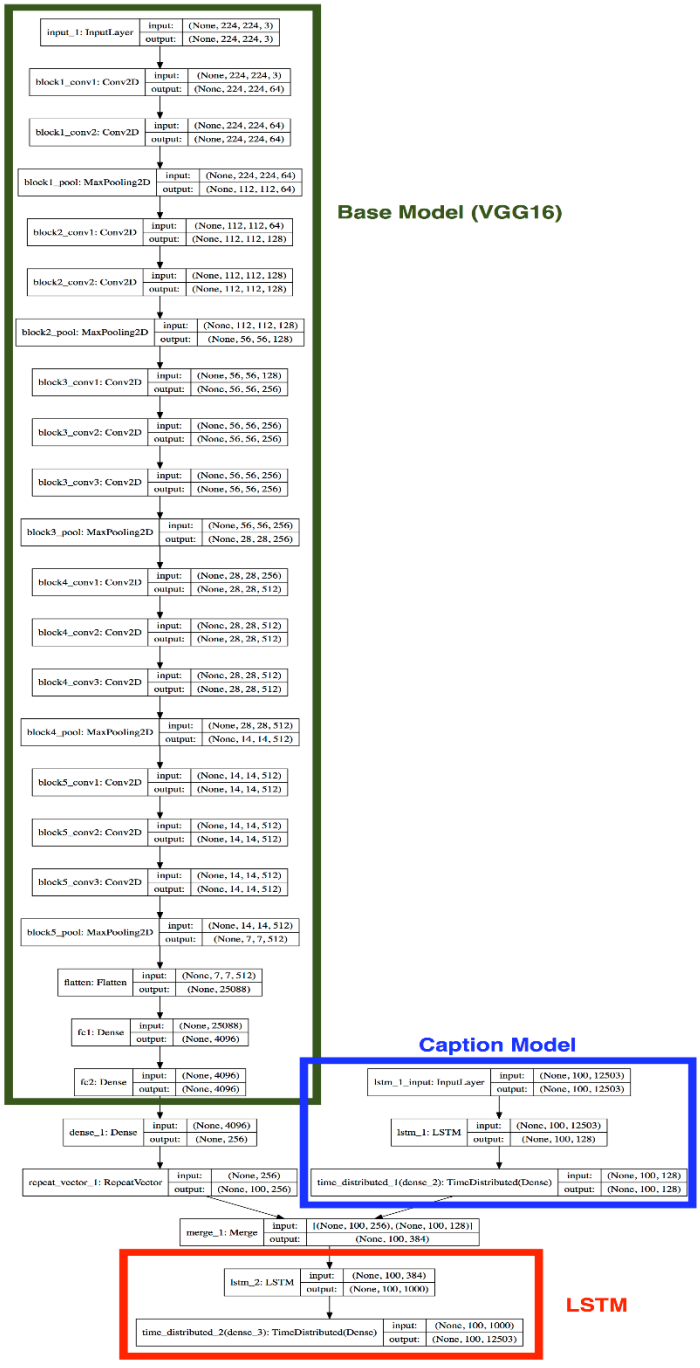
Report3: Image Caption

1. Structure of model



2. Introduction

2.1 process of training

2.1.1 Target

Training a model and generate some well-training weights.

2.1.2 process

For images, we first generate a list which contains the information of each image. We use these image as input and feed them into the next base model. We use frozen layers (VGG16, VGG19, etc.) and its weights to achieve the feature of images. These features can be merged with language model and take part in the training process.

For words, we use the sentences in train.txt as the ground true. And we can also generate a dictionary by using words frequency. Then we use LSTM to generate sentence and match the generated one with the ground true. According to this process can give a feedback and adjust the weights.

2.2 Process of generate sentence

2.2.1 Target

Add a function to generate a sentence for a given image.

2.2.2 Process

Actually, the whole process is really similar to training process. First, we should talk about generate sentence function(we will call it as generate() in the rest part of this report). In my design, the original input of this function is an image and a word(<SOS>). Then we can use model.predict() to feed the input into our model and predict the next word. I will put the index of generated word in to a list. And use this list as the next language part input.

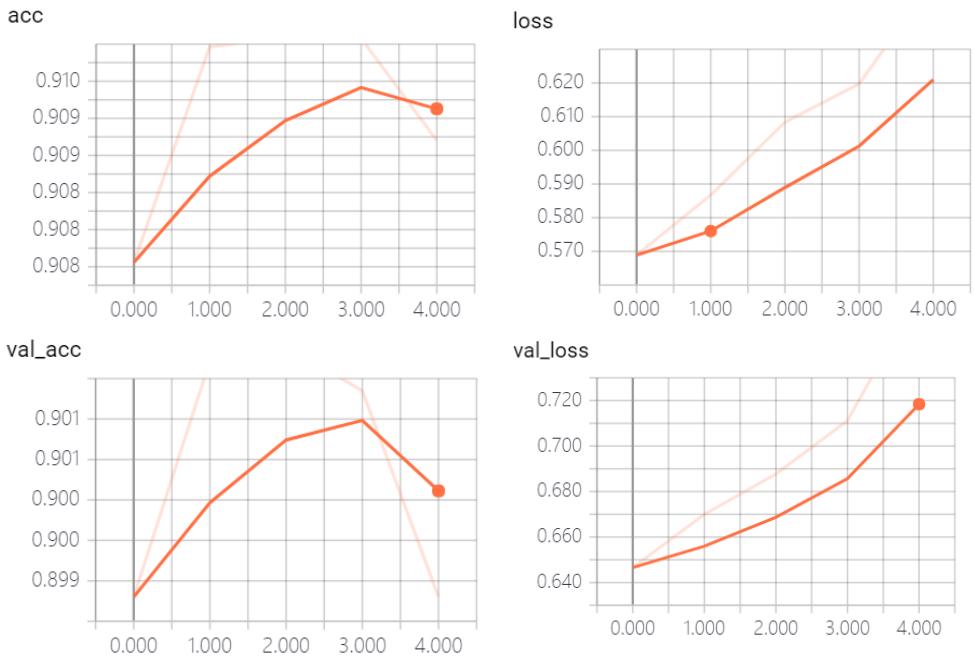
I want to explain why I said that the process is similar to training process. Because we use model.predict() which means that the input should do the same thing as training process in model. The biggest difference of those training and generating sentence is ground true and generate weight. Training need to compare the generate word with ground true and adjust weights. But generating just uses the weights and use a dictionary which generate by training processing.

3. Dataset: flickr30k

The Flickr30k dataset is a collection of over 30,000 images with 5 crowdsourced descriptions each. It is commonly used to train and evaluate neural network models that generate image descriptions. An untested assumption behind the dataset is that the descriptions are based on the images, and nothing else. Here are the authors (about the Flickr8k dataset, a subset of Flickr30k): “By asking people to describe the people, objects, scenes and activities that are shown in a picture without giving them any further information about the context in which the picture was taken, we were able to obtain conceptual descriptions that focus only on the information that can be obtained from the image alone.”

4. Improve accuracy

4.1 Original model



4.2 different base models. (refer to: keras.applications)

4.1.2 introduction

Kera's Application Module provides Keras models with pre-training weights that can be used for forecasting, feature extraction, and finetune.

Follow-up also on the following models of the parameters introduced: Xception, VGG16, VGG19, ResNet50, InceptionV3. All of these models (except Xception) are compatible with Theano and Tensorflow.

According to the top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88

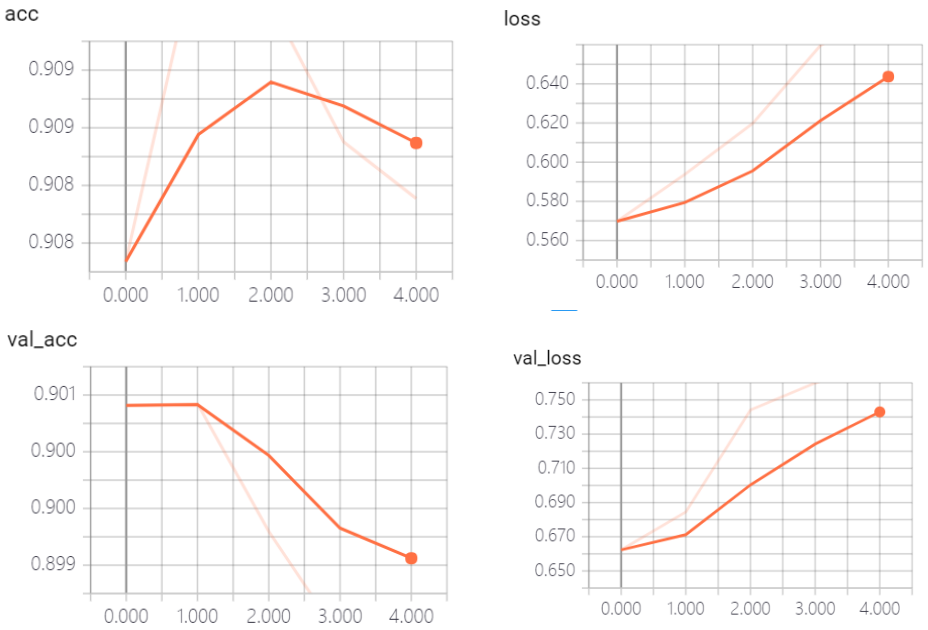
At first, I want to use Xception because of its higher accuracy. Then I found it is not compatible with tensorflow. Therefore, I choose VGG19 as a different base model instead of VGG16.

4.1.2 result

(to startup tensorboard)

```
C:\Users\dingk>tensorboard --logdir=C:/Users/dingk/desktop/cse498/assignment3/v19/log
Starting TensorBoard b'54' at http://DESKTOP-30A835J:6006
(Press CTRL+C to quit)
```

(the results on tensorboard)

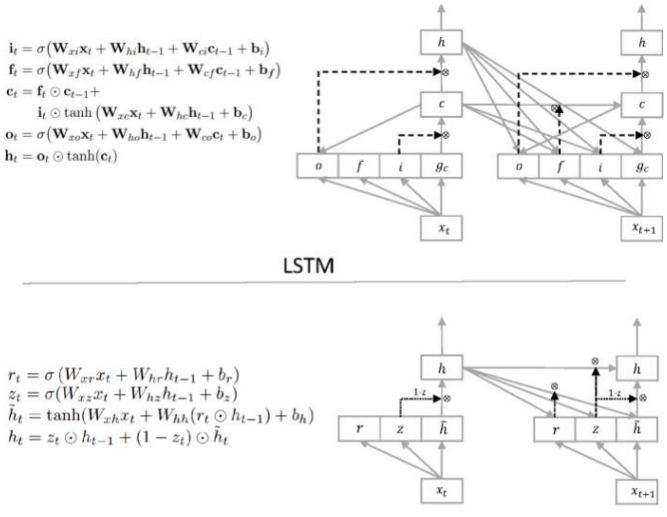


According to the accuracy and loss, there should be an overfitting.

4.3 different RNNs. (refer to: keras.layers.recurrent)

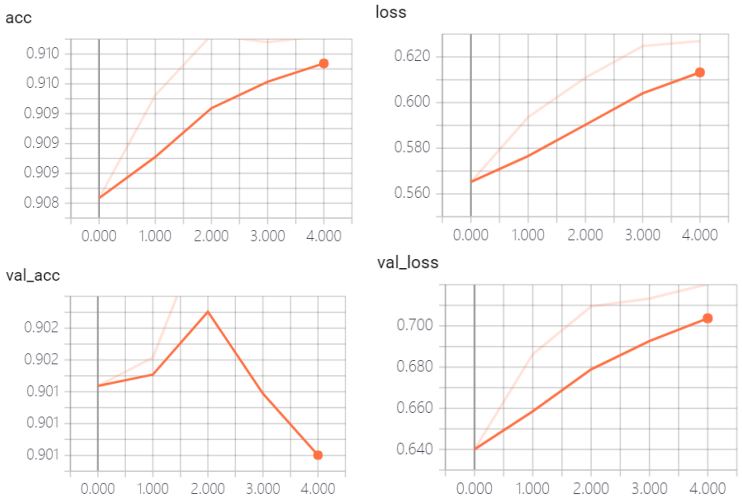
4.3.1 Introduction

There are some kinds of RNN: RNN, SimpleRNN, GRU, LSTM. In fact, LSTM and GRU all can be said to be gated hidden unit. They have a similar effects, but comparing to the LSTM, GRU has a less number of parameters. Therefore it is more difficult to overfitting.



Finally, I choose GRU as a different RNN.

4.3.2 Result



A	B	C
Wall time	Step	Value
1.51E+09	0	0.901437
1.51E+09	1	0.901615
1.51E+09	2	0.902495
1.51E+09	3	0.900618
1.51E+09	4	0.90042

We can check weights in checkpoints. It shows that the highest accuracy is epoch=3.

According to the accuracy and loss, there should be an overfitting. And it is more significant than before.

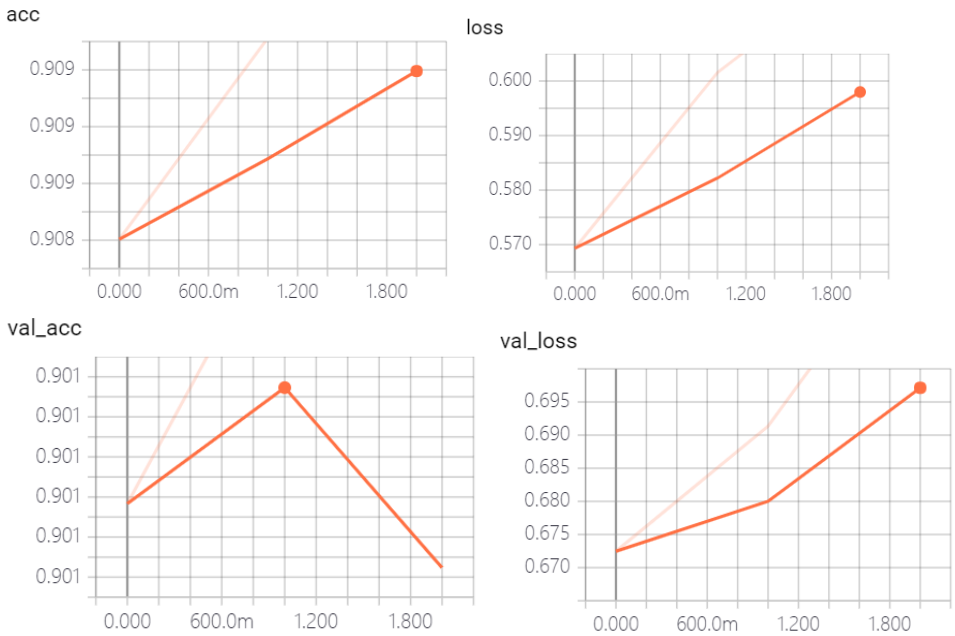
4.4 data augmentation by randomly rotating, fliping, or shifting training images.

4.4.1 Introduction

Data augmentation adds value to base data by adding information derived from internal and external sources within an enterprise. Data is one of the core assets for an enterprise, making data management essential. Data augmentation can be applied to any form of data, but may be especially useful for customer data, sales patterns, product sales, where additional information can help provide more in-depth insight.

Data augmentation can help reduce the manual interventation required to developed meaningful information and insight of business data, as well as significantly enhance data quality.

4.4.2 result



Actually, I cannot believe this is the result of data augmentation. There might be some unexpect problem of my data augmentation. Comparing to the result of just using VGG19+GRU， I think the result should be better. But in this result image, we can see there is still an overfitting. I will continue looking for the reason.

5. Image caption

5.1 design of function to generate a sentence for a given image

Now I want to use a simple pseudocode to show my algorithm of generating sentence:

```
input: image path
output: sentence
{
Image=load(image path)
First_word= '<SOS>'
Sentence=[]
word_index_list= caption2index(first_word)
for i in range of (100):
        padding(word_index_list)
        padded to onehot
        model.predict([image, onehot])
        extract new word index
        word_index_list.append(new word index)
        sentence.append(index2caption(new word index))
}
```

There is a important point that I should remind:
Because we use the model which is same as training, we should pay attention to the dimension of input.

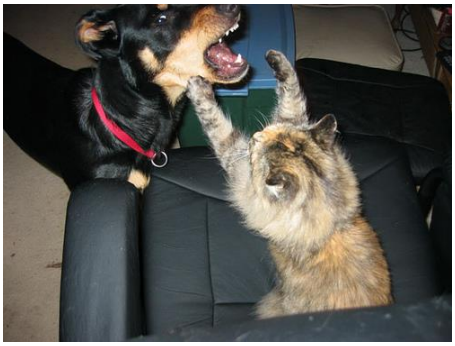
input_1: InputLayer	input:	(None, 224, 224, 3)
	output:	(None, 224, 224, 3)

lstm_1_input: InputLayer	input:	(None, 100, 12503)
	output:	(None, 100, 12503)

We should always remember, the image input of the model is 4 dimension and the word input is 3 dimension.

5.2 result

5.2.1 the process of generate the last word of a whole sentence(there can be many same steps of generate the next word. Therefore I just show the last time of the whole generate process)



```
*****input_word_index_list [[ 7529  2252  4258 10289  2143  8717  2252  6391 10935   627     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0]]
*****input_word_index_list (1, 100)
(100, 12503)
******input_onehot [[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 1.  0.  0. ...,  0.  0.  0.]
 [ 1.  0.  0. ...,  0.  0.  0.]
 [ 1.  0.  0. ...,  0.  0.  0.]]
******input_onehot_shape (100, 12503)
*****predict [[[  1.06752607e-08   5.02590691e-10   7.22348681e-10 ...,   1.64962954e-09
     2.54122284e-07   2.90324143e-09]
  [  2.03292097e-06   1.27854207e-07   1.48802528e-07 ...,   1.91294504e-07
     6.31532941e-07   6.09204903e-07]
  [  2.86321028e-06   3.57594203e-08   5.53126398e-08 ...,   4.04547826e-08
     3.70372400e-05   3.32472695e-07]
  ...,
  [  1.00000000e+00   1.84062638e-18   3.62014023e-18 ...,   7.14358347e-20
     4.30634435e-25   1.15464125e-18]
  [  1.00000000e+00   1.83828256e-18   3.61704823e-18 ...,   7.12926355e-20
     4.26735884e-25   1.15234428e-18]
  [  1.00000000e+00   1.83592799e-18   3.61387600e-18 ...,   7.11529906e-20
     4.22985590e-25   1.15005195e-18]]]
*****predict_shape (1, 100, 12503)
*****output_word_line [[  8.72819417e-09   9.33126633e-17   1.33246250e-16 ...,   1.30900525e-16
    3.48663867e-14   2.52914827e-16]]
*****output_word_line (1, 12503)
```

```
*****output_word_line [[  8.72819417e-09   9.33126633e-17   1.33246250e-16 ...,   1.30900525e-16
    3.48663867e-14   2.52914827e-16]]
*****output_word_line (1, 12503)
*****output_word_list [[8.728194167417769e-09, 9.331266333277585e-17, 1.3324625000676155e-16, 2.66558346
*****index 592
*****predict_max 1.0
*****predict_input_word_index_list [[7529, 2252, 4258, 10289, 2143, 8717, 2252, 6391, 10935, 627, 592]]
*****word ['<SOS> a dog is running through a grassy field . <EOS>']
input_index_list****** [[7529, 2252, 4258, 10289, 2143, 8717, 2252, 6391, 10935, 627, 592]]
sentence_list******* ['<SOS> a dog is running through a grassy field . <EOS>']
```

```
input_index_list****** [[7417, 8468, 1993, 623, 8468, 7173, 7471, 435, 11452, 623, 4940, 6707, 8468, 1015, 1113]]
sentence_list******* ['<SOS> a man in a blue shirt is standing in front of a building .']
*****input_word_index_list [[ 7417  8468  1993   623  8468  7173  7471   435 11452   623  4940  6707
   8468  1015  1113     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0]]
*****input_word_index_list (1, 100)
(100, 12503)
******input_onehot [[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 1.  0.  0. ...,  0.  0.  0.]
 [ 1.  0.  0. ...,  0.  0.  0.]
 [ 1.  0.  0. ...,  0.  0.  0.]]
******input_onehot_shape (100, 12503)
*****predict [[[  6.12803660e-22   8.81849040e-28   1.55123442e-26 ...,   5.85287013e-33
     5.17555359e-33   3.70822682e-32]
  [  9.62113566e-23   5.19842340e-28   1.11776475e-26 ...,   6.75573317e-34
     5.70820103e-34   6.32151621e-33]
  [  4.09939635e-22   5.08694066e-27   3.53863009e-25 ...,   1.10231031e-31
     1.05470866e-31   8.63588021e-31]
  ...,
  [  1.00000000e+00   1.16335635e-24   2.05124014e-23 ...,   2.78190339e-24
     2.14212457e-24   1.22977672e-23]
  [  1.00000000e+00   2.09719889e-26   1.47903516e-26 ...,   1.14538127e-24
     9.15759043e-25   1.41498316e-24]
  [  1.00000000e+00   6.37896902e-26   1.99428080e-25 ...,   3.76604587e-25
     3.27426999e-25   1.19334569e-24]]]
*****predict_shape (1, 100, 12503)
*****output_word_line [[  1.34314876e-31   6.61987116e-37   9.95775872e-36 ...,   0.00000000e+00
    0.00000000e+00   0.00000000e+00]]
*****output_word_line (1, 12503)
```

```
*****output_word_line (1, 12503)
*****output_word_list [[1.343148762410576e-31, 6.6198711606160625e-37, 9.957758721651674e-36, 0.0, 0.0, 3.976429620001333e-19, 0.0,
*****index 5129
*****predict_max 1.0
*****predict_input_word_index_list [[7417, 8468, 1993, 623, 8468, 7173, 7471, 435, 11452, 623, 4940, 6707, 8468, 1015, 1113, 5129]]
*****word ['<SOS> a man in a blue shirt is standing in front of a building . <EOS>']
input_index_list****** [[7417, 8468, 1993, 623, 8468, 7173, 7471, 435, 11452, 623, 4940, 6707, 8468, 1015, 1113, 5129]]
sentence_list******* ['<SOS> a man in a blue shirt is standing in front of a building . <EOS>']
```

5.2.2 analysis

the first image is using xin_weight.hdf5 and the second image is using my own training weight. And there is a really strange thing: using my own weight can only generate one sentence no matter I use which image. And using xin's weight, the model can only identify dog and people. I think there may be some problem in my training model. Maybe it shows the superiority of LSTM. Even if GRU is more difficult to overfitting than LSTM, the ability of learning maybe be weaken at the same time.

5.3 some unexpected problems of version

5.3.1 the different usage of zip in python2 and python3

we can use zip() directly to do matrix transpose in python2. But we cannot do this in python3, we should write a loop(for) so that we can see the result of using zip.

(result we can get in python3 when we using zip())

```
<zip object at 0x000001F8ADAA91C8>
```

(solution)

```
for index, zip_word_index_list in enumerate(zip(input_word_index_list_pad),
                                                start=1):
    input_onehot = self.convert2onehot(zip_word_index_list[0])
    print(input_onehot.shape)
```

We see the results of zip() as a list, and then we do a traversal for this list so that we can effectively solve the problem, and see the a normal result.

5.3.2 decode error

There is still some important thing I want to write down as a record. The problem of encoding. The words.txt, test.txt and train.txt may happen some encoding problems on windows. You can just add two commands to avoid this problem:

```
UnicodeDecodeError: 'gbk' codec can't decode byte 0x9d in position 4499: illegal multibyte sequence
```

(Solution)

```
with open(WORDS_PATH, 'r',encoding='utf-8',errors='ignore') as reader:
        words = [x.strip() for x in reader.readlines()]
```

When you add: encoding='utf-8',errors='ignore', this problem can be solved quickly.

6. summary

To be honest, this is really a difficult assignment for me. I spent about 2 weeks on this assignment. One week was used to understand the whole model and another week was used to programming and debug. I found that I still has a limited programming and debug skill. I spent so much time on debug.

I have a question about grammar before. After this assignment, I am sure that I have understand LSTM. It can generate sentence just because it had read too much sentence. At first, I don't know how to finish this assignment. Therefore, I read some material about image caption. I knew about NLP. I think it should be a more intelligent and robust way than LSTM. I think this way can be called a real way of generating. I have no time to learn more about it. But I imagined if we can use NLP to generate sentence? It can be another interesting issue. I hope I can learn it by myself in future.

What's more, I learnt how to use tensorboard to show the accuracy and loss. I understand the reason of the low grade of my last two assignment. It is really a limited work what I did in the last two assignments. I also learnt how to use HPC and WINscp. WINscp can be an useful tool for windows users when we need to upload files to HPC.