9457 - val_loss: 1.3796 - val_acc: 0.6629
Epoch 20/20
11797/11797 [==============================] - 26s 2ms/sample - loss: 0.1208
- acc: 0.9674 - val_loss: 1.3930 - val_acc: 0.6621
```

Report your accuracy results on the validation set.

The best validation loss of the model is 0.9626 at epoch 5, and the best accuracy of the model is 0.7079 at epoch 14.

In [ ]: