



CPSC-8430-001

Deep Learning: HW-2

Dineshchandar Ravichandran
C19657741
Email: dravich@g.clemson.edu

Contents

| | |
|---|---|
| Introduction | 3 |
| Git Hub Location: | 3 |
| 1. Solution | 3 |
| 1.1. Data Pre-processor | 3 |
| 1.2. Models: | 5 |
| 1.2.1 Encoder and Decoder | 5 |
| 1.2.2 Attention Layer: | 5 |
| 1.3. Training and Testing: | 6 |

Introduction

- In this assignment, we develop a sequential-to-sequential model with an attention mechanism to generate captions for input videos.

Git Hub Location:

https://github.com/DineshchandarR/CPSC-8430-Deep-Learning-001/tree/main/HW2_1

1. Solution

- To generate the captions for the video input, the solution has the following major parts:
 - Data pre-processor
 - Base Model (S2VT)
 - Encoder and Decoder layer
 - Attention Layer
 - Training and Testing
 - BLEU score
- The models as mentioned above are structured in the following scripts:
 - Main
 - Data pre-processor
 - Base Model (S2VT)
 - Training
 - Models
 - Encoder and Decoder layer
 - Attention Layer
 - Testing
 - Testing
 - BLEU score
 - Hw2_seq2seq.sh:
 - Shellcode to execute testing with the test data directory and test out file location.

1.1. Data Pre-processor

- This file reads all the training video features and the corresponding training video captions and stores it in a feature dictionary and a caption dictionary.
 - **Dictionary:**
 - This function reads the caption JSON file and stores the vital words. Dictionary contains all the captions in the English language for the videos. Where in many instances, a single video may have multiple captions for it. The dictionary stores all the crucial words for captions against the video. This is achieved by analyzing words in the captions. If the word count exceeds the threshold (more than three times of repetition), the word will be stored in the dictionary. The words which appear less than the threshold will be stored as unknown <UNK> in the dictionary to keep the dictionary size low and only store the vital information.

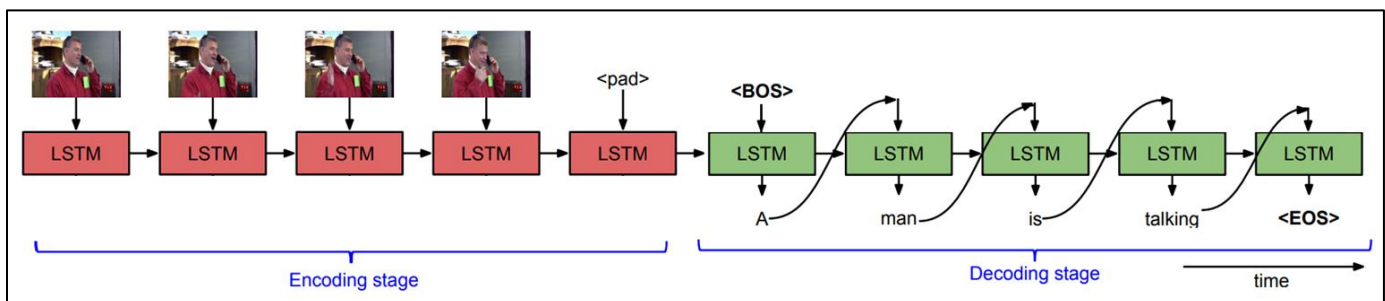
- This operation is done as we cannot directly feed the input data into the model. Hence these sentences are converted into words and their respective numerical index so the model can work on the input data.

- Tokens used to store the data in the dictionary:
 - <PAD>: Pad the sentences to the same length to maintain uniformity.
 - <BOS>: Beginning of the sentence, an identifier to generate the output sentence.
 - <EOS>: End of sentence, an identifier to signal the system end of output sentence.
 - <UNK>: This token is used when the word is not stored in the dictionary as mentioned above based on their appearance so that these words will be treated as unknown.
- **Data processor:** This function will load the video labels and their respective features based on the video ids. This will feed the captions in a dictionary and generate tensor output for the video features and their captions.

1.2. Models:

1.2.1 Encoder and Decoder

- This script contains implementing the sequence-to-sequence encoder-decoder model using 2-layer 2 layers of GRU (RNNs) implemented; the video is processed and encoded (encoderRNN) in the first layer of the RNN, and an output is generated with the help of the decoderRNN. Tokens are employed in the decoding procedure to segment the captions depending on the beginning and closing verses. Then processing is applied to the video to generate the English words.
- Dropout percentage = 0.3, hyperparameters used for the encoders and decoders.
- The GRU network has a similar structure as illustrated below:

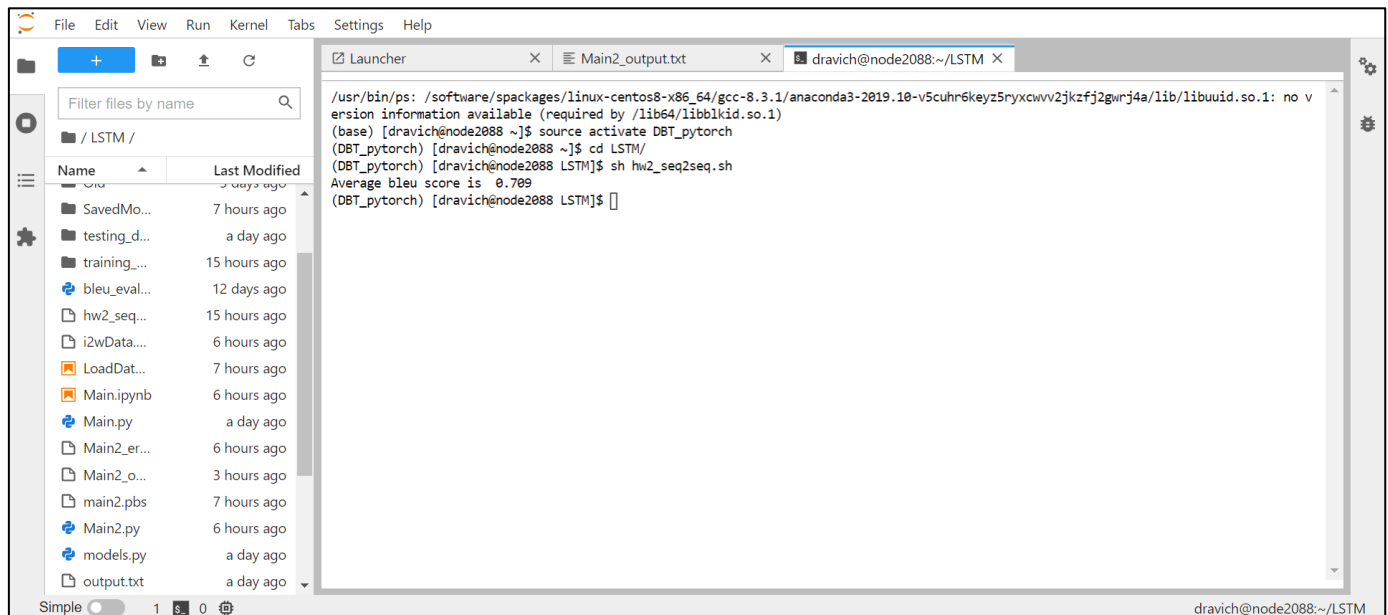


1.2.2 Attention Layer:

- An attention layer is implemented on encoder hidden states to improve the performance of the underlying model. At each decoding time step, the model can peep at different regions of the inputs.
- The decoder's hidden state and encoder's output are used as a matching function to generate a scalar, which is then processed through softmax, and the decoder's last new hidden state is delivered to the next step.

1.3. Training and Testing:

- Both training and testing model are defined in the MAIN.py, but only the train is executed while calling main. The training is done with the captions and the features provided in the assignment with the following parameters:
 - Batch Size = 10 (Trained with 126, 32, 10 where Batch size = 10 was generating lowest loss).
 - Epochs = 100
 - Learning rate = 0.0001
 - Loss Function = nn.CrossEntropyLoss()
 - Optimizer = Adam.
 - Training Sample Size = 1450
 - Video features dimension=4096
 - Video frame dimension=80
- Testing model is executed in the "test_run.py" the testing batch size = 10, using the model generated by the testing. The test function will generate the predicted caption for the test videos. The generated output caption is then compared with the ground truth caption so the BLEU_eval function can generate a BLEU score.
- With the given test data, the above-implemented S2VT model can generate a BLEU score of 0.709, as illustrated below:



The screenshot shows a Jupyter Notebook interface with a file browser on the left and a terminal window on the right. The file browser displays a list of files and folders in the /LSTM directory, including 'Main2_output.txt', 'Main2.py', 'Main2_er...', 'Main2_o...', 'main2.pbs', 'models.py', 'output.txt', 'testing_d...', 'training_...', 'bleu_eval...', 'hw2_seq...', 'i2wData...', 'LoadDat...', 'Main.ipynb', and 'Main.py'. The terminal window shows the execution of a script named 'hw2_seq2seq.sh' in the 'LSTM' directory. The output of the script is as follows:

```
/usr/bin/ps: /software/spackages/linux-centos8-x86_64/gcc-8.3.1/anaconda3-2019.10-v5cuhr6keyz5ryxcwv2jkzfj2gwrj4a/lib/libuuid.so.1: no version information available (required by /lib64/libblkid.so.1)
(base) [dravich@node2088 ~]$ source activate DBT_pytorch
(DBT_pytorch) [dravich@node2088 ~]$ cd LSTM/
(DBT_pytorch) [dravich@node2088 LSTM]$ sh hw2_seq2seq.sh
Average bleu score is 0.709
(DBT_pytorch) [dravich@node2088 LSTM]$
```