

CPSC 8430: Deep Learning

Homework 2

Video Caption Generation

- Yaswanth Poreddy

Abstract:

Both the discipline of natural language processing and computer vision have challenged the generation of video captions using natural language automatically. Real-time videos will have complex dynamics. Therefore the technologies we use should be sensitive to temporal structure and allow for variable-length input and output. The Encoder-Decoder framework will be used, and the MSVD dataset will be used for model assessment. We use two Long Short Term Memory (LSTM) units for encoding and the other for decoding. To learn video representation, an encoder with deep neural networks is used. On the other side, the decoder will generate a sentence for the learned representation. We use the Beam search algorithm to generate efficient captions.

Introduction:

Automatically generating video captions using natural language has been a challenge for both natural language processing and computer vision. Recurrent Neural Networks (RNNs), which mimic sequence dynamics, were shown to be effective in visual interpretation. A video description must manage a variable-length input series of frames and a variable-length output sequence of words, while an image description will try to handle a variable-length output sequence of words.

The model will be restricted to the particular format using Subject, Object, and Verb, as compared to the template-based approach used in most earlier studies. The result is determined by the probability map of properly matching the words. Since it restricts itself to a specific sequence, the text generated in this case may be irrelevant to the image. It can't be encouraged in real-time. Our model should be able to learn new images and be dynamic. So let's create a sequence-to-sequence model that takes frames as input and provides variable-length text output. Long Short Term Memory (LSTM), a type of RNN, would be used to improve the capability of a sequence-to-sequence model.

In seq-to-seq tasks like speech recognition and machine translation, LSTMs have been shown to be efficient. A stacked LSTM performs the encoding, which

encodes the frames one by one. The LSTM is given the output of a Convolutional Neural Network applied to each input frame's intensity values. Once all of the frames have been read, the model generates a sentence word by word. By pooling the representation of all frames, LSTMs are used in one of the approaches to generate video subtitles. They use mean-pools on CNN output features to get a single feature vector. The main disadvantage of this method is that it ignores the organization of the video frames and fails to change any temporal information.

Attention mechanisms have been incorporated into many neural networks because they let salient features come to the forefront dynamically as needed. Machine translation, image captioning, and video captioning are just a few of the activities that have been observed to benefit from attention models. At the decoder stage of our model, we include an attention layer that can retrieve valuable information from the previous word input.

Dataset and Features:

The MSVD (Microsoft Video Description) dataset was used in this experiment. About 120K sentences make up the Microsoft Research Video Description Corpus (MSVD) dataset. Mechanical Turk workers were paid to optically canvas a short video snippet and then describe the action in a single line. There are 1550 video clips in the dataset, each lasting 10 to 25 seconds. For training and testing, we've split the videos into 1450 and 100 videos, respectively. After preprocessing with pre-trained CNN VGG19, we stored the features of each video in the format of 80x4096. To get the most out of the training, we don't reduce any frames from the video.

Approach:

Our Model is a seq-to-seq model which takes sequence of video frames as input and generate sequence of words as output.

Version Requirement:

We are using the following technologies for functioning of the Model.

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

Execution:

- In the preprocessing stage, padded sequences and maskings are applied to the data. With packed padded sequences, we can use our RNN to solely process the non-padded elements of our input sentence. Masking is a technique used to make the model ignore certain elements we don't want it to see, such as padding elements. This process is carried out to improve performance. The training and testing data were obtained from the power point presentation and saved to the 'MLDS hw2 1 data' folder on the local machine. Vocab size should minimum be 3.
- Tokenization:
 1. <pad>: Pad the sentence to the same length
 2. <bos>: Begin of sentence, a sign to generate the output sentence.
 3. <eos>: End of sentence, a sign of the end of the output sentence.
 4. <unk>: Use this token when the word isn't in the dictionary or just ignore the unknown word.

The dictionary of id and caption of respective videos are stored in two object files. The dictionary is being built using the object files 'vid id.obj', 'dict caption.obj', and 'dict feat.obj'.

sequence.py is used for execution, at my active node in the Palmetto cluster, I used the following command:

```
python sequence.py
/home/yporedd/Yash/MLDS_hw2_1_data/training_data/feat/
/home/yporedd/Yash/MLDS_hw2_1_data/training_label.json
```

I got the following results:

```
(tf_gpu) [yporedd@login001 VC]$ python sequence.py /home/yporedd/Yash/MLDS_Data/MLDS_hw2_1_data/training_data/feat/ /home/yporedd/Yash/MLDS_Data/MLDS_hw2_1_data/training_label.json
From 6098 words filtered 2881 words to dictionary with minimum count [3]

Caption dimension: (24232, 2)
Caption's max length: 40
Average length of captions: 7.711084516342027
Unique tokens: 6443
ID of 17th video: kWLNZzuo3do_25_32.avi
Shape of features of 17th video: (80, 4096)
Caption of 17th video: The target got more bullet holes in it
(tf_gpu) [yporedd@login001 VC]$
```

Our next task is to train the model we created. There are two python files in this folder. We'll use sequence.py to build the sequence-to-sequence model. With the help of train.py, we will train the model. I have used the Hyperparameters mentioned in the Home work presentation.

I have run the following command in palmetto cluster on my active node using ssh.

```
python train.py /home/yporedd/Yash/MLDS_hw2_1_data/training_data/feat/
/home/yporedd/Yash/MLDS_hw2_1_data/training_label.json
./output_testset_manish.txt
```

There are 3 Args in the above command. The first Arg is the path of the feature

```
0100/1450
The Average BLEU Score of model is: 0.6959820600809971
Saving model with BLEU Score : 0.6960 ...
Highest [10] BLEU scores: ['0.6960', '0.4309']
Epoch# 1, Loss: 2.2536, Average BLEU score: 0.6960, Time taken: 31.19s
Training done for batch:0050/1450
Training done for batch:0100/1450
Training done for batch:0150/1450
Training done for batch:0200/1450
Training done for batch:0250/1450
Training done for batch:0300/1450
Training done for batch:0350/1450
Training done for batch:0400/1450
Training done for batch:0450/1450
Training done for batch:0500/1450
Training done for batch:0550/1450
Training done for batch:0600/1450
Training done for batch:0650/1450
$Killed
(tf_gpu) [yporedd@login001 VC]$
```

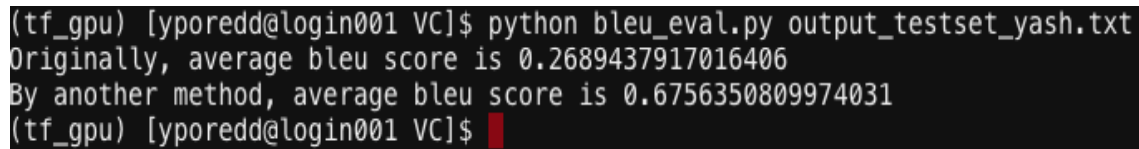
map which are in the format of .npy file and second Arg is the path of the testing_label.json file which has captions for a particular video id.

After each epoch, I calculated the Bleu score. The array of blue scores are shown in descending order of the scores.

Results:

I have calculated the Bleu scores using the following command.

```
python bleu_eval.py output_testset_yash.txt
```

A terminal window screenshot with a black background and white text. The prompt is '(tf_gpu) [yporedd@login001 VC]\$'. The command 'python bleu_eval.py output_testset_yash.txt' has been executed. The output shows two lines: 'Originally, average bleu score is 0.2689437917016406' and 'By another method, average bleu score is 0.6756350809974031'. The prompt is repeated at the end of the second line, followed by a red cursor block.

```
(tf_gpu) [yporedd@login001 VC]$ python bleu_eval.py output_testset_yash.txt
Originally, average bleu score is 0.2689437917016406
By another method, average bleu score is 0.6756350809974031
(tf_gpu) [yporedd@login001 VC]$
```

From the above screen shot we can see that the average Bleu score has reached around 0.6756 after few epochs(5).

After training the model for 200 epochs, the score reached almost 0.662.

References:

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

