CPSC 8430: Deep Learning – Homework 2

Sadegh Sadeghi Tabas

GitHub: https://github.com/sadeghitabas/CPSC8430-DeepLearning.git

Video Caption Generation

In this homework a sequential-to-sequential model presented, trained and tested in order to generate a caption for videos. The basic idea of Video Caption Generation is to input a video and get a stream of captions for the actions happening in the video. This is achieved by following deep learning techniques and training of the dataset provided.

Requirements:

- Python 3
- torch 1.7.1
- scipy 1.6.2
- MSVD Dataset (1450 videos for training, 100 for testing)

Dictionary:

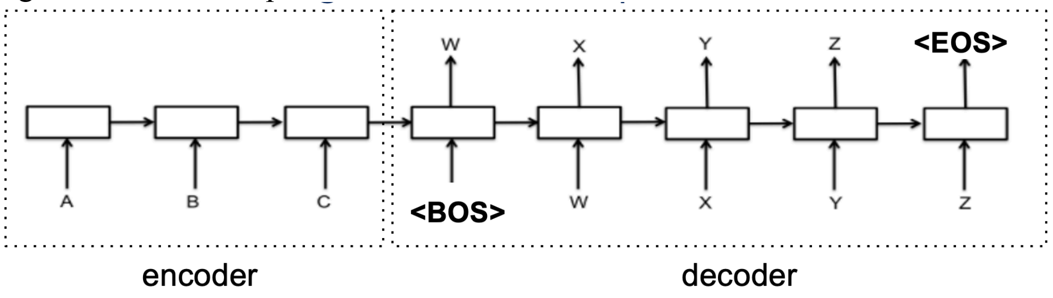
To implement the seq2seq model, first a dictionary function created to convert words to indices and vice versa. Below tokens has been used to define the dictionary function.

- Dictionary - most frequently word or min count
- Other tokens : <PAD>, <BOS>, <EOS>, <UNK>
  - <PAD> : Pad the sentence to the same length
  - <BOS> : Begin of 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.

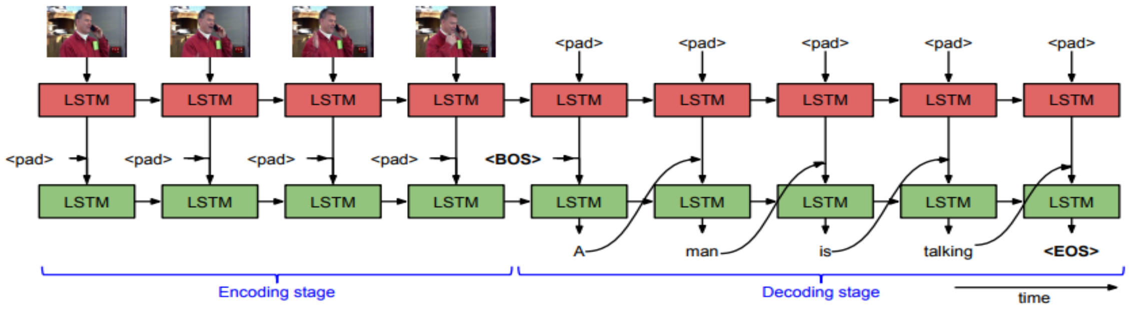
As we cannot feed the input data straight to the model, we need to split the sentences into words and then representing those words as numbers. The words are uniquely indexed in the dictionaries.

Base Model:

Next, the baseline for seq2seq model (S2VT) implemented. In this model 2 layers of GRU (RNNs) implemented, in the first layer of the RNN, the Video is processed and encoded (encoderRNN), and an output is generated with the help of the decoder (decoderRNN). In the decoding process, tokens are used to segment the captions based on the beginning and ending verse and perform processing over the video to produce the actual words.

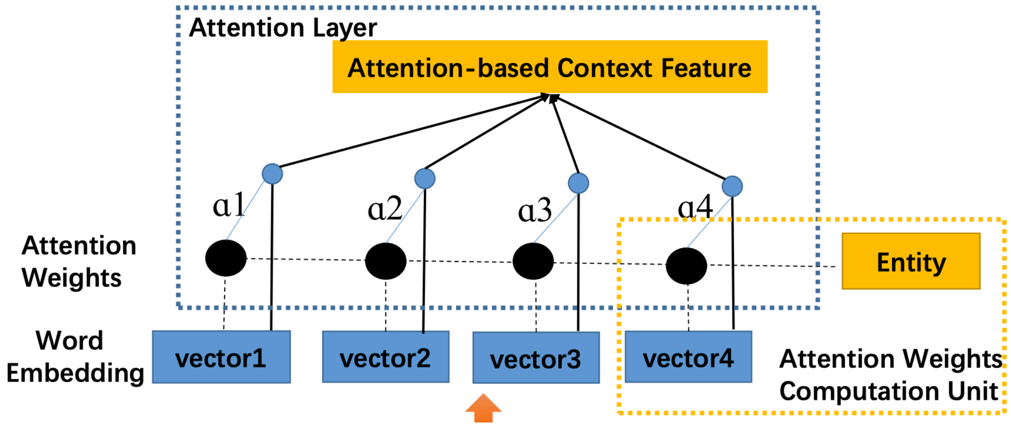


Below figure illustrates the processing of encoding the video using encoderRNN (a GRU layer) and generating video using decoderRNN (a GRU layer).



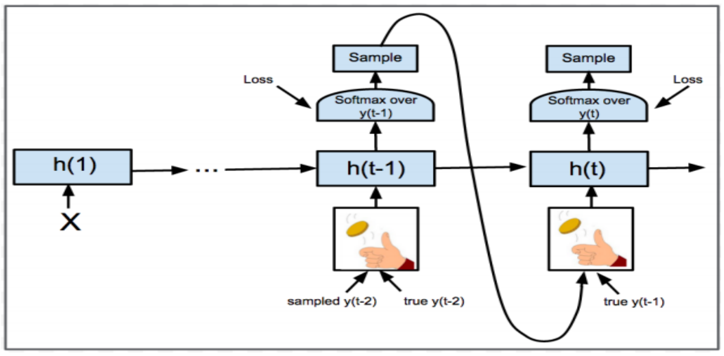
Attention Layer:

In order to enhance the performance of base model, an attention layer implemented on encoder hidden states. This allows model to peek at different sections of inputs at each decoding time step. The structure of attention layer implemented in this study is similar to the one presented by Shen and Huang (2016). The hidden state of the decoder and output of the encoder are used as a matching function to obtain a scalar, and then this scalar is passed through softmax and the last new hidden state is sent to the next time step of the decoder.



Schedule Sampling:

At inference, the unknown previous token is replaced by a token generated by the model itself. This discrepancy between training and inference can yield errors that can accumulate quickly along the generated sequence. This problem called exposure bias" problem. To solve this issue based on the study of Bengio et. al, (2015), we need to feed (groundtruth) or (last time step's output) as input at odds while we are training the model. The below figure illustrates the process of schedule sampling method implemented in this homework.



Once the model and above-mentioned modules implemented, the model trained using MSVD train dataset (including 1450 videos and their caption in json label file). Then the model weights saved and has been used to evaluate the model using MSVD test dataset (including 1450 videos and their caption in json label file). The CrossEntropyLoss function is considered as loss function of the model. At the final stage, in order to evaluate the model performance, the Bleu_eval function called and the BLEU score shows the model structure performance.
In order to find the optimal structure of the model different hyperparameter set investigated and the result is presented in the below table.

Hyper-parameter set	Num of Epochs	Learning rate	Batch size	Num of Hidden layers	Dropout rate	Teacher Learning Ratio	Vocab size	BLEU score
1	100	0.001	128	512	0.3	0.7	Min count > 3	0.69
2	200	0.0005	128	512	0.5	0.7	Min count > 3	0.65
3	150	0.0005	128	512	0.2	0.7	Min count > 3	0.686
4	100	0.001	128	512	0.1	0.7	Min count > 3	0.69
5	80	0.001	128	512	0.3	0.7	Min count > 3	0.688
6	**80**	**0.0001**	**128**	**512**	**0.1**	**0.7**	**Min count > 3**	**0.692**