Deep Learning Homework 2: Video Captioning

Mary Damilola Aiyetigbo

GitHub Link: https://github.com/dimky01/8430-Deep-Learning/tree/main/hw2

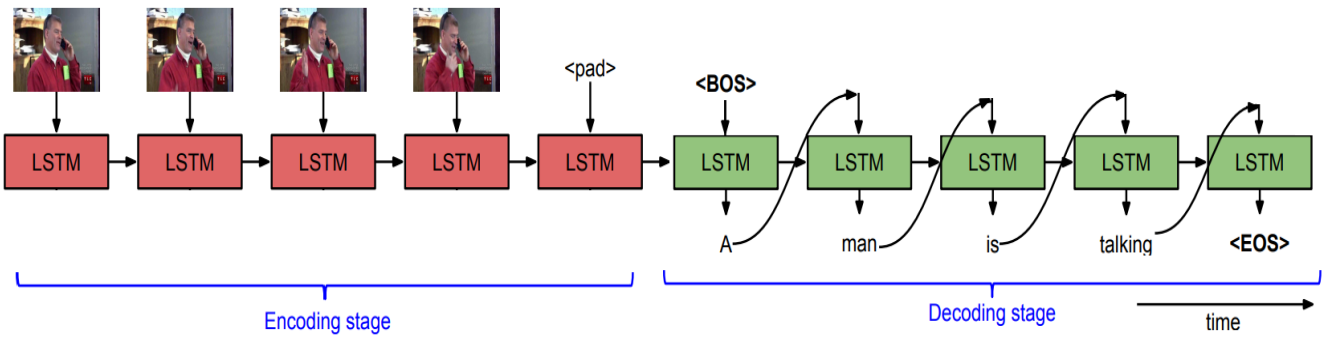
In this homework, a sequence-to-sequence model was created and trained to generate captions for videos. When a video image is fed into the trained model, the model can generate a description of what is happening in the video, called "Caption."

To implement this model, the following python files were created

1. **Dataloader.py**: This code reads in the input feature files and caption files using custom dataset classes (VideoDataset for the training data and TestDataset for test data). The output of these classes was fed into PyTorch Dataloader to load the dataset in batching into the model. Vocabulary and data preprocessing was also implemented in this file
2. **Model.py**: The encoder, decoder, and attention models were defined in this file
3. **Train.py:** This file contains the training process of the model
4. **Test.py**: code used to test model with unseen data after the training. The output of this code is the average BLEU score and generated captions in a *.txt* file
5. **hw2_seq2seq.sh**: shell command to run the test file

MODEL DESCRIPTION

The sequence-to-sequence model (SV2T) consists of two recurrent neural networks (RNN): Encoder and Decoder. The encoder processes the input while the decoder generates the output. Gated Recurrent Unit (GRU) was used for both the encoder and decoder. The output of the encoder was used as input to the decoder, and the last hidden state of the encoder was used to initialize the hidden state of the decoder. Below is an image illustrating the model architecture from the encoder processing the video input using the GRU layer and the output fed to the GRU layer of the decoder to predict the caption words at every time step.



ATTENTION MODEL

To improve the model's performance, the attention model was implemented. This model allows an RNN to pay attention to specific input parts that are considered important, which improves the model's performance. The output of the encoder and hidden state of the decoder were used as a matching function to calculate the attention weights.

Vocabulary

A dictionary of words in the training captions was created, and words that occur with a frequency less than three counts were excluded. The dictionary consists of the most frequent word with min count of 3

Tokens：<PAD>, <SOS>, <EOS>, <UNK>

- <PAD>: Pad the sentence to the same length
- <SOS>: Begin of a sentence, a sign to generate the output sentence.
- <EOS>: End of sentence, a sign of the end of the output sentence.
- <UNK>: Use this token when the word isn't in the dictionary or just ignore the unknown word.

Teacher Forcing

During training, the model predicts a word at every time step *t,* and loss is computed. This word will be fed into the model as input at the next time step *t+1*. However, if the predicted word is wrong, the result at the other time steps will most likely be incorrect, resulting in a high error rate, and the model might not learn. Therefore, teacher force was used by feeding the next word in the ground truth caption as input to the decoder at every time step. However, the RNN model will need to use the prediction as input for the next prediction during testing. This causes a discrepancy between training and testing, and this might lead to poor model performance and instability known as Exposure Bias. To avoid exposure bias, the teacher's force ratio was set to **70% (0.7)** so that the model could converge faster and generalize to the test data as well.

Model Training and Testing

Three models with different parameter settings were trained to evaluate the performance of each model. The BLUE score was used to evaluate the performance of each model. Below are the hyperparameters used across the models.

Hyperparameters

- Loss function: Cross Entropy loss

- Learning rate = 0.0003
- Input Size = 4096
- Hidden Size = 512
- Optimizer = Adam
- Weight Decay = 0.00001
- Dropout Rate = 0.2
- Batch size = 64
- Min word count = 3
- Teacher Force Ratio = 0.7

The variations between these models were the decoder hidden state, number of epochs, and the size of the embedding dimension. These variations had an impact on the BLEU score when the model was evaluated on the test data

Model0:

For this model, the hidden state of the decoder was initialized to zero instead of using the encoder hidden state

- Decoder Hidden State: initialized to zero (0)
- Embed Dimension=256
- Epochs = 120
- **Average BLEU score = 0.59**

The average BLEU score for the model was the lowest

Model 1:

- Decoder Hidden State: Encoder hidden state
- Embed Dimension=128
- Epochs = 100
- **Average BLEU score = 0.62**

Model 2:

- Decoder Hidden State: Encoder hidden state
- Embed Dimension=256
- Epochs = 120

- **Average BLEU score = 0.635**

This model had the highest bleu score of 63%. These results show that initializing the decoder with the encoder's state improves the performance of the model

The graph below shows the plot of the model loss at every epoch for Model2.