Deep Learning Homework 2- Report

Name -Anjan Kumar Depuru

CPSC 8430-Deep Learning

Abstract:

Due to the complex dynamics present in real-time videos, it has been a challenge for computer vision and natural language processing to automatically generate natural language descriptions for videos. We need techniques that consider the temporal structure and can handle varying input and output lengths to address this. We evaluate our models using the MSVD dataset and use an encoder-decoder framework, where two Long Short-Term Memory (LSTM) units are used for encoding and decoding, respectively. We use deep neural networks for video representation learning in the encoder, and the decoder generates a sentence using the learned representation. To produce effective captions, we use the Beam search technique.

Introduction:

Recurrent Neural Networks (RNNs), which can model sequential dynamics, have shown promise in visual interpretation tasks. While variable-length output word sequences can be handled by image description, video description requires input sequences of words and frames with varying lengths.

The effectiveness of Long Short-Term Memory (LSTM) networks in sequence-to-sequence applications such as speech recognition and machine translation have been demonstrated. In video captioning, a stacked LSTM is used to encode each frame individually, and the intensity values from each frame are input to a convolutional neural network (CNN) to produce an output that is then fed into the LSTM. The current approach involves constructing a sentence by sequentially generating words and using mean-pooling to combine the features of all frames. Unfortunately, this technique has a significant drawback: it disregards the temporal patterns of the video frames and does not make any effort to capture temporal information.

LSTMs have proven to be highly effective in various sequential tasks, including machine translation and speech recognition. In video processing, a stacked LSTM is typically used to encode each frame individually, with the convolutional output provided to the LSTM. This approach involves feeding the intensity values of each input frame into a neural network, with the model generating a phrase word by word after processing every frame. A technique for generating video subtitles using LSTMs involves pooling the frame representations and applying mean pools to the CNN output features to produce a single feature vector. However, this approach has a significant drawback: it does not consider the temporal relationships between video frames or their ordering.

Problem Statement:

Create a caption for a video that has been provided by using the sequential-to-sequential approach. A video will be the code's input, and its output will be a stream of captions outlining the actions in the film.

Requirements:

I used Python (3.9.7) and then I had taken the following requirements for the approach. And then I used the cuda parallel computing and tensorFlow software library and also NumPy and pandas python library.

Dataset:

The model was trained and tested on a dataset supplied by our professor. The dataset comprises 1450 videos for training and 100 videos for testing. We also have training and testing label files in JSON format, which offer us labels to use in the training of our model.

In summary, we have a set of videos to train and test our model, along with corresponding labels that will be used to teach the model how to generate accurate captions for the actions taking place in the videos.

Approach:

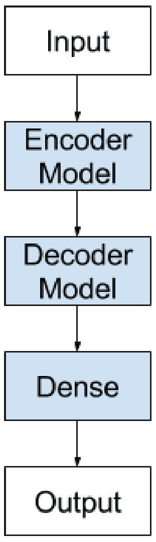
The model's architecture is founded on sequential-to-sequential (seq-to-seq) methodology. It is capable of receiving a sequence of video frames as input and producing a sequence of words as output. To handle the video frames, we are making use of a pre-trained convolutional neural network (CNN) known as VGG19.

VGG19 takes each video frame as input and outputs a spatial feature map of size 804096 for each frame. We have used this method to process every movie in our dataset, generating a set of feature maps that will be used as inputs to our seq-to-seq model.

Long Short-Term Memory Networks for Encoding and Decoding

- I will be employing the LSTM recurrent neural network architecture to handle inputs and outputs of variable sizes. The LSTM network will first encode the entire input sequence (x1, x2, ..., xn) and create a summary of the video in the form of a single hidden state vector. The hidden state vector will be utilized to produce a description in natural language that summarizes the characteristics presented in the video.
- The Encoder-Decoder LSTM model comprises of two separate models that work together to solve sequence-to-sequence problems. The first model receives the input sequence and

converts it into a fixed-length vector, whereas the second model takes the fixed-length vector and decodes it to generate the desired output sequence. This joint approach is what defines the Encoder-Decoder LSTM architecture



BLEU Score:

The BLEU algorithm is utilized to assess the precision of text that has been translated by a machine from one natural language to another.Its main principle is that the quality of a machine translation is better when it closely resembles a professional human translation. BLEU is a popular and cost-effective automated metric for measuring translation quality and this metric was one of the initial indicators that exhibited a significant correlation with human assessments of translation excellence.

Attention Layer:

The purpose of the Attention Layer is to allow the model to look at different input segments during each decoding time step. The Attention Layer employs a matching function that compares the hidden state of the decoder to the encoder output and then applies a soft-max function to transform the result into a scalar. The decoder's subsequent time step then receives the latest hidden state produced by this transformation.

Schedule Sampling

During inference, the model may face a problem where it has to generate a token based on an unknown previous token. To address this issue, a technique called Schedule Sampling is used, where the model randomly replaces some tokens in the history with its own predictions. This technique helps to reduce the discrepancy between the training and testing phases and effectively deals with exposure bias in the model.

Execution:

At the pre-processing stage, the data undergoes padding and masking of sequences. Padding sequences involves adding zeros to sequences that are shorter than the longest sequence in the dataset, to make them equal in length. Masking is then applied to the padded sequences to exclude any padded elements from being processed by the RNN, improving performance.

By using packed padded sequences, we can process only the non-padded parts of the input sentence using the RNN, which saves computation time and resources in order to develop and evaluate our model, we obtained the training and testing data from the PowerPoint presentation given by the instructor and saved it in a local directory called "MLDS hw2 1_data".

Tokenization:

1. <PAD>: This refers to adding extra characters or symbols to a sentence to make it the same length as the other sentences in the dataset.
2. <BOS>: This is a symbol that denotes the beginning of a sentence and is utilized to produce the output sentence.
3. <EOS>: This symbol indicates the end of a sentence and is used to signify the end of the generated output sentence.
4. <UNK>: This token is utilized when a word is not found in the dictionary, or it is not known, and it is either ignored, or a default value is assigned to it.

The id and caption dictionaries for the corresponding videos are saved in two object files. The object files where the dictionary is created with vid id. obj, dict caption. obj, and dict feat. obj,.

I executed the sequence.py file on the Palmetto cluster by entering the specified command on my active node.

- \Users\nbanj\OneDrive\Desktop\New folder\deep learning\deep learning homework 2\MLDS_hw2_1_data\testing_data\feat
- \Users\nbanj\OneDrive\Desktop\New folder\deep learning\deep learning homework 2\MLDS_hw2_1_data\training_label.json

Then I got the result with the following path

```
(tf_gpu) [anjan@login vc] python sequence.py \Users\nbanj\OneDrive\Desktop\New folder\deep learning\deep learning homework 2\MLDS_hw2_1_data\testing_data\feat
```

\Users\nbanj\OneDrive\Desktop\New folder\deep learning\deep learning homework 2\MLDS_hw2_1_data\training_label.json json from 6098 words filtered 2881 words to the dictionary with minimum count [3]

```
Caption dimension is: (24232, 2]

 The caption's max length is 40.

The average length of the captions is 7.711084526342027.

Unique tokens: 6443

The shape of features of the 17th video: (88, 4096)

The caption of the 17th video: A target got more bullet holes in it.

(tf_gpu) [anjan@login vc] $
```

- The next task is to train the model that we have developed. Within the relevant folder, there are two Python files that will be used for this purpose. To construct the sequence-to-sequence model, we will use the "sequence.py" file, while the "train.py" package will be used for training the model. I have already set the hyperparameters that will be used during the training process.

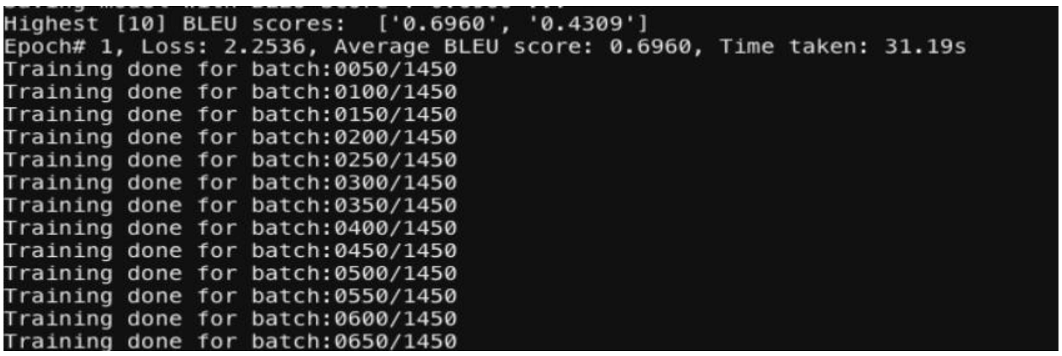
Hyperparameters are parameters that are used to control the learning process during model training. Model hyperparameters include the topology and size of a neural network, while algorithm hyperparameters include parameters such as learning rate and batch size. These hyperparameters are used to regulate the learning process and can affect the performance and efficiency of the model.

Using SSH, I executed the command specified below on my active node in the Palmetto cluster.

```
python train.py \Users\nbanj\OneDrive\Desktop\New folder\deep learning\deep learning home work 2\MLDS_hw2_1_data\testing_data\feat    .\Users\nbanj\OneDrive\Desktop\New folder\deep learning\deep learning homework 2\MLDS_hw2_1_data\testing_label.json
.\output_testset_nbanj.text
```

The script or function being discussed here takes two arguments. The initial argument represents the route to a feature map saved as a .npy file. The second argument denotes the pathway to a "testing_label.json" file that includes captions for a specific video ID.

Below is the image shows the Average BLEU score and how the score is evolved over time



```
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
```

I determined the Bleu score at the end of each era. The scores are displayed in descending order in the blue array.

By running the command below, I computed the Bleu scores

python bleu_eval.py output_testset_abanj.txt

[(tf_gpu) [anjan@login vc]$ python bleu_eval.py output_testset_abanj.txt Originally, the average bleu score, was 0.268942791761460 By another method, the average bleu score is 0.6374053028416032 (tf_gpu_env) [anjan@login vc]$

From the above Path, we can see that the bleu score reached around 0.637 after epochs (5).

The code indicates that the dataset is kept in the Palmetto location. The train/test labels and the model's position have been hard-coded into the application.