Deep Learning

Image captioning

U Kang
Seoul National University

In This Lecture

- Generate caption from given image
- Data from Flicker8k dataset

Outline

- **Introduction**
- Data
- Preprocessing Codes
- Answers

Motivation

- Automate generating description of images
- Can also be useful to those who are blind

Goals

- Generate caption which describes given image well

answer
dog is playing in the deep snow
dog running through deep snow pack
dog runs through the deep snow
white dog is running through the snow
white dog running through snow

Problem Definition

- **Given:** Image from Flicker8k dataset
- **Generate:** caption that describes given image

Outline

- ☑ Introduction
- ☐ **Data**
- ☐ Preprocessing Codes
- ☐ Answers

Training Dataset

- Images and captions from Flicker8k dataset
- 6,000 train images / 1,000 test images
- Each image has 5 captions

Providing data (1)

- Image files
- data/train_images, data/test_images
  - Each image has unique id
  - Ex) 57422853_b5f6366081.jpg

Providing data (2)

- Text files
- ./data/train_captions.txt, ./data/test_captions.txt
  - Each row has image id and its caption

```
3364160101_c5e6c52b25 game of hockey is being played
3364160101_c5e6c52b25 man wearing sports uniform runs down the field as another follows
3364160101_c5e6c52b25 player in purple white and black attempts to make play near the end of field
3364160101_c5e6c52b25 number fiveteen runs with the lacrosse ball
3364160101_c5e6c52b25 two lacrosse players are running on the sportsfield
2507312812_768b53b023 black dog and grey dog run on the beach
2507312812_768b53b023 black dog and white dog are running across sandy beach by the ocean
2507312812_768b53b023 black dog is chasing another dog along the beach
2507312812_768b53b023 two dogs are running on sand along the body of water
2507312812_768b53b023 two dogs are running on the beach along the water
1732217138_aa0199ef87 the couple eat their meal outside
1732217138_aa0199ef87 two people sit on bench
1732217138_aa0199ef87 two people sitting facing away with dreadlocks
1732217138_aa0199ef87 two people with dreadlocks
1732217138_aa0199ef87 two people with dreadlocks sit on wooden bench
```

Outline

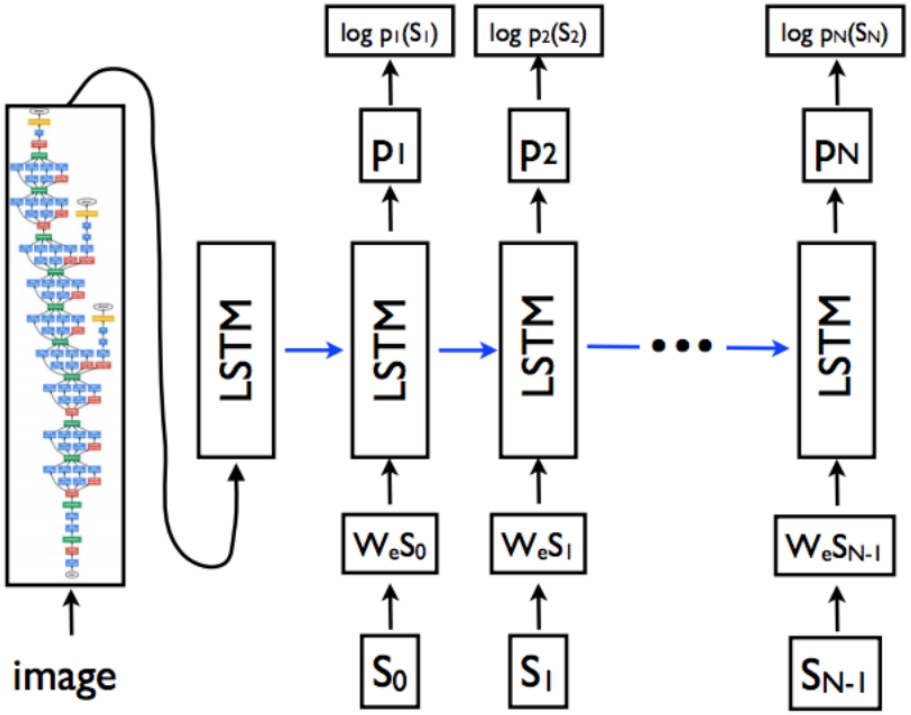
- ☑ Introduction
- ☑ Data
- ☐ **Preprocessing Codes**
- ☐ Answers

Import libraries

- We will use libraries below, so install these using ‘pip install’
  - tensorflow
  - numpy
  - pickle
  - nltk
  - Ipython
  - keras
  - matplotlib
  - tqdm

Architecture (1)

- One possible architecture is below



Architecture (2)

- Image feature will be fed into RNN as first input
- Word sequence has 2 additional token: <START>, <END> to specify start and end of generated sentence
- Each word will be embedded into word vector
- Embedded word vector will be fed into RNN network

Extract feature from Image (1)

- We will use pre-trained VGG16 to shorten our training time
  - The last layer of CNN is output layer
  - The layer before the last layer represents feature of input image
  - You can also implement your own CNN (but it will take some time)

Extract feature from Image (2)

- Import modules

```
import numpy as np
import tensorflow as tf
import pickle
import gzip
import random
import time
import os
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model
from IPython.display import Image, display
from nltk.translate.bleu_score import corpus_bleu, sentence_bleu  # nltk == 3.4.5
from tqdm import tqdm_notebook as tqdm
```

Extract feature from Image (3)

- Load pre-trained VGG16 and use the layer before last layer

```
# load VGG16
start_time = time.time()
model = VGG16()
model = Model(inputs=model.inputs, outputs=model.layers[-2].output) # drop last prediction layer
end_time = time.time()
print('Elapsed for loading VGG16:', end_time-start_time, 'sec')
```

Extract feature from Image (4)

- Extract image feature after forward pass of VGG16

```
# extracting feature using VGG16
start_time = time.time()
image_dir = './data/{}_images'.format(dtype)
for i, filename in enumerate(os.listdir(image_dir)):
    image_path = './data/{}_images/{}'.format(dtype, filename)
    image = load_img(image_path, target_size=(224, 224, 3)) # VGG16 input size
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0) # according to keras document
    image = preprocess_input(image) # according to keras document
    feature = model.predict(image)
    feature = np.squeeze(feature) # (1, 4096) -> (4096)
    image_id = filename.split('.')[0]
    features[image_id] = feature
    print('{0:>4}/{1:<4}-->{2}'.format(i+1, len(os.listdir(image_dir)), image_id), end='\r')

end_time = time.time()
print('Elapsed for extracting feature:', end_time-start_time, 'sec')
```

Extract feature from Image (5)

- Save extracted features to file for further use
  - It takes much time in extracting features

```
# saving feature as file
start_time = time.time()
with gzip.open('./data/{}_features.pickle.zip'.format(dtype), 'wb') as f:
    pickle.dump(features, f)

end_time = time.time()
print('Elapsed for saving feature:', end_time-start_time, 'sec')
```

```
Elapsed for loading VGG16: 2.8949849605560303 sec
Elapsed for extracting feature: 1125.3097143173218 sec
Elapsed for saving feature: 54.11047005653381 sec
End loading train data
Elapsed for loading VGG16: 2.825206995010376 sec
Elapsed for extracting feature: 185.02324557304382 sec
Elapsed for saving feature: 9.02383303642273 sec
End loading test data
```

Directory structure

- Until now, we have
  - Image directory
    - ./data/train_images
    - ./data/test_images
  - Caption text files
    - ./data/train_captions.txt
    - ./data/test_captions.txt
  - Extracted feature
    - ./data/train_features.pickle.zip
    - ./data/test_features.pickle.zip

Loading the Dataset

- Loaded features and captions are ‘dict’ type
  - You should try to handle these data on you own

```
train_features, train_captions = load_data('train')
test_features, test_captions = load_data('test')
```

```
print('# of images in train:', len(train_features), len(train_captions))
print('# of images in test:', len(test_features), len(test_captions))

image_id = list(train_features.keys())[0]
print('feature example')
print(train_features[image_id])
print('shape:', train_features[image_id].shape)
print('caption example')
print(train_captions[image_id])
```

```
# of images in train: 6000 6000
# of images in test: 1000 1000
feature example
[0.         0.         0.9035611 ... 0.         0.         0.        ]
shape: (4096,)
caption example
['two dogs are playing catch in field', 'two dogs attempt to get small ball', 'two dogs jump for the ball',
'two white dogs are chasing ball with one of them airborne', 'two white dogs are playing with an orange ball
in the dry grass']
```

Preprocess (1)

- We will add <START> and <END> token to each caption
  - To specify start and end of sequence
- We will make 2 dictionaries
  - word_to_idx: convert word to integer index
  - idx_to_word: convert integer index to word
- We will use only words that appear more than 20 (=threshold) times
- Length of dictionary is our vocabulary size

Preprocess (2)

- Corresponding codes are as follows

```
# return word_to_idx, idx_to_word, vocab_size
def preprocess_vocab(captions):
    threshold = 20 # threshold of occurence
    count = dict()
    word_to_idx, idx_to_word = dict(), list()

    for caption_list in captions.values():
        for c in caption_list:
            tokens = c.split(' ')
            for t in tokens:
                if t in count:
                    count[t] += 1
                else:
                    count[t] = 1

    print('# total of words:', len(count))
    print('# of words that appears >= {}:'.format(threshold), len([w for w in count if count[w] >= threshold]))

    idx_to_word = [w for w in count if count[w] >= threshold]
    idx_to_word.append('<START>')
    idx_to_word.append('<END>')
    for i, w in enumerate(idx_to_word):
        word_to_idx[w] = i

    return word_to_idx, idx_to_word, len(idx_to_word)
```

Preprocess (3)

- Convert train feature and captions to unrolled list

```
def dict_to_list(features, captions):
    image_ids, feat, capt = [], [], []
    for image_id in features:
        _feat = features[image_id]
        _capt = captions[image_id]
        for c in _capt:
            image_ids.append(image_id)
            feat.append(_feat)
            c = '<START> ' + c + ' <END>'
            capt.append(c)

    return image_ids, np.array(feat), np.array(capt)
```

```
image_ids, feats, capts = dict_to_list(train_features, train_captions)
print('len(image_ids):', len(image_ids))
print('len(feat):', len(feats))
print('len(capt):', len(capts))
```

```
len(image_ids): 30000
len(feat): 30000
len(capt): 30000
```

Test measure (1)

- **BLEU score**
  - **BLEU** (**bilingual evaluation understudy**) is a measure for evaluating the quality of text

$$precision_i = \frac{\#\ of\ \ matched\ \ i\text{-}gram\ \ between\ \ hypothesis\ \ and\ \ reference}{\#\ of\ \ possible\ \ i\text{-}gram\ \ in\ \ hypothesis\ \ sentence}$$

$$bleu = min(1, \frac{hypothesis\ \ length}{reference\ \ length})(\prod_{1}^{4} precision_i)^{\frac{1}{4}}$$

Test measure (2)

- In our practice, we can use BLEU score between true captions (as reference) and our generated caption (as hypothesis)
- Two types of BLEU score in nltk
  - sentence_bleu (for 1 example)
    - 1 hypothesis sentence, N reference sentences
  - corpus_bleu (for multiple examples such as whole train set or whole test set)
    - list of hypothesis sentences, list of (N reference sentences)

Test measure (3)

- sentence_blue can be calculated using 1 hypothesis sentence with N reference sentences
  - N also can be 1
  - If N>1, then use maximum match count
- Let's consider 1-gram
  - hypothesis: 'this is a test'
  - reference: 'this is small test'
  - bleu = 3/4 = 0.75
  - 3 means matched 1-gram: 'this', 'is', 'test'
  - 4 means possible 1-gram: 'this', 'is', 'small', 'test'

Test measure (4)

- corpus_blue can be calculated using N hypothesis sentences with M reference sentences
  - If N==1, it is same with sentence_bleu
  - If N>1, micro-average between all hypotheses
- Let's only consider 1-gram
  - hypotheses: 'this is a test', 'this is big tiger'
  - references: 'this is small test', 'this is cat'
  - bleu between hypothesis1 and references = 3/4
  - bleu between hypothesis2 and references = 2/4
  - corpus_bleu = micro-avg = (3+2)/(4+4) = 5/8

Test measure (5)

- See nltk document
  - https://www.nltk.org/_modules/nltk/translate/bleu_score.html
  - Be careful about parameter of 'sentence_bleu()', 'corpus_bleu()'
    - weights
    - references
    - hypothesis

Questions?