CPSC-8430: Deep Learning - Homework 2

This assignment is all about building, training and testing the model to generate captions for a given set of videos and analyzing the BLEU score.

Github Code Link: https://github.com/laxman2405/hw2/tree/main/hw2_1

As a part of training the model, we trained the model and the model is stored in file **hw2_model_laxman.h5**. Due to the large size of the file, it had difficulties in uploading and downloading from Github. So, I have uploaded the model to the drive link mentioned below.

Model Drive Link:
https://drive.google.com/file/d/1mb503m1A6AD1cYE9MCROsDY1p_cCQf5M/view?usp=drive_link

Download the model from above and place it in the same folder as testing_model.py to get the BLEU score. Also, this assignment is performed using CUDA, so testing this model on CUDA supported machines will attain results without any issues.

Instructions to run the code

The code is run in two steps ie., training and testing. The training_data and testing_data folders can be downloaded from the provided **MLDS** dataset.

1. Model Training

This is the first step to train the model from ground level and this is responsible for iterating over multiple epochs, vocabulary building and training the model. We are performing training in **training_model.py** and it expects 2 arguments. First, the training_data feat folder and second, training_label.json file as given in the data set. Below is the example,

python3 training_model.py /local/path/training_data/feat /local/path/training_label.json

Replace /local/path with the path in your local system. This will run for 200 epochs and generates required trained model and object files needed for testing.

2. Model Testing

For testing, there are a couple of options.

a. You can directly run **testing_model.py** before training the model, but it requires the trained model (.h5), object files (.obj). If those files are not present, please choose option (b).
b. Here, you have to follow **Step-1: Model Training** which will generate the required files for testing and run the below command.

   sh hw2_seq2seq.sh /local/path/testing_data/feat captions.txt

Replace /local/path with the path in the local system and this will generate a new file **captions.txt** containing all the predicted captions for the videos, along with the BELU score.

Below is the brief description of each of the python files used for generating captions.

Training_model.py

1. TrainingData Class

- This class handles all the preprocessing of the training videos and corresponding captions.
- It tries to load captions from training_label json file and tokenizes them using a word-to-index dictionary and returns a pair of video features and tokenized captions.

2. train_batch_data()

- This function tries to sort the captions by length and pads the captions to the maximum sequence length in the batch.

3. train_model()

- This is the function to train the actual Seq2Seq Model by iterating over **200 epochs.**
- It takes the model, train_loader and loss function as parameters and computes the average loss at each iteration.
- At each iteration, passes video features and captions to the model, captures loss using **CrossEntroypLoss with learning ratio of 0.001** and uses **Adam Optimizer.**
- Stores the average loss at each iteration in loss_history.

4. main()

- To load the word to index mappings from the training videos and json file and dump into **word_to_index.obj file**. Also, set up the train data loader with a **batch size of 256.**
- Initializes the Encoder, decoder and Seq2Seq model with the required parameters.
- Then, the loss function is passed to the train_model() to compute losses obtained after each epoch iteration.
- The trained model is saved in **hw2_model_laxman.h5** file.

Process_videos.py

This file prepares to generate vocabulary mappings by filtering out low-frequency words. Below are the steps involved in the get_mappings()

- Initially, read all the video features files available in the given training dataset and load them into a dictionary.
- Next, load the captions of those videos from the training json file and using the regular expressions remove unwanted characters.
- Now, to build vocabulary we set **min_word_count to 4** and filter out words fewer than 4 and create word-to-index and index-to-word mappings using special tokens such as <pad> (padding), <bos> (beginning of sentence), <eos> (end of sentence), and <unk> (unknown).

- We list out a few video statistics and return back index mappings to the training_model.py file.

Seq_2_Seq_model.py

This implements a sequence-to-sequence model for captions generation and has few components as listed below

1. Attention

- This class is responsible for ensuring the decoder is paying attention to the relevant parts of the encoder's input while generating outputs.
- Here, it is calculating the attention weights between current decoder state and encoder outputs.
- Generating context vectors by weighting encoder outputs based on attention scores.

2. Encoder

- It usually processes the input and compresses it into a hidden state for further processing by the decoder.
- Here, we use components like compress, dropout and gated recurrent unit (GRU) for processing input sequence.

3. Decoder

- This finally generates the output based on the encoder's hidden state and context vector from the attention mechanism.
- Here, it includes embedding layers, GRU, attention mechanism components for processing.
- Uses features such as teacher forcing, inference mode and beam search for improving the output quality and accuracy.

4. Seq2Seq

- This is used as a wrapper to setup interaction between encoder and decoder and invokes training or inference methods based on the mode.
- Training mode uses teacher forcing, Inference mode generates sequences without ground truth and beam search improves outputs during decoding.

Testing_model.py

1. TestDataSet Class

- To load the video feature clips from the testing_data folder of the dataset and returns a list of videoIDs and corresponding features.

2. compute_blue_score()

- This function computes the BLEU score between ground truth captions and the model's predicted captions.

3. test_model()

- This model takes in the trained model, test label json file, test loader and index to word pickle file as inputs.
- Invokes the sequence model with inference mode to generate captions for test videos.
- Converts predictions to readable captions, stores them, and prints the average BLEU score using ground truth labels.

4. main()

- Loads the index_to_word pickle file and test_loader using DataLoader class.
- Load the trained model and invoke test_model to generate predictions and calculate BLEU score.

Training Model Output

Once we run the training_model.py file as mentioned above, we get the below output showing a few video statistics like video feature shape, total number of captions, unique captions and maximum captions for a single video.

```
[lmadipa@nodeau02 HW2]$ python3 training_model.py /home/lmadipa/HW2/training_data/feat/ /home/lmadipa/HW2/training_label.json
Filtered vocabulary: 2423 words (min count: 4) from 6097 total words.

Video feature shape: (80, 4096)
Total captions: 24232
Unique captions: 20074
Max captions for a single video: 37
```

After running for 200 epochs, training is completed in **8939.739** seconds as shown in below figure.

```
Epoch: 195/200: Average Loss: 1.565
Epoch: 196/200: Average Loss: 1.565
Epoch: 197/200: Average Loss: 1.563
Epoch: 198/200: Average Loss: 1.563
Epoch: 199/200: Average Loss: 1.561
Epoch: 200/200: Average Loss: 1.561
Training completed in 8939.73954820633 seconds.
```

Testing Model Output

After running the hw2_seq2seq.sh file by passing command line arguments, we got an average BLEU score of **0.643** as shown below, which resembles the accuracy in predicting the captions for the testing videos.

```
[lmadipa@nodeau02 HW2]$ sh hw_seq2seq.sh /home/lmadipa/HW2/testing_data/feat/ /home/lmadipa/HW2/captions.txt
  model = torch.load(model_path)
Average BLEU score: 0.6433691236195409
```

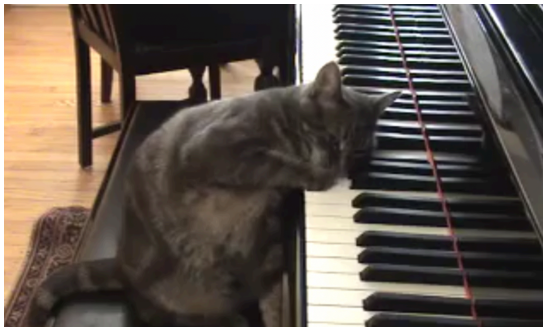
Videos with Generated Captions

lmadipa@nodeau02:~/HW2 | captions.txt | training_model.py

```
38 JntMAcTlOF0_50_70.avi,man is walking in a garden
```

lmadipa@nodeau02:~/HW2 | captions.txt | training_model.py

```
54 f9Won2JpOEU_60_80.avi,cat is playing a
```



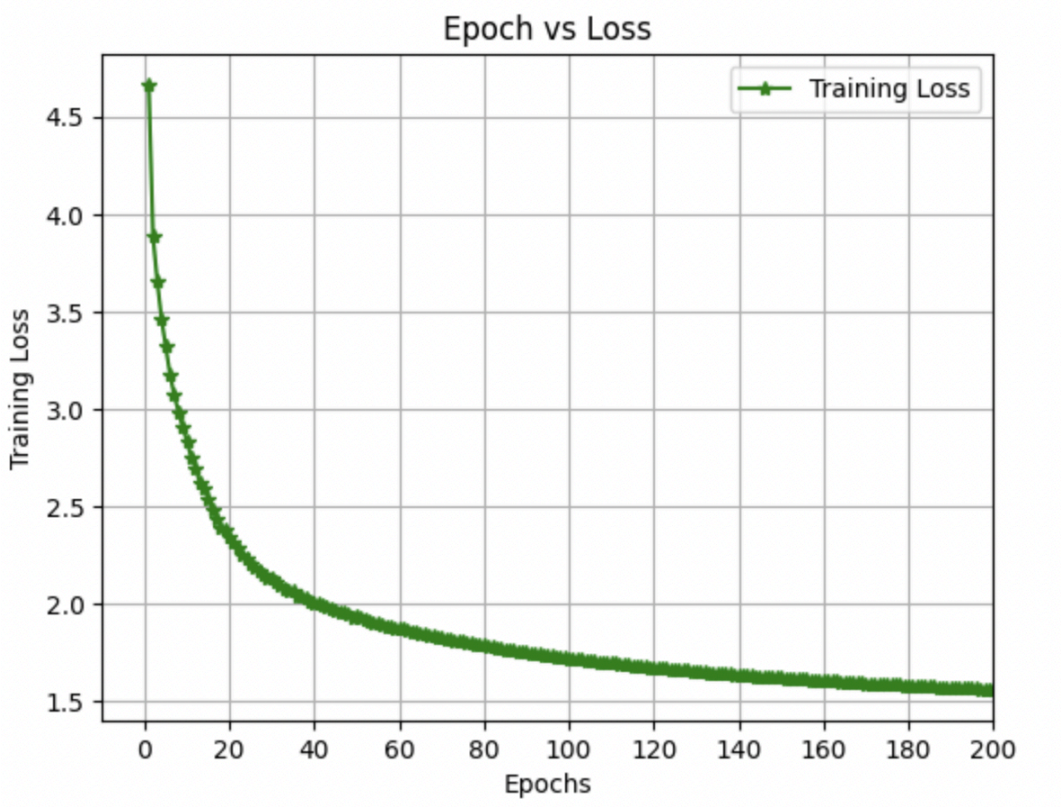
lmadipa@nodeau02:~/HW2 | captions.txt | training_model.py

```
49 TZ860P4iTaM_15_28.avi,cat is playing the piano piano
```

lmadipa@nodeau02:~/HW2 | captions.txt | training_model.py

```
13 6q1dX6thX3E_286_295.avi,man is talking
```

Graph representing Epoch Vs Training Loss



As we see from the above graph, we trained the model for 200 epochs and If we observe the initial loss was very high, but as the epoch increases loss greatly reduced, indicating the model is learning with the given parameters, and as it reaches the end, the model might converge and there might be minimal chance of improvement in training.