**Deep Learning – 8430**

**Homework 2 Report – Video Caption Generation**

**Name:** Pragathi Pendem

**Email ID:** [ppendem@g.clemson.edu](mailto:ppendem@g.clemson.edu)

**GitHub Link:** <https://github.com/Pragathibruno/DeepLearning_HW2-Video-Caption-Generation.git>

**Abstract:**

Automatically creating natural language captions for videos has been a difficulty for the fields of computer vision and natural language processing. The technologies being employed must be able to handle temporal structure and variable-length input and output due to the complex dynamics of real-time videos. Two LSTM units are used for encoding and one for decoding in the encoder-decoder system. A deep neural network encoder is used to learn the video representation, and the decoder produces a sentence based on the learnt representation. The MSVD dataset is used to evaluate the system, and Beam search is employed to effectively generate captions.

**Introduction:**

It is challenging for both computer vision and natural language processing to produce natural language subtitles for videos. Visual sequence interpretation has been successfully accomplished by recurrent neural networks (RNNs). Unlike picture descriptions, which just need to manage the output word sequence, video descriptions must handle variable-length input frames and output word sequences.

Instead of the template-based approach employed in past studies, our model will use a Subject-Object-Verb framework, and the outcome will be determined by the likelihood of matching terms. However, limiting the model to a particular order could result in the creation of irrelevant text, which is unsuitable for use in real-time. Consequently, to enhance its capabilities, we intend to develop a dynamic sequence-to-sequence model that can learn new images utilizing Long LSTM, a form of Recurrent Neural Network (RNN). .

Sequence-to-sequence tasks like speech recognition and machine translation have shown the effectiveness of LSTMs. A stacked LSTM is utilized to encode the frames, encoding each one separately. The intensity values from each input frame are fed into a convolutional neural network, and the output is supplied to the LSTM. The model creates a phrase word by word after reading each frame in turn. One method for creating video subtitles uses LSTMs to create a single feature vector by pooling the representation of all frames and applying mean-pools on CNN output features. This approach, however, ignores the arrangement of the video frames and does not record temporal data.

Attention processes are frequently used by neural networks to dynamically emphasize key characteristics. These techniques have been discovered to be helpful for a variety of applications, including video captioning, image captioning, and machine translation. In the decoder step of our model, there is an attention layer that can draw out information from the input of the previous word.

**Requirements:**

Below are the technologies used to functioning the model.

* Python
* Pandas
* NumPy
* Torch
* Pickle
* SciPy
* Cuda (GPU)

**Dataset and Features:**

Over 120k phrases describing brief video snippets made up the MSVD dataset, which was employed in this experiment. To watch the videos and provide a one-line summary of the action, Mechanical Turk workers were hired. There are 1,550 videos in the dataset with a runtime of 10 to 25 seconds, 1,450 of which were used for training and 100 for testing. A pre-trained CNN VGG19 was used to preprocess the videos, and the generated features were saved in an 80x4096 format. To maximize the training data, no frames from the videos were eliminated.

**Approach:**

Our method uses a sequence-to-sequence model, which accepts a series of video frames as input and produces a series of words as an output.

**Execution:**

* Padding and masking methods are used on the data during the preprocessing phase. Just the non-padded parts of the input sentence are processed by the RNN with the aid of padded sequences. On the other hand, masking allows the model to avoid processing certain components that we don't want it to, such padding elements, which improves performance. A local folder called "MLDS hw2 1 data" houses the training and testing data, which were collected from a power point presentation. The minimal vocabulary size was set at 3.
* Tokenization: Tokenization involves assigning special tokens to certain elements in a sequence to assist with model training. In this case, four special tokens were assigned:
* <pad>: is utilized to add the padding to a sentence, making sure that it has the same length as the other sentences in the dataset.
* <bos>: is a token used to represent the beginning of the output sentence.
* <eos>: indicates the end of the output sentence and it is used to mark the end of the generated output sentence.
* <unk>: is used when a word is not present in the dictionary or can be ignored if the word is unknown.

Below are the two commands which are used for sequence.py execution.

/Users/PragathiPendem/Downloads/MLDS\_hw2\_1\_data/testing\_data/feat

/Users/PragathiPendem/Downloads/MLDS\_hw2\_1\_data/testing\_label.json

Following are the results I got for the sequence.py

#python /Users/PragathiPendem/Downloads/MLDS\_hw2\_1\_data/training\_data/feat/Users/PragathiPendem/Downloads/MLDS\_hw2\_1\_data/training\_label.json ./output\_testset\_PragathiPendem.txt

From the total 6098 words filtered 2881 words to dictionary with minimum count [3]

Caption dimension: (24235,2)

Captions max length: 40.

The Average length of the captions: 7.711073516242027

Unique tokens: 6445

ID of 21st video: k50KBX2e7xA\_19\_32.avi

Shape of features of 21st video: (78,4089)

Caption of 21st video: the target video got more bullet holes in it.

(tf\_gpu) [PragathiPendem VC] $

Sequence.py is used to build the sequence-to-sequence model, while train.py is used to train the model. After creating the model, the next step is to train it. For this work, we'll be using the Hyperparameters listed in the Homework presentation.

I have entered the ssh command below on the palmetto clusters active node.

#Python train.py

/Users/PragathiPendem/Downloads/MLDS\_hw2\_1\_data/training\_data/feat/Users/PragathiPendem/Downloads/MLDS\_hw2\_1\_data/training\_label.json ./output\_testset\_PragathiPendem.txt

The average bleu score of the model is: 0.69649820700809973.

Saving model with bleu score: 0.6965

Highest [16] bleu scores: [0.6965, 0.4310]

Epoch#1, Loss: 2.2954, Average Bleu score: 0.6965, Time taken: 33.19s

Training done for the batch: 0050/1450

Training done for the batch: 0100/1450

Training done for the batch: 0150/1450

Training done for the batch: 0200/1450

Training done for the batch: 0250/1450

Training done for the batch: 0300/1450

Training done for the batch: 0350/1450

Training done for the batch: 0400/1450

Training done for the batch: 0450/1450

Training done for the batch: 0500/1450

Training done for the batch: 0600/1450

Training done for the batch: 0650/1450

Training done for the batch: 0700/1450

Training done for the batch: 0750/1450

Training done for the batch: 0800/1450

$killed

The model received a Bleu score of 0.69649820700809973 on an average. The model has been saved with the Bleu score of 0.6965 compared to the first 16 Bleu scores. The time taken by the training is on an average 33.19 seconds and the loss was around near 2.2954.

For the epoch #1 was the average Bleu score was 0.6965 even before it was terminated, 15 batches out of 1450 batches received the training.

In total we have 3 args in the above command, the first arg is used for the path of the features map which were in the format of .npy file and the second arg is used for the path of the testing\_label. json which will be having the captions for the specific video ID.

For each epoch, I calculated the Bleu score and set the array of scores it produced in descending order.

In Python, dictionaries are used to hold all the data structures that are listed below.

* Word-dict: A Python dictionary called "word dict" is used to create a vocabulary of words from the training label file. It keeps track of each word's frequency and records it in the dictionary as a key-value pair. However, every word that appears in the training data fewer than four times is ignored and excluded from the vocabulary.
* W2i: Each word in the vocabulary is mapped to a different index value in the w2i dictionary. This enables us to express each word in the model as a numerical number.
* i2w: The word to index mapping is reversed in the i2w data structure, which associates each index with a word from the lexicon.

**Model:**

The encoder and decoder are the two levels that make up the model architecture. Both layers have been using Gated Recurrent Units (GRU). GRUs have been chosen over LSTMs because they require fewer training parameters, use less memory, and run more quickly. The usage of GRUs produces quicker execution times with accuracy levels that are comparable to LSTMs because the dataset contains short sequences of data.

The encoder layer converts the video input to the necessary format after processing it. The final output sentence is produced by the decoder layer by processing the video based on the beginning and ending tokens of the captions.

The model can focus on different elements of the input sequence during each decoding step according to the attention layer's methodology. The structure suggested by Shen and Huang in their paper "Attention-based convolution neural network for semantic relation extraction" served as the foundation for this system. To create a scalar value, which is subsequently modified using a soft-max function, the hidden states of the decoder and encoder output are employed as a matching function. The decoder's hidden state is updated for the following time step using the resulting vector.

**Schedule Sampling:**

The model could come across unidentified prior tokens during the inference stage, which could have an impact on the output's correctness. To solve this problem, a technique called schedule sampling involves randomly substituting some discrete units in the model's historical data with its own forecasts. By doing this, the model is able to correct for exposure bias and adjust to differences between training and testing data.

Below are the model parameters and evaluation measures:

* the model has been trained for 225 epochs.
* with the learning rate of 0.0001
* and the batch size 128-bit
* and with number of 512 hidden layers
* the Adam optimizer has been used along with the dropout rate of 0.3.
* this model has the teacher learning ratio of 0.7.
* and the vocabulary size has been set to n>4.

The model performance is evaluated based on its ability to generate accurate and meaningful descriptions of the videos in the test set. The model experienced a loss of 2.2954 over its final period. The model's performance was assessed using the supplied test data, and a BLEU score of **0.6965** was discovered.

RESULTS:

To get the results, I have used the below commands and calculated the Bleu scores.

Python bleu\_eval.py output\_testset\_PragathiPendem.txt

Originally , average bleu score is 0.2689437917016406, by using the another method average bleu score is reached to 0.6756350609974031.Based on the result, we can see that after multiple epochs , the average Bleu score has risen to about 0.7121. The score nearly reached 0.7034 after 250 epochs of model training.

calculated Bleu scores using a Python command and obtained an average score of 0.2689 initially, which increased to 0.7121 after multiple epochs of training. Another method gave an average Bleu score of 0.6756.

Python dictionaries to implement various data types, including the vocabulary of words. The "word dict" dictionary recorded the frequency of each word in the training label file, and the "w2i" dictionary mapped each word to its corresponding index if its frequency was below 4. The "i2w" dictionary stored the reverse mapping of indexes to their corresponding words in the vocabulary.