DataLab Cup 2: Image Captioning 喔~~原來可以醬做

- Modified model: Make the model more complexity, GRU(512); add dropout layer; accept word position input
- Training strategy: Try curricular learning; speedup generate batch function
- Feature extraction: Capture the position of current word; reduce the image feature dimension to 512 or 1024 instead of 256

In [1]:
```
import os
import _pickle as cPickle
import urllib.request

import pandas as pd
import scipy.misc
import numpy as np

from keras.models import Model
from keras.layers import Input, Dense, Embedding, Reshape, GRU, merge,D
ropout, TimeDistributed, LSTM
from keras.optimizers import RMSprop
from keras.models import load_model
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from IPython.display import Image, display, SVG

from pre_trained.cnn import PretrainedCNN

%matplotlib inline
output_notebook()
```

```
Using Theano backend.
Using gpu device 0: GeForce GTX 1070 (CNMeM is disabled, cuDNN 5105)
```

Loading BokehJS ...

Preprocess: Text

In [2]:
```
vocab = cPickle.load(open('dataset/text/vocab.pkl', 'rb'))
print('total {} vocabularies'.format(len(vocab)))
```

```
total 26900 vocabularies
```

In [3]:
```
def count_vocab_occurance(vocab, df):
    voc_cnt = {v:0 for v in vocab }

    for img_id, row in df.iterrows():
        for w in row['caption'].split(' '):
            voc_cnt[w] += 1
    return voc_cnt
```

```
df_train = pd.read_csv(os.path.join('dataset', 'train.csv'))

print('count vocabulary occurances...')
voc_cnt = count_vocab_occurance(vocab, df_train)

# remove words appear < 100 times
thrhd = 100
x = np.array(list(voc_cnt.values()))
print('{} words appear >= 100 times'.format(np.sum(x[(-x).argsort()] >=
thrhd)))
def build_voc_mapping(voc_cnt, thrhd):
    """
    enc_map: voc --encode--> id
    dec_map: id --decode--> voc
    """
    def add(enc_map, dec_map, voc):
        enc_map[voc] = len(dec_map)
        dec_map[len(dec_map)] = voc
        return enc_map, dec_map
    # add <ST>, <ED>, <RARE>
    enc_map, dec_map = {}, {}
    for voc in ['<ST>', '<ED>', '<RARE>']:
        enc_map, dec_map = add(enc_map, dec_map, voc)

    for voc, cnt in voc_cnt.items():

        if cnt < thrhd: # rare words => <RARE>
            enc_map[voc] = enc_map['<RARE>']
        else:
            enc_map, dec_map = add(enc_map, dec_map, voc)
    return enc_map, dec_map

enc_map, dec_map = build_voc_mapping(voc_cnt, thrhd)
#print(enc_map)
#print(dec_map)
# save enc/decoding map to disk
cPickle.dump(enc_map, open('dataset/text/enc_map.pkl', 'wb'))
cPickle.dump(dec_map, open('dataset/text/dec_map.pkl', 'wb'))
vocab_size = len(dec_map)
print( vocab_size)
def caption_to_ids(enc_map, df):
    img_ids, caps = [], []
    for idx, row in df.iterrows():
        icap = [enc_map[x] for x in row['caption'].split(' ')]
        icap.insert(0, enc_map['<ST>'])
        icap.append(enc_map['<ED>'])
        img_ids.append(row['img_id'])
        caps.append(icap)
    return pd.DataFrame({'img_id':img_ids, 'caption':caps}).set_index(['
img_id'])


def caption_to_ids(enc_map, df):
    img_ids, caps = [], []
    for idx, row in df.iterrows():
        icap = [enc_map[x] for x in row['caption'].split(' ')]
        icap.insert(0, enc_map['<ST>'])
        icap.append(enc_map['<ED>'])
        img_ids.append(row['img_id'])
        caps.append(icap)
    return pd.DataFrame({'img_id':img_ids, 'caption':caps}).set_index(['
```

```
img_id'])


enc_map = cPickle.load(open('dataset/text/enc_map.pkl', 'rb'))
print('[transform captions into sequences of IDs]...')
df_proc = caption_to_ids(enc_map, df_train)
df_proc.to_csv('dataset/text/train_enc_cap.csv')
# id to word
def decode(dec_map, ids):
    return ' '.join([dec_map[x] for x in ids])

dec_map = cPickle.load(open('dataset/text/dec_map_4B.pkl', 'rb'))

print('And you can decode back easily to see full sentence...\n')
for idx, row in df_proc.iloc[:8].iterrows():
    print('{}: {}'.format(idx, decode(dec_map, row['caption'])))
```

```
count vocabulary occurances...
2184 words appear >= 100 times
2187
[transform captions into sequences of IDs]...
[transform captions into sequences of IDs]...
And you can decode back easily to see full sentence...

536654.jpg: <ST> baggage above close stools cycle not very baggage hole
toothbrushes baggage bat close with <ED>
536654.jpg: <ST> stools cycle logs laid very baggage hole dishwasher <ED
>
536654.jpg: <ST> stools cycle cross laid very baggage hole giant baggage
with his <ED>
536654.jpg: <ST> bicycle pulling sand cat cross middle laid very baggage
with his <ED>
536654.jpg: <ST> stools cycle logs granite laid very baggage middle with
potted <ED>
15839.jpg: <ST> baggage comes window sails rolls baggage bag skateboard
tall wires <ED>
15839.jpg: <ST> baggage upright sails reflecting not uncooked close bagg
age bag <ED>
15839.jpg: <ST> upright sails rolls wetsuit bag close below rural <ED>
```

In [7]:

```
#use a vector to represent a word
def generate_embedding_matrix(w2v_path, dec_map, lang_dim=200):
    out_vocab = []
    embeddings_index = {}
    f = open(w2v_path, 'r')
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    f.close()
    # prepare embedding matrix
    embedding_matrix = np.random.rand(len(dec_map), lang_dim)
    for idx, wd in dec_map.items():
        if wd in embeddings_index.keys():
            embedding_matrix[idx] = embeddings_index[wd]
        else:
            out_vocab.append(wd)
    print('words: "{}" not in pre-trained vocabulary list'.format(','.j
oin(out_vocab)))
    return embedding_matrix
```

```
dec_map = cPickle.load(open('dataset/text/dec_map_4B.pkl', 'rb'))
embedding_matrix = generate_embedding_matrix('pre_trained/glove.6B.100d.
txt', dec_map, lang_dim = 100)
print ("embedding shape",embedding_matrix.shape)
```

```
words: "<ST>,<ED>,<RARE>,selfie,skiis" not in pre-trained vocabulary lis
t
embedding shape (2187, 100)
```

The embedding matrix gives a pre-trained vector to represent a single. At the begining, we used a 100-dimension vector to describe. To remain more information for a single word, we then implemented 200-dimension version. The performance seemed to be improved but limited. On the other hand, it took more time on training the model since we made the featrue more complexity. Considering the trade-off between time and tiny improvement, finally, we used the 100-dimension version word2vec.

Preprocess: Image

In [6]:
```
#load 抽好的 image feature...
img_train = cPickle.load(open('dataset/train_img256.pkl', 'rb'))
img_test = cPickle.load(open('dataset/test_img256.pkl', 'rb'))
```

After tuning parameter and modifying our model several times, we found out that the improvement of the score was limited. Therefore, We then tried to re-extract the image feature by VGG16 and use PCA to reduce the dimension. This time the dimension was reduced to 512 and 1024, which were larger then the original one.

Generate training data

In [8]:
```
#generate batch consider position
import math
### Original version ###
def generate_batch(img_map, df_cap, vocab_size, size=32):
    imgs, curs, nxts, position = None, [], None, None

    for idx in np.random.randint(df_cap.shape[0], size=size):
        row = df_cap.iloc[idx]
        cap = eval(row['caption'])
        if row['img_id'] not in img_map.keys():
            continue
        img = img_map[row['img_id']]
        for i in range(1, len(cap)):
            nxt = np.zeros((vocab_size))
            nxt[cap[i]] = 1
            now_position = [ 0 for i in range(52)]
            now_position[i-1] = 1

            curs.append(cap[i-1])
            position =  now_position if position is None else np.vstack([
position , now_position])
            nxts = nxt if nxts is None else np.vstack([nxts, nxt])
            imgs = img if imgs is None else np.vstack([imgs, img])
    return imgs, np.array(curs).reshape((-1,1)), nxts , position
```

```
### Modified Version ###
def __generate_batch3(img_map, df_cap, vocab_size, index_arr):
    arr_len = len(index_arr)
    if(arr_len<32):
        imgs, curs, nxts, positions = None, [], None, None
        for idx in index_arr:
            row = df_cap.iloc[idx]
            d = row['img_id']
            if d not in img_map.keys():
                continue
            cap = eval(row['caption'])
            img = np.repeat([img_map[row['img_id']]], [len(cap)-1], axis
=0)
            curs.extend(cap[0:len(cap)-1])
            nxt = np.zeros((len(cap)-1, vocab_size))
            position = np.zeros((len(cap)-1, 52))
            for i in range(1, len(cap)):
                nxt[i-1][cap[i]] = 1
                position[i-1][i-1] = 1
            nxts = nxt if nxts is None else np.vstack([nxts, nxt])
            imgs = img if imgs is None else np.vstack([imgs, img])
            positions = position if positions is None else np.vstack([po
sitions , position])
    else:
        m = arr_len >> 1
        imgs, curs, nxts, positions = __generate_batch3(img_map, df_cap,
vocab_size, index_arr[0:m])
        img, cur, nxt, position = __generate_batch3(img_map, df_cap, voc
ab_size, index_arr[m:arr_len])
        positions = np.vstack([positions, position])
        nxts = np.vstack([nxts, nxt])
        imgs = np.vstack([imgs, img])
        curs.extend(cur)
    return imgs, curs, nxts, positions

def generate_batch3(img_map, df_cap, vocab_size, size=32, random=True,
rnd=1):
    if random:
        index_arr = np.random.randint(df_cap.shape[0], size=size)
    else:
        lf = (rnd-1) * size
        rt = (rnd) * size
        if lf >= df_cap.shape[0]:
            raise Exception('QQQ')
        if rt > df_cap.shape[0]:
            rt = df_cap.shape[0]
        index_arr = range(lf, rt)
    imgs, curs, nxts, positions = __generate_batch3(img_map, df_cap, voc
ab_size, index_arr)
    return imgs, np.array(curs).reshape((-1,1)), nxts, positions

def generate_batch3_curriculum(img_map, df_cap, vocab_size, size=100000
, partition=50, rnd=0):
    rate = 1 / partition
    lf = math.floor(rate*rnd*size)
    rt = math.floor(rate*(rnd+1)*size)
    index_arr = range(lf, rt)
    imgs, curs, nxts, positions = __generate_batch3(img_map, df_cap, voc
ab_size, index_arr)
    return imgs, np.array(curs).reshape((-1,1)), nxts, positions
```

Our model requires **the position of the current input word in a caption**. Therefore, we adapted the original generate_batch function. The caption with max sentence length has 52 words, so the extra input is a 52-dimension one-hot vector (The distribution of caption length is shown below), which records whether the current word is on a centain position. **This vector somehow offers the information about occurance probability of single word in different position. After experimenting, it did boost our performance, from 0.85 to 0.80.** Moreover, since it took lots of time to generate the image-caption pairs while training, **speedup** was required!

generate batch accelerating

To deal with the time consuming while generate the training samples, we modified the generate_batch function. Obviously, it is the process to generate the "nxts" matrix that make the generate_batch function inefficient. In the original for-loop, the vstack function will merge two matrices. Therefore, the time complexity for a single merge is sum of the elements of the two matrices. After preciously computing, the time complexity of merging the "nxts" matrix is

$$O(\;\sum\limits_{idx=0}^{size-1}\sum\limits_{i=1}^{len(cap[idx])-2}\{vocab\_size [(\sum\limits_{p=0}^{idx-1}\sum\limits_{q=1}^{len(cap[p])-2}1)+(i-1)]+vocab\_size)\}\;)$$
For instance, the time complexity of a caption with 12 word length (including and) is

$$ \begin{array}{rl} & O(\;\sum\limits_{idx=0}^{size-1}\sum\limits_{i=1}^{12-2}\{vocab\_size [(\sum\limits_{p=0}^{idx-1}\sum\limits_{q=1}^{12-2}1)+(i-1)]+vocab\_size)\}\;) \\ = & O(\;\sum\limits_{idx=0}^{size-1}\sum\limits_{i=1}^{10}vocab\_size(10\,idx+i)\;) \\ = & O(\;\sum\limits_{idx=0}^{size-1}vocab\_size\,(100\,idx+55)\;)\\ = & O(\;vocab\_size[50\,size\,(size-1)+55\,size]\;)\\ \end{array}$$

That is, the original time complexity is quadratic time with respect to $size$. Besides, the large constant (about 50*$vocab\_size$=50*2187=109350) in the for mula decrease the efficiency dramastically. Therefore, in order to speedup, reducing the use of vstack function was our priority. **Here we call the function at outer for-loop instead of inner one.**

In [9]:
```
def generate_batch2(img_map, df_cap, vocab_size, size=32):
    imgs, curs, nxts = None, [], None
    #for idx in np.random.randint(df_cap.shape[0], size=size):  # 真正在run的時候要把這行註解回覆
    for idx in range(size):   # 真正在run的時候要把這行刪掉(否則不會隨機抽樣)
        row = df_cap.iloc[idx]
        d = row['img_id']
        if d not in img_map.keys():
            continue
        cap = eval(row['caption'])
        img = np.repeat([img_map[row['img_id']]], [len(cap)-1], axis=0)
        curs.extend(cap[0:len(cap)-1])
        nxt = np.zeros((len(cap)-1, vocab_size))
        for i in range(1, len(cap)):
            nxt[i-1][cap[i]] = 1
        nxts = nxt if nxts is None else np.vstack([nxts, nxt])
        imgs = img if imgs is None else np.vstack([imgs, img])
    return imgs, np.array(curs).reshape((-1,1)), nxts
```

After the optimization, by calculating the time complexity of above example using modified generate_batch, it became

$$ \begin{array}{rl} & O(\;\sum\limits_{idx=0}^{size-1}10\,vocab\_size\,(idx+1)\;)\\ = & O(\;vocab\_size[5\,size\,(size+1)]\;)\\ \end{array}$$

We could find out that the time complexity is still quadratic time with respect to $size$. **However, the constant become one-tenth of the former one (about 5*$vocab\_size$ = 5*2187=10935)**.

Eventhough the efficiency was highly improved, we still not satisfied with it because it took about 20 minutes to generate 5000 training samples. To reducing the annoying waiting time, we then kept optimize it !!

The "nxts" matrix is accumulated by a single "nxt" matrix with different "idx". According to the current method, once a "nxt" matrix is done, we append it to "nxts" matrix and ;that is, we have to copy the huge "nxts" matrix just for adding a row, which is inefficient ! Therefore, we maintained the balance of two matrics when merging to avoid time wasting. **Briefly, we optimized generate_batch2 function by implementing devide-and-conquer.**

In [10]:
```
#Final version generate batch
def __generate_batch3(img_map, df_cap, vocab_size, index_arr):
    arr_len = len(index_arr)
    if(arr_len<32):
        imgs, curs, nxts = None, [], None
        for idx in index_arr:
            row = df_cap.iloc[idx]
            d = row['img_id']
            if d not in img_map.keys():
                continue
            cap = eval(row['caption'])
            img = np.repeat([img_map[row['img_id']]], [len(cap)-1], axis
=0)
            curs.extend(cap[0:len(cap)-1])
            nxt = np.zeros((len(cap)-1, vocab_size))
            for i in range(1, len(cap)):
                nxt[i-1][cap[i]] = 1
            nxts = nxt if nxts is None else np.vstack([nxts, nxt])
            imgs = img if imgs is None else np.vstack([imgs, img])
    else:
        m = arr_len >> 1
        imgs, curs, nxts = __generate_batch3(img_map, df_cap, vocab_size
, index_arr[0:m])
        img, cur, nxt = __generate_batch3(img_map, df_cap, vocab_size, i
ndex_arr[m:arr_len])
        nxts = np.vstack([nxts, nxt])
        imgs = np.vstack([imgs, img])
        curs.extend(cur)
    return imgs, curs, nxts

def generate_batch3(img_map, df_cap, vocab_size, size=32, random=True,
rnd=1):
    if random:
        # index_arr = np.random.randint(df_cap.shape[0], size=size)
        index_arr = range(size)
    else:
        lf = (rnd-1) * size
```

```
        rt = (rnd) * size
        if lf >= df_cap.shape[0]:
            raise Exception('QQQ')
        if rt > df_cap.shape[0]:
            rt = df_cap.shape[0]
        index_arr = range(lf, rt)
    imgs, curs, nxts = __generate_batch3(img_map, df_cap, vocab_size, in
dex_arr)
    return imgs, np.array(curs).reshape((-1,1)), nxts
```

To implement divide-and-conquer, we first recorded the "idx"s and stored them into a "index_arr". Secondly, divided the the "index_arr" list equally and do the same thing recursively untill the generate_batch function can generate "nxts" in a short time. The we set the threhold of "indel_arr" length to 32.

After all, since the two matrics have almost same size, the time spending on duplicating elements would be reduce.

In conclusion, the depth of the binary tree would have maximum $\lceil\,\log_{2}size\,\rceil$ level. Let the root be 0, there would be maximum 2^{k} times of merging in the k_{th} level. In addition, the time complexity of every single merging in a certain level would not excess $O(\;10\,vocab\_size (\frac{size}{2^{k+1}}+\frac{size}{2^{k+1}})\;)$. Therefore, total time complexity become $O(\;\sum\limits_{k=0}^{\lceil\,\log_{2}\,size\,\rceil-1}\,[2^{k}\cdot(10\,vocab\_size\,\cdot\frac{size}{2^{k}})]\;) = O(10\,vocab\_size \cdot size \cdot \,\log_{2}size)$

So far, **we have already optimized the function to the time complexity of log-linear with respect to $size$.** Basically, it can generate 50000 samples in one minute which is a significant improvement !

In [12]:
```
df_cap = pd.read_csv('dataset/text/train_enc_cap_4B.csv')
%time img1, cur1, nxt1 = generate_batch3(img_train, df_cap, vocab_size,
size=50000)
```

```
CPU times: user 57.1 s, sys: 6.47 s, total: 1min 3s
Wall time: 1min 7s
```

Generate 50000 samples took about one minute.

RNN model

In [3]:
```
def image_caption_model(vocab_size=2187, embedding_matrix=None, lang_di
m=100, img_dim=256,position_dim = 52 , clipnorm=1, lr=1e-3):
    # text: current word
    lang_input = Input(shape=(1,))
    position_input  = Input(shape = (52,))
    print(lang_input)
    if embedding_matrix is not None:
        x = Embedding(output_dim=lang_dim, input_dim=vocab_size, init='g
lorot_uniform', input_length=1, weights=[embedding_matrix])(lang_input)
    else:
        x = Embedding(output_dim=lang_dim, input_dim=vocab_size, init='g
lorot_uniform', input_length=1)(lang_input)

    lang_embed = Reshape((lang_dim,))(x)
```

```
    # img
    img_input = Input(shape=(img_dim,))
    # text + img => GRU
    x = merge([img_input, lang_embed,position_input], mode='concat', con
cat_axis=-1)
    x = Dropout(0.25)(x)
    x = Reshape((1, lang_dim+img_dim+position_dim))(x)
    x = GRU(512)(x)

    # predict next word
    out = Dense(vocab_size, activation='softmax')(x)
    model = Model(input=[img_input, lang_input,position_input], output=
out)
    # choose objective and optimizer
    model.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=
lr, clipnorm=clipnorm))
    return model

model = image_caption_model()
model.summary()
```

```
input_4
____________________________________________________________________________
_____
Layer (type)                     Output Shape          Param #     Connect
ed to
============================================================================
=============================
input_4 (InputLayer)             (None, 1)             0

____________________________________________________________________________
_____
embedding_2 (Embedding)          (None, 1, 100)        218700      input_
4[0][0]
____________________________________________________________________________
_____
input_6 (InputLayer)             (None, 256)           0

____________________________________________________________________________
_____
reshape_3 (Reshape)              (None, 100)           0           embeddi
ng_2[0][0]
____________________________________________________________________________
_____
input_5 (InputLayer)             (None, 52)            0

____________________________________________________________________________
_____
merge_2 (Merge)                  (None, 408)           0           input_6
[0][0]
                                                                   reshape_3
[0][0]
                                                                   input_5[0
][0]
____________________________________________________________________________
_____
reshape_4 (Reshape)              (None, 1, 408)        0           merge_2
[0][0]
____________________________________________________________________________
_____
gru_2 (GRU)                      (None, 1024)          4402176     reshape
```

```
_4[0][0]
__________________________________________________________________________
_____________________________
dense_2 (Dense)                 (None, 2187)         2241675     gru_2[0
][0]
==========================================================================
============================
Total params: 6862551
__________________________________________________________________________
_____________________________
```

Our RNN model has following slightly change.

1. Extra input: position of current word, which makes the sentence more grammatical
2. Dropout layer with 75% remain
3. Increas input size of Gated Recurrent Unit layer (512 dim).

Those changes did improve our performance.

Training

In [ ]:

```
learning_rate = 0.0005
clipping = 1
batch_size = 3000
print ("learning rate: ", learning_rate)
print ("clipping: ", clipping)
print ("batch size: ", batch_size)
model = image_caption_model(vocab_size=vocab_size, embedding_matrix=emb
edding_matrix
                            ,clipnorm=clipping
                            ,lr = learning_rate
                           ,lang_dim = 100
                            ,img_dim=1024
                           )


hist_arr = []
epoch =8
print ("epoch: ",epoch)
# model = load_model('model_ckpt/curriculum.h5')
for j in range(epoch):
    for i in range(50):
        img1, cur1, nxt1, position1 = generate_batch3(img_train, df_cap,
vocab_size, size=10000,random=True)
#         img1, cur1, nxt1, position1 = generate_batch3_curriculum(img_t
rain, df_cap, vocab_size, partition=50, size=500000,rnd=i)
        hist = model.fit([img1, cur1,position1], nxt1, batch_size = batc
h_size, nb_epoch=1,
                         verbose=0,validation_data=([img2, cur2,position2
], nxt2)
                        , shuffle=True
                    #,callbacks=[early_stop]
                        )
    print ("iter: ",j+1," loss: ",hist.history["loss"]," val_loss " ,hi
st.history["val_loss"])
    hist_arr.append(hist)
    del img1, cur1, nxt1,position1
```

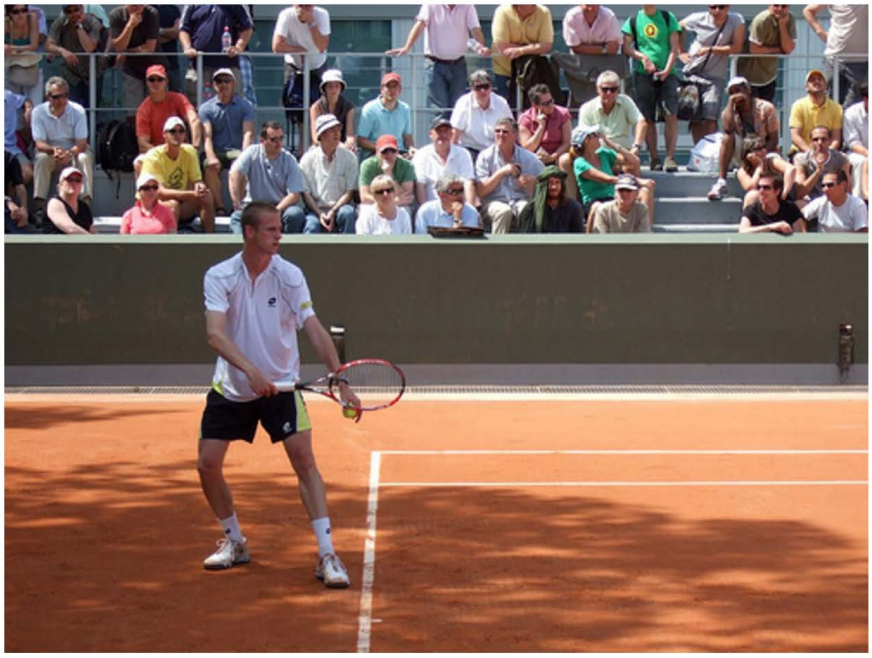
```
loss =[]
val_loss = []
for j in range(epoch):
    loss += hist_arr[j].history['loss']
    val_loss += hist_arr[j].history['val_loss']
hist_path, mdl_path = 'model_ckpt/demo.pkl', 'model_ckpt/demo.h5'
cPickle.dump({'loss':loss, 'val_loss':val_loss}, open(hist_path, 'wb'))
model.save(mdl_path)
```

In term of training strategy, it is impossible to load all the samples into memory. Therefore, we ran the generate_batch function everytime before training and used for-loop to obtain all the 500000 plus samples. At first, we tuned the parameters to overfit the small data. However, once we loaded bigger data, the performance was under expectation. Thus we tuned parameters on bigger-size data (about 60000 samples). **By observing the descent situation within just a first few epochs, we could easily determine whether the parameters are suitable.**

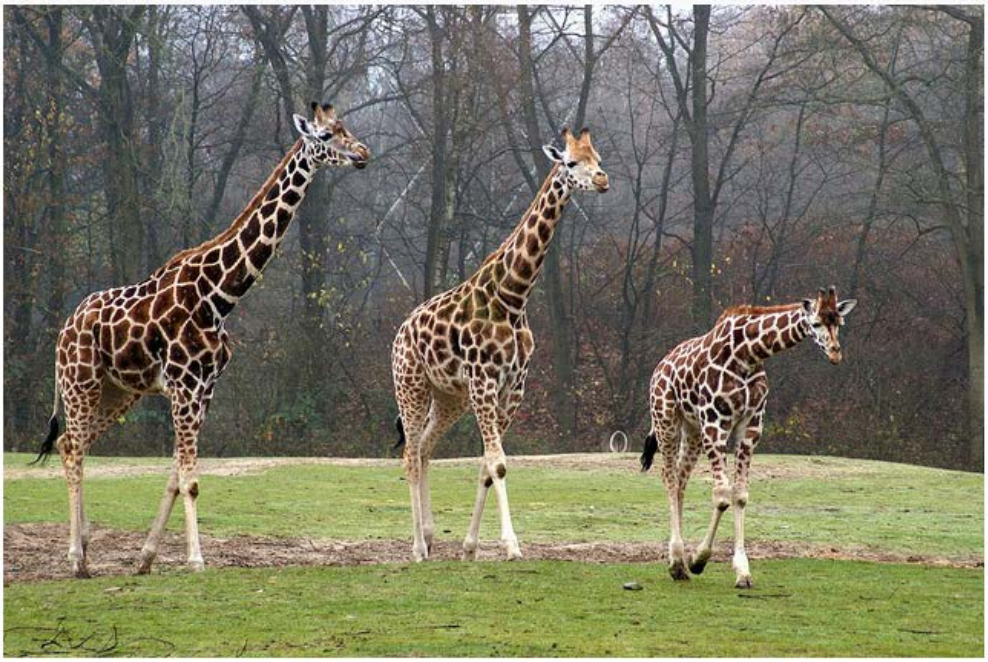
hyperparameter tuning

1. Batch size = 1024
2. Learning rate = 0.0005
3. Gradient clipping = clipnorm(1)
4. Lang_dim = 100
5. Img_dim = 256

Caption demo



```
[generated] a tennis player gets ready to hit a tennis ball
[groundtruth] a large crowd watches a tennis player while he <RARE> a ball
```



[generated] two giraffes are standing in a field
[groundtruth] three giraffes are walking in a field

Beam search

We also tried Beam search method to generate our caption. To begin with, we did not use softmax function to normalize the probability of k-best word. Not to mention, the performace was worse than original method. After adding the softmax function, and tired beam size from 3 to 5, the performance became normal but there was no improvement. We guess that the original captions we generate are good enough; that is, the problem might be the model.

Curriculum learning

First, we sorted the training data by caption length then generated samples. However,no use.