ESE 577: Deep Learning Algorithms and Software Spring 2022

Project Report for Final Project

May 20, 2022

Yang Xie 114394356 is mainly responsible for algorithm, code implementation
Ziyan Yuan 114627023 is mainly responsible for project report and slides

Table of Contents

1.Introduction

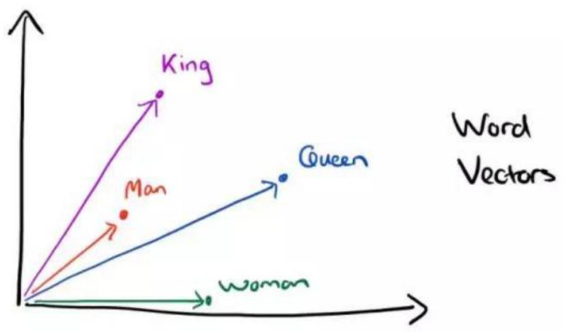
Convolutional neural networks (CNNs) have made great achievements in image processing, with their convolutional and pooling structures that are good at extracting information from images, while recurrent neural networks (RNNs) are more commonly used in NLP, where RNNs and their various variants are better at processing context due to their memory capabilities. However, many aspects of the NLP domain have achieved excellent results with CNNs, such as semantic analysis, query retrieval, text classification, and other tasks. This project will use LSTM for prediction.

2.Description

This project, we use CBOW + LSTM.

CBOW is used for compressed tokens. In our article, for each unique token, they have a one-hot code respectively. But, one-hot code is [0,0,1,......,0,0,0],which has many dimensions. In this way, the project needs more calculations. We introduced CBOW to mitigate this situation. By training this model, we can get a embedding-martix. This method of reducing high dimensionality to low dimensionality is what we call word-embedding. Here is an easy diagram.

$$\vec{King} - \vec{Man} + \vec{Woman} = \vec{Queen}$$

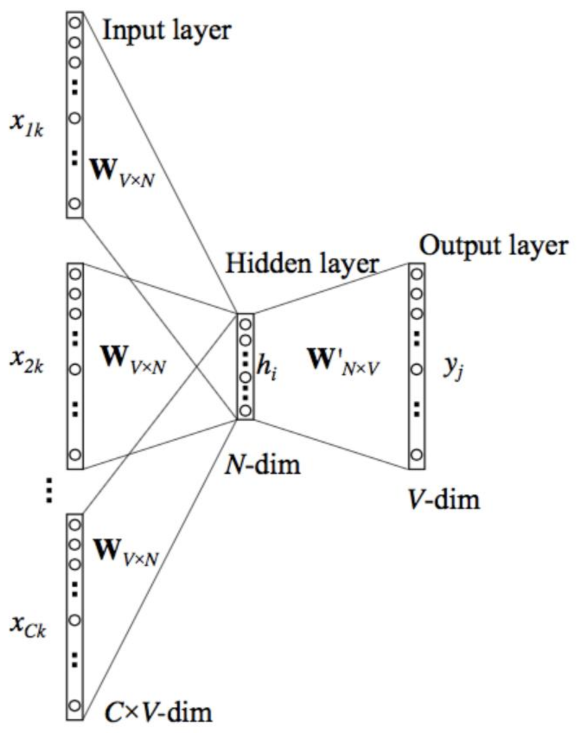


When we want to sreach a word, we can get their vector by doing less calculation.The following picture is an good example.

The left array is one-hot code. The 3x5 martix is embedding martix training by CBOW model. Finally we will get a lower dimensional vector, so called embedding word at right.

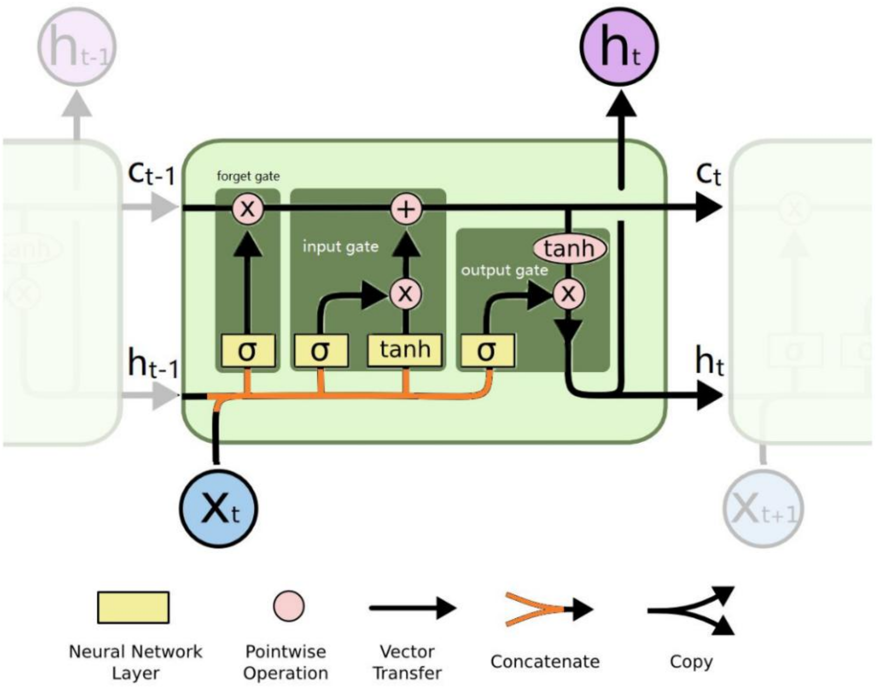
$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

This diagram shows that how to train this model to get embedding-martix. Its implementation is in part3.

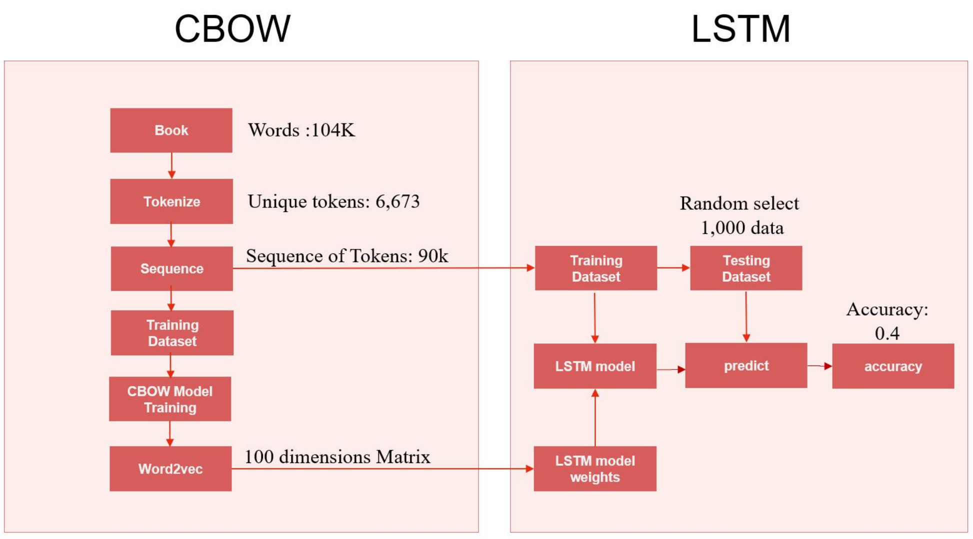


The LSTM model is used for text prediction.

Here is a framework of LSTM. And following pictiure shows that symbols meaning.



LSTM is actually a complex logic gate. This model can be trained after the calculation of selection gates, input gates, forgetting gates, etc.



3. Algorithm and Implementation

Libaray: Python 3.9; keras 2.43; torch 1.11; numpy 1.18.5; tqdm 4.64; matplotlib 3.4.3; sklearn 0.0; tensorflow 2

In [1]:

```
import torch
import torch, nn as nn
import torch, nn. functional as F
import torch, optim as optim
from torch, utils, data import Dataset, DataLoader
import numpy as np
from collections import Counter
from tqdm import tqdm
from matplotlib import pyplot as pit
from ski earn, decomposition import PCA
import string
from keras. callbacks import LambdaCallback
from keras. layers, recurrent import LSTM
from keras. layers, embeddings import Embedding
from keras. layers import Dense, Activation
from keras. models import Sequential
from sklearn. metrics import accuracy_score
```

In [2]:

```
,,,
ESE577 Final Project
©data: 2022/05/16
©author1: Yang Xie 114394356
@author2: Ziyan Yuan 114627023
, , ,
```

Out [2]:

```
'\nESE577 Final Project\n@data: 2022/05/16\n@author: Yang Xie 114394356\n@partner: Z iyan Yuan 114627023\n
```

In (3):

```
class Book2DataSet(): def ___init___(self, data_file_path, window_size):
        with open(data_file_path,' r', encoding='utf-8') as f:
            doc = f. read() words_tokenized = self. clean_doc(doc)
        # [ (['project', 'gutenberg', 'ebook' ], ' of'), (['gutenberg', 'ebook', 'of' ], 'the'), (['
        eboo self. context_target = [([words_tokenized[i- (j+1)] for j in reversed(range(window_size)) ],
        for i in range(window_size, len(words_tokenized)-window_size)] self, vocab = Counter(words_tokenized)
        self. word_to_idx = (word_tuple[0]: idx for idx, word_tuple in enumerate(self, vocab. most_com self.
        idx_to_word = list (self. word_to_idx. keys())
        self. vocab_size = len(self, vocab)
        self. window_size = window_size

    def —getitem___(self, idx):
        context = torch, tensor([self. word_to_idx[w] for w in self. context_target[idx] [0]]) target =
        torch, tensor([self. word_to_idx[self. context_target[idx][1]]])
        return context, target

    def __len__(self):
        return len(self. context_target)

    # turn a doc into clean tokens
    def clean_doc(self, doc):
        doc = doc. replace (' " replace ('
        split into tokens by white space tokens = doc. split ()
        remove punctuation from each token
        tokens = [' ' if w in string, punctuation else w for w in tokens]
        remove remaining tokens that are not alphabetic tokens =[word for word in tokens if word. isalpha0]
        tokens = [word, lower() for word in tokens]
        return tokens
```

In the Book2Dataset, we get the tokens and other data by removing the various symbols and numbers from the text.

In [4]:

```
def print_k_nearest_neighbour(X, idx, k, idx_to_word):
    dists = np. dot((X - X[idx]) ** 2, np. ones(X. shape[1])) idxs = np. argsort(dists)[:k]
    print C The {} nearest neighbour of {} are: '. format(str(k), idx_to_word[idx])) for i in idxs:
        print (idx_to_word[i])
    return idxs
```

In [5]:

```
class CBOW(nn. Module):
    def __init__(self, vocabsize, embedding_dim, window_size): super (CBOW, self). _______init___ () self,
        embeddings = nn.Embedding(vocab_size, embedding_dim) self, linear = nn. Linear(embedding_dim,
        vocab_size) self.windowsize = windowsize

    def forward(self, inputs): embeds = torch, sum(self, embeddings(inputs), dim=1) out =
        self.linear(embeds) tisoftmax compute log probability log_probs = F. log_softmax(out, dim=1)
        return log_probs
```

In the CBOW function, we use softmax to compute log probability. Wthen we choose to use NLLLoss later(negative log likelihood loss). Cross-entropy loss is equivalent to log softmax + NLLLoss. So In the CBOW model, we try to calculate its CEloss.

In [6]:

```
#STEP1, CBOW model training params definition
WINDOWS_SIZE = 3
EMBEDDING_DIM = 100
BATCH_SIZE = 500
NUM_EPOCH = 30
data_f ilejpath = '. /data/The Adventures of Sherlock Holmes.txt'
```

The WINDOWS_SIZE can determine the strength of the tokens relationship. When the size is small, the tokens will show a close relationship. In another way, the model thinks that they have similar meaning. EMBEDDING_DIM is dimensions. It can determine the size of embedding matrix. This part was discussed in part2, one-hot code(6673 dimensions) multiple by embedding martix(6673x100) = embedding words (100 dimensions). Each calculation requires one-hot code and matrix multiplication instead of taking one-hot code directly, reducing many computational steps.

In [7]:

```
WSTEP2, get training dataset for CBOW
data = Book2DataSet(data_file_path, WINDOWS_SIZE)
print C Total sequences of tokens: %d' % len(data. context_target))
print('Unique Tokens: %d' % data. vocab_size)
print(' The first 10 sequences of tokens: %s' % data. context_target[:10])
```

```
Total sequences of tokens: 90342
Unique Tokens: 6673
The first 10 sequences of tokens: [(['project', ' gutenberg', 'ebook'], 'of'), ([' gut enberg', ' ebook', '
of' ], ' the'), ([' ebook', ' of', ' the' ], ' adventures'), ([' of', 'th e , ' adventures' ]...
```

In [10]:

```
#STEP3, prepare CBOW training model
model = CBOW(data. vocab_size, EMBEDDING_DIM, WINDOWS_SIZE) Woptimizer = optim. SGD(model, parameters(),
lr=0. 001) optimizer = optim. Adam(model, parameters(), lr=0.01) loss_function = nn. NLLLoss()
losses =[] data_loader = DataLoader(data, batch_size=BATCH_SIZE) cuda_available = torch, cuda.
is_available()
```

In [15]:

```
#STEP4, start the training process
for epoch in range(NUM_EPOCH):
    total_loss = 0
    for context, target in tqdm(data_loader):
        if context. size()[0] != BATCH SIZE:
            continue
        if cuda_available:
            context = context. cuda()
            target = target, squeeze(1). cuda()
            model = model. cuda()
        else :
            target = target, squeeze(1)
        model. zero_grad()
        log_probs = model(context)
        loss = loss_function(log_probs, target)
        loss, backward ()
        optimizer, step()
        total loss += loss, item()
    losses, append(total loss)
    print(' epoch {}/{}; total_loss:      . format(str(epoch), str(NU_EPOCH), str(total_loss)))
```

```
100% I ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■I 181/181
[00:07<00:00, 22. 63it/s]

The 0/30 total_loss: 488. 55293107032776

100% I ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■


■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■I 181/181
[00:07<00:00, 22. 84it/s]

The 1/30 total_loss: 487. 1513934135437

100% I ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

NUM_EPOCH = 30. In this for loop, we calculate the value of loss. The loss function is equivalent to Crossentropy loss.

modeLzero_grad():Emptying of past gradients;
loss.backwardf):backward propagation, calculating the current aradient;
optimizer.step():Update the parameters accordina to the Gradient.

```
In [16]: #STEP5, get the training results, this is the word2vec matrix of CBOW embed_matrix = model,
embeddings, weight, detach (). cpu (). numpy ()
# test the results, it will show some words that have a close relation to the word ' she', ' is', ' goo
print_k_nearest_neighbour(embed_matrix, data. word_to_idx['she' ], 10, list(data. word_to_idx. keys()))
print_k_nearest_neighbour(embed_matrix, data. word_to_idx[' is' ], 10, list(data. word_to_idx. keys()))
print_k_nearest_neighbour(embed_matrix, data. word_to_idx[' good' ], 10, list(data. word_to_idx. keys()))

The 10 nearest neighbour of she are:
she
he
i
you
it
the
and
to
a
we
The 10 nearest neighbour of is are:
is
was
to
and
of
have
in
the
had
you
The 10 nearest neighbour of good are:
good
the
you and
i
it
to
he
that
one

Out[16]:

array ([106,   0,   11,    1,    2,    8,    4,   10,    7,  45], dtype=int64)
```

First, we get the embed matrix from the model. Then we can send some words to function and find their first k nearest neighbors.

```
In [17]: #STEP6, visualize the word2vec results
it convert matrix to diet {wordl :word_vecl, word2:word_vec2... } word_2_vec = {}
for word in data. word_to_idx. keys ():
    word_2_vec [word] = embed_matrix [data. word_to_idx [word],:]
print("First 5 word2vec: ", list (word_2_vec. items0)[: 5])
#save the word2vec results
with open('. /data/CBOW_en_wordvec. txt', ' w') as f:
    for key in data. word_to_idx. keys ():
        f. write (' \n')
        f• writelines + str (key) +          + str (word_2_vec [key]))
    f. write (' \n' )
pea = PCA(n_component s= 2)
principalcomponents = pea. fit_transform(embed_matrix)
#  reduce the word2vec size from EMBEDDING_DIM to 2 dimension
#  (wordl:(vecl, vec2),word2:(vecl, vec2)...)
word2ReduceD imens ionVec = {}
for word in data. word_to_idx. keys ():
    word2ReduceDimensionVec[word] = principalcomponents[data. word_to_idx[word],:]
#  plot 1000 words, words that have close relation will stay close to each other pit.figure(figsize=(20,
20))
count = 0
for word, wordvec in word2ReduceDimensionVec. items():
    if count < 1000:
        pit. scatter(wordvec[0], wordvec[1])
        pit. annotate(word, (wordvec[0], wordvec[1]), fontsize=10)
        count += 1
pit. show()
```

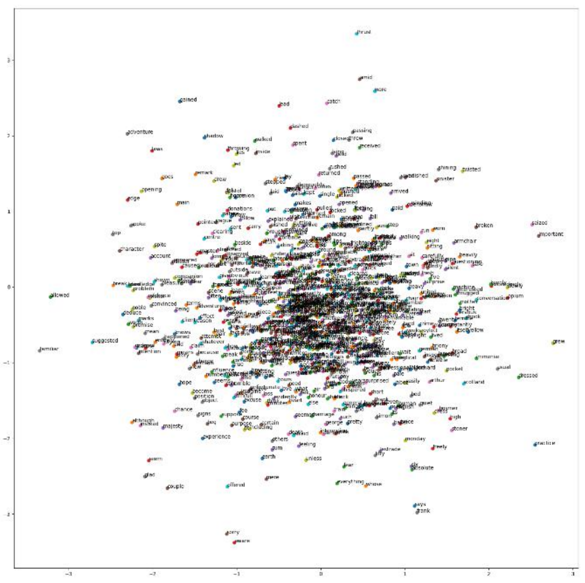
```
First 5 word2vec:   [('the', array ([ 0.        ,0. 11144606, -0. 08233763, 0.3009
5506,   0. 14033234,
         0.15400892,   0.           -0.14218096,  -0.03888544,    0. 19191384,
         0.          -0.28852928,    0.3273135 ,  -0.06072379,  -0.11118877,
         0.065249 ,    0.25723365,  -0.02940894,  -0.428934 ,   -0. 04831664,
        -0.          -0. 00566144,  -0.0966971 ,    0.07548676,  -0.2191062 ,
        -0.          -0. 01490634,   0.1380332 ,    0.23239295,  -0.0863198 ,
        -0.            0.            0.02826435,    0.           -0. 12287671,
         0.            0.            0.1272434 ,  -0. 22533551,  -0. 06378961,
        -0.          -0.33992273,  -0.23208304,  -0.08350085,  -0.670814 ,
         0.          -0.082219 ,    -0.06723426,    0.           -0.224698 ,
         0.6849605 ,  -0. 06430642,  -0.16752285,  -0.16051802,    0. 27072984,
         0.37000299,   0.           -0.27287275,    0.2610803 ,  -0.15704419,
         0.            0.            0.7958804 ,  -0.23858535,    0. 04279077,
        -0.            0.1210693 ,    0.15278268,    0.05631325,  -0.591776 ,
         0.          -0. 06592049,    0.43755317,  -0.08430469,  -0.07854283,
        -0.581647 ,  -0. 04671841,  -0.30519408,    0.           -0.03036803,
         0.          -0. 18048441,    0.10562635,    0.           -0.16263787,
        -0.            0.           -0.047768 ,      0.1481356 ,    0. 17884962,
```

```
In [18]: #STEP7, preparing the training and testing data for LSTM
train  x  =  np.zeros  ([len(data.  context_target),  WINDOWS_SIZE],  dtype=np.  int32)  train_y  =  np.
```

```
zeros([len(data. context_target) ], dtype=np. int32)
i = 0
for context, target in tqdm(data):
    train_x[i] = context
    train_y[i] = target
    i += 1 print('train_x shape:', train_x. shape) print C train y shape:', train_y. shape) itPseudo-
random Sequence, randomly select 1000 test data for testing test_dataset_size = 1000
test_x = np. zeros([test_dataset_size, WINDOWS_SIZE], dtype=np. int32) test_y = np.
zeros([test_dataset_size], dtype=np. int32)
np. random, seed(42) shuffled_indices = np. random, permutation(len(data)) test_indices =
shuffled_indices[:test_dataset_size] test_x = trainx[test_indices] test_y = train_y[test_indices] 100% I
```

```
■■■■■■■■■■■■■■■■■■■■■■■■■■ I 90342/90342 [00:01<00:00, 76765. 53it/ s] train_x shape:
(90342, 3) train_y shape: (90342,)
```

In our project, our testing data come from training data. So later the accuracy is not the real accuracy, that's the training data accuracy. At the following picture, those words that close to each other have close relationship.



```
In [20]:

#STEP8, preparing LSTM model, param ' weights* is from the CBOW training results model = Sequential()
model, add(Embedding(input_dim=data. vocab_size, output_dim= EMBEDDING_DIM, weights= [embed_matrix]))
model, add(LSTM(units=EMBEDDING_DIM))
model, add(Dense(units=data. vocab_size))
model, add (Activation (' softmax'))
model, compile(optimizer adam' , loss='sparse_categorical_crossentropy' )
print(model, summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 100)         667300

lstm_1 (LSTM)                (None, 100)               80400

dense_1 (Dense)              (None, 6673)              673973

activation_1 (Activation)    (None, 6673)              0

=================================================================
Total params: 1, 421, 673
Trainable params: 1, 421,673
Non-trainable params: 0
_________________________________________________________________

None
```

In the CBOW part, we generate two things, one is sequences and one is embed_matrix. Here we send the martix to LSTM model.

```
In [21]:

#get the best prediction
def best_prediction(preds, temperature=1.0):
  if temperature <= 0: return np. argmax (preds) preds = np. asarray (preds). astype (J floated ) preds
  = np. log(preds) / temperature exp_preds = np. exp(preds) preds = exp_preds / np. sum(exp_preds)
  probas = np. random, multinomial(1, preds, 1) return np. argmax (probas)

#put all best predictions is a list
def get_best_predictions(preds):
  pred_y = np.zeros ([test_dataset_size], dtype=np.int32)
  for idx, pred in enumerate(preds):
      pred_y [idx] = bestjrediction (pred)
  return predy

#give a sample of 3-words sequence and get the 4th prediction word.
def generate next(text, num_generated=1):
  word_idxs = [data. word_to_idx [word] for word in text, lower (). split ()] for i in
  range(num_generated):
    x    =    np.    array(word_idxs).    reshape(1,    3)    prediction    =    model,    predict(x)    idx    =
    best_prediction(prediction[-1], temperature=0. 7) word_idxs. append(idx)
  return ' J. join (data. idx_to_word[idx] for idx in word idxs)


In [28]:

ftSTEPIO, to get a sample 4th prediction and calculate the predict accuracy of the test dataset def
on_epoch_end(epoch, _):
  print C Generating text after epoch: %d' % epoch)
  #here we can put some 3-words sequence samples to get the prediction 4th word
  texts =[
      'my eyes tell', # my eyes tell me
      'you explain your', # you explain your process
      'hunting crop came', # hunting crop came down
      'do us some', # do us some harm
      'Then I made', # Then I made inquiries
      'he would do', # he would do nothing
  ]
  for text in texts:
    sample = generate_next(text)
    printC %s__ -> %s' % (text, sample))
  #test dataset accuracy_calculation
  pred_y = get_best_predictions(model, predict(test_x))
  print(' test accuracy score: %f' % accuracy_score(test_y, pred_y))
```

Here we feed sequences into the model for prediction, and a group of 3 words is used to predict the fourth word. Here I set 6 texts as examples, these examples are obviously observe compared to the accuracy.

```
In [29]: #STEP9, LSTM model training
model. fit(train_x, train_y, batch_size= 500, epochs=30,
callbacks=[LambdaCallback(on_epoch_end=on_ep( model, save C. /data/lstm_train_dim100_epoch30" )

Epoch 1/10

180/181 [==============>.] - ETA: 0s - loss: 2. 2054Generating text
after epoch: 0
my eyes tell___ -> my eyes tell me
you explain your___ -> you explain your and hunting crop came____ -> hunting crop came down do us some -
> do us some harm
Then I made__ -> then i made inquiries he would do -> he would do with
test accuracy_score: 0.339000 ______

181/181 [==] ==========:                          :]- 6s 34ms/step - loss: 2. 2061
Epoch 2/10 _______________________
181/181 ============
after epoch: 1                                    :]-ETA: 0s - loss: 2. 1710Generating text
my eyes tell___ -> my eyes tell me you explain your_______ -> you explain your he hunting crop came__ ->
hunting crop came down do us some -> do us some harm
Then I made__ -> then i made inquiries
```

Set the parameters here and start training.

4. Conclusion and Result

CBOW is trained to obtain a word2vec matrix, which reduces a lot of operations compared to one-hot code.

LSTM training can reduce the loss and make the prediction closer to the original text.

The accuracy of the training part of the prediction is good, but the actual prediction is very poor, only about 5% can be predicted. If the error tolerance part could be given to choose among 10 candidates, the accuracy would be improved. We think this is the direction we are pursuing for optimization.

5. Reference

[1]. http://www.360doc.com/content/20/0301/12/46886353_895906230.shtml

[2]. https://towardsdatascience.com/lstms-in-pytorch-528b0440244

[3]. https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa

[4]. https://zhuanlan.zhihu.com/p/42717426