HW2:
Video Caption Generation

By: Ankita Pal Chatterjee
Date: 04/03/2020

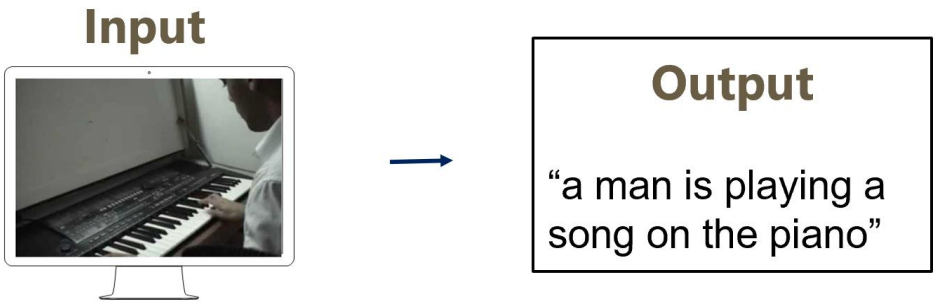
Contents

1. Introduction

In this homework, I generate caption of video frame to frame. In order to do that, I build a model in machine learning using PyTorch and Tensorflow. The model built here is a sequence to sequence model. Encoder-decoder model is the most commonly used model of Recurrent Neural Network. The source is video here. The model encodes the information into a particular representation. This representation will be mapped as a target output sequence frame to frame.

2. Goal

The goal of this homework is video caption generation. Here we should provide a short video as an input and the corresponding caption that depicts the video should be the output of the model. Below is a depiction of the same:



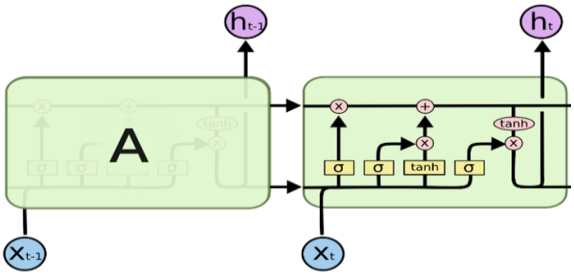
3. Challenges

We need to know why achieving the above-mentioned goal is an issue of any kind. Sequence to sequence modeling scope is broader than just the machine translation task. However, there is very little source documents available on the same topic.
There are multiple challenges we can face during implementation:

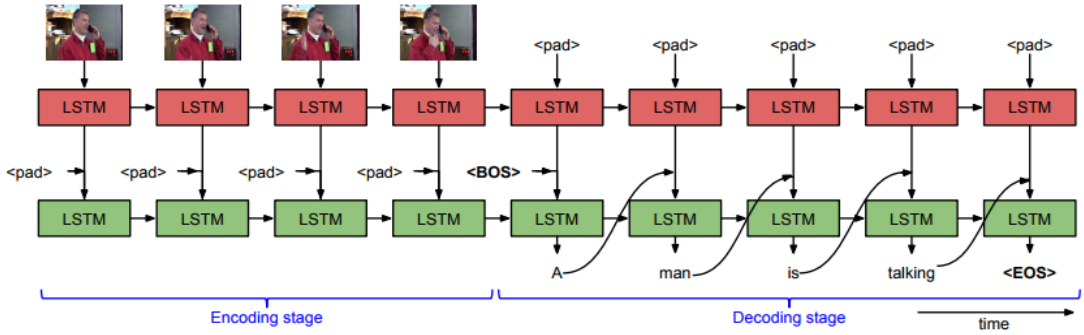
a. Different attributes of video (object, action)
b. Variable length of I/O

4. Method Understanding

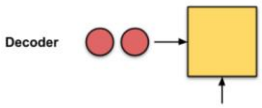
Sequence to sequence is a type of model using RNN (Recurrent Neural Network). It can be used as a model for machine interaction and machine translation. Autoencoders are a type of self-supervised learning model. LSTM (Long Short-Term Memory) unit cell output than project to a vocabulary-size vector.

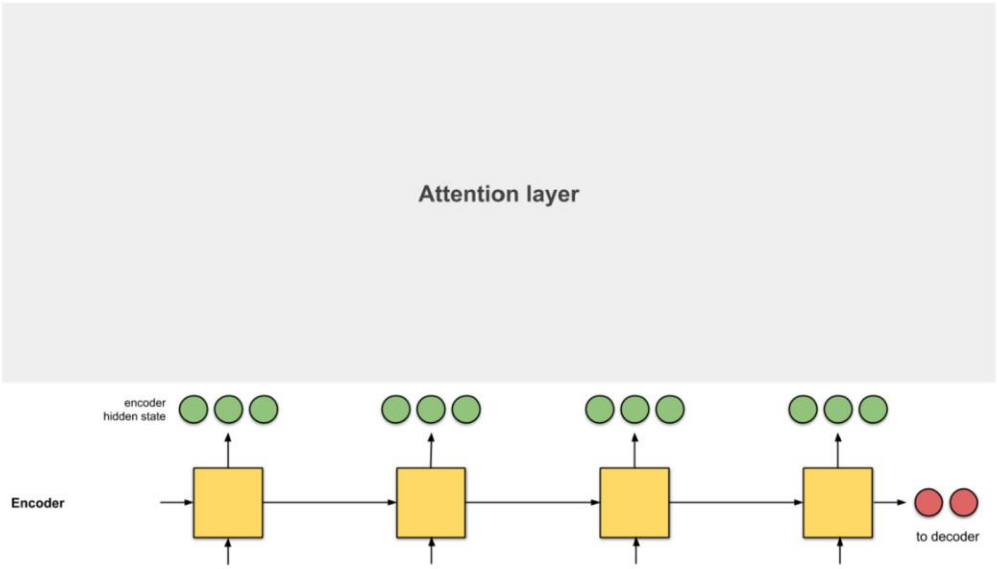


The two layers of LSTM model consists of Encoder and Decoder. Below is an image explanation of the model:



Attention is an interface between the encoder and decoder that provides the decoder with information from every encoder hidden state.





5. All Version Requirement:

python 3.6.0
tensorflow-gpu==1.15.0
cuda==9.0
numpy== 1.14
pandas==0.20

6. Execution Steps

i) In this step preprocess data by adding packed padded sequences and masking. Packed padded sequences allow us to only process the non-padded elements of our input sentence with our RNN. Masking is used to force the model to ignore certain elements we do not want it to look at, such as attention over padded elements. This step is executed for performance boosting.
The training and testing data were downloaded from the power point presentation provided and kept in local under 'MLDS_hw2_1_data' folder. vocab size should minimum be 3.
Tokenize:

a. <pad> : Pad the sentence to the same length
b. <bos> : Begin of sentence, a sign to generate the output sentence.
c. <eos> : End of sentence, a sign of the end of the output sentence.
d. <unk> : Use this token when the word isn't in the dictionary or just ignore the unknown word.

Object files 'vid_id.obj',' dict_caption.obj',' dict_feat.obj' are being written to build dictionary.
Below is the command that I executed in Ipython console. The corresponding screenshot is as follows:

!python sequence.py ./MLDS_hw2_1_data/training_data/feat/ ./MLDS_hw2_1_data/training_label.json

```
In [125]: !python sequence.py ./MLDS_hw2_1_data/training_data/
feat/ ./MLDS_hw2_1_data/training_label.json
From 6098 words filtered 2881 words to dictionary with minimum
count [3]

shape of caption: (24232, 2)
Maximum length of the captions: 40
Average length of the captions: 7.711084516342027
Number of unique tokens: 6443
ID of first video: -8y1Q0rA3n8_108_115.avi
Shape of features of first video: (80, 4096)
Caption of first video: A man slices through a two liter plastic
bottle of soda pop with a sword
ID of fifth video: -bjOB4zS0uE_100_105.avi
Shape of features of fifth video: (80, 4096)
Caption of fifth video: A man is broken a bottle by a sword
```

ii) Next task is to perform training on the model. This step included two .py files. One is to build the sequence to sequence model. Another is to train the machine on the model built. While training below parameters have been taken:

Learning rate: 0.001
batch_size: 50
use_attention: True
max_encoder_steps: 64
max_decoder_steps: 15,
embedding_size: 1024
beam_size: 5 (with beam search True)

!python train.py ./MLDS_hw2_1_data/testing_data/feat/ ./MLDS_hw2_1_data/testing_label.json ./output_testset.txt

```
Saving model with BLEU score: 0.5761 ...
Highest [10] BLEU scores:  ['0.5869', '0.5761', '0.5761', '0.5761', '0.5761', '0.4718']
Epoch# 5, Loss: 2.0134, Average BLEU score: 0.5761, Time taken: 283.66s
0050/1450
0100/1450
0150/1450
0200/1450
0250/1450
0300/1450
0350/1450
0400/1450
0450/1450
0500/1450
0550/1450
0600/1450
0650/1450
0700/1450
0750/1450
0800/1450
0850/1450
0900/1450
0950/1450
1000/1450
1050/1450
1100/1450
1150/1450
1200/1450
1250/1450
1300/1450
1350/1450
1400/1450
0050/1450
0100/1450
Average bleu score is 0.5190163880675748
Saving model with BLEU score: 0.5190 ...
Highest [10] BLEU scores:  ['0.5869', '0.5761', '0.5761', '0.5761', '0.5761', '0.5190',
'0.4718']
Epoch# 6, Loss: 1.9802, Average BLEU score: 0.5190, Time taken: 211.91s
0050/1450
0100/1450
0150/1450
0200/1450
```

Bleu Score has been calculated after each epoch

```
Highest [10] BLEU scores:  ['0.5761', '0.5761', '0.5761', '0.5761', '0.5761', '0.5761',
'0.5761', '0.5761', '0.5761', '0.5761']
```

```
ighest [10] BLEU scores:  ['0.5761', '0.5761', '0.5761', '0.5761', '0.5504', '0.5504',
0.5190', '0.5190']
poch# 7, Loss: 1.9103, Average BLEU score: 0.5761, Time taken: 261.43s
```

```
verage bleu score is 0.5760704024416632
Highest [10] BLEU scores:  ['0.5941', '0.5761', '0.5761', '0.5761', '0.5761', '0.5761'
'0.5761', '0.5761', '0.5761', '0.5761']
poch# 57, Loss: 1.3722, Average BLEU score: 0.5761, Time taken: 227.66s
0050/1450
```

7. Results

After finishing all 200 epochs in training file, I executed the bleu_eval.py with the below command:

!python bleu_eval.py testset_output.txt

Bleu_score	**0.6110**

8. References:

1) http://www.cs.utexas.edu/users/ml/papers/venugopalan.iccv15.pdf
2) https://gist.github.com/vsubhashini/38d087e140854fee4b14