HOMEWORK 2

VIDEO CAPTION GENERATION

GitHub: https://github.com/Joel-kiran/DeepLearning/tree/main/HW2/HW2_1

Introduction

With vast amount of data produced every second, generating caption for the video has become an important task, as it can be used for sentiment analysis of video content and describing the video for the visually impaired. With the advancement of sequence-to-sequence model we can generate the captions for the video, we will be using GRU for encoder and decoders network, we will also be using attention mechanism to focus on a particular input position to predict the output at that position. We will be using either greedy or beam search for predicting the captions. Finally, we will be using BLEU score to evaluate the models predicted output.

Dataset

We will be using the MSVD (Microsoft video description) Dataset, which contain about 1450 videos for training and 100 videos for testing, each video is of variable length, we are also provided with a caption file which contains the captions for each video in the training and test set. We are also provided with encoder output features for each video in the form of .npy file, which can directly be fed to the decoder network. Hence, I will be only using the .npy file for our training and testing purpose, as I will be building only the decoder network.

Preprocessing

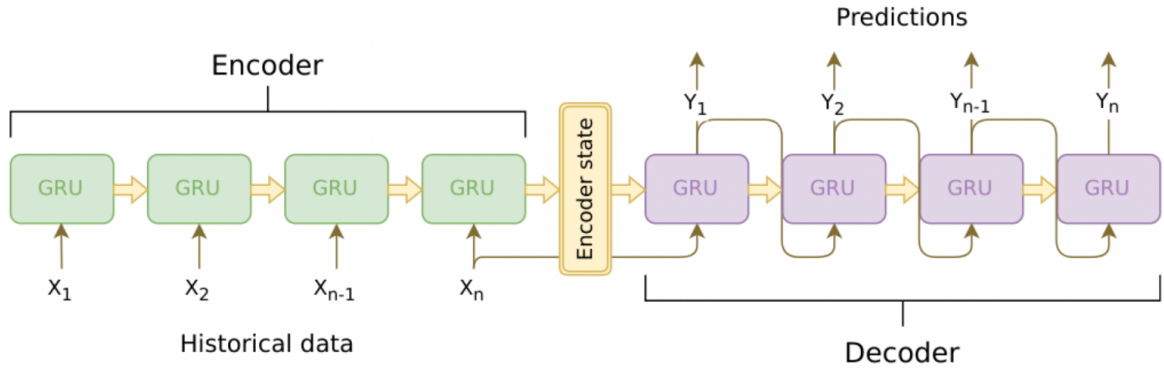
I will be using the encoder decoder sequence to sequence model to solve this problem, but since the encoder output features for the video are already given in the dataset, I'll be using these features directly to build my decoder network.

The feature for each video is 80*4096, which means each video as 80 frames and 4096 features. Initially I will be creating a vocabulary set by parsing through the training captions and storing the vocabulary, I will maintain two dictionary to store word to index and index to word mapping which will later be used during the time of training and prediction. The vocabulary will also contain four additional terms "<BOS>", "<EOS>", "<PAD>" and <"UNK>" to denote beginning of string, end of string, padding and unknown string. PAD is used to generate caption strings of same length. Each training video caption is parsed and the corresponding new caption will be generated and stored which will contain "<BOS>", "<EOS>", "<PAD>" and <"UNK>". The same process is repeated on the validation set as well. Vocabulary size is 2411 after performing the above steps. The word to index and index to word dictionary is stored in the form of a pickle file which is later used in predict.py file for prediction.

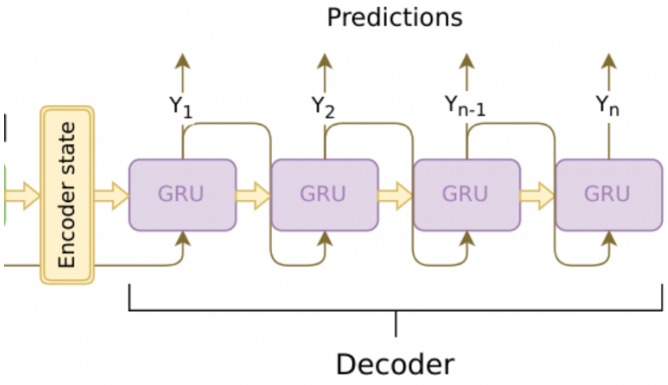
Training

```
S2VT(
  (attn): Attn(
    (nn_attn): Linear(in_features=1024, out_features=1, bias=True)
    (dropout): Dropout(p=0.3, inplace=False)
  )
  (gru1): GRU(512, 512, dropout=0.3)
  (gru2): GRU(1536, 512, dropout=0.3)
  (embedding): Embedding(2411, 512)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc1): Linear(in_features=4096, out_features=512, bias=True)
  (out): Linear(in_features=512, out_features=2411, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

The above layers were used for building the model, GRU is used instead of LSTM, because GRU executes faster compared to LSTM because it as lesser number of parameters and the amount of memory used is also less. We use attention layer to focus on the specific parts of the video to generate a word at Tn in caption. We also use embedding layer, embedding layer can be thought of as an alternate to one-hot encoding along with dimensionality reduction, neural networks can't understand text, hence we need to convert them to numbers in order to feed to network, here we try to use embedding layer to convert the captions to fixed size output so that they can be fed to the network while training. We use SoftMax at the last year in order to predict the output word which as the highest probability.

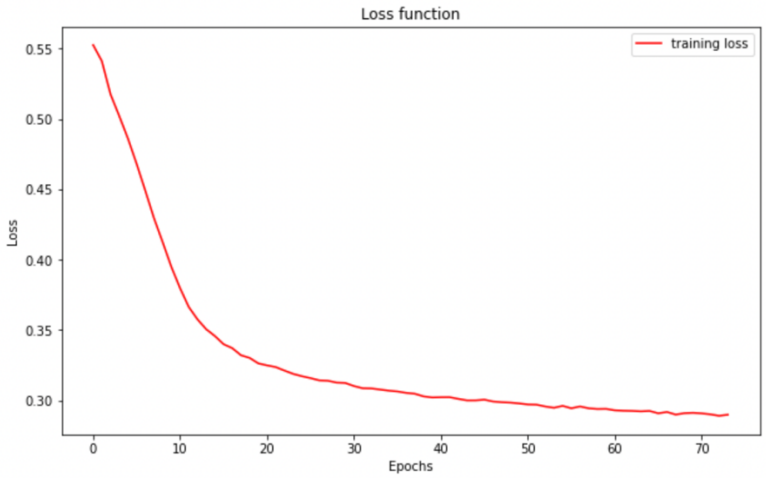


The above network is ideally used to perform our task, which consists of encoders and decoders, since our dataset consists of the output from encoder layer, we don't need to worry about building the encoders, we need to concentrate on building only the decoder network. Hence, we will use the below model to perform our tasks.



A pictorial way to represent on how the model is trained is shown in above figure, to predict word at time step Tn we use the hidden state at time step Tn-1 and either the predicted output at Tn-1 or the actual output from the training caption at Tn-1 to get output at Tn, this is decided by the teacher forcing ratio. We later collect this output from GRU and feed to Dense layer followed by SoftMax, which is used to get the word with highest probability at Tn. This process is repeated until we get "<EOD>" which is when we stop.

The model is trained for 70 epochs with batch size 64, output sequence length is 80, loss function is nll_loss, optimizer is adam and the learning rate is 0.001. we save the model whenever the training loss reaches the minimum for a particular epoch.

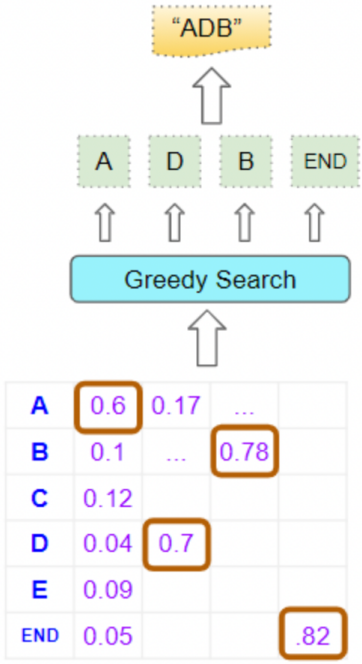


The above figure represents the training loss curve, we can clearly see that as we increase the epochs the training loss decreases.
The model with the least loss is saved which will later be used during prediction.

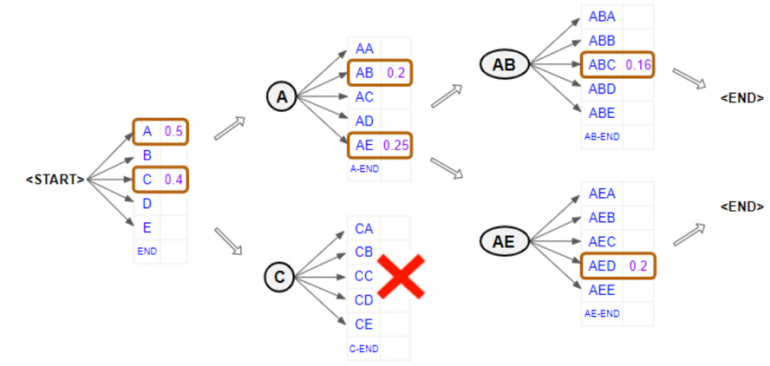
Testing

I will be using the videos given in the test repository and the model which was saved during the training step will be used for testing, there are 100 videos. We can either use beam search or greedy search to generate the caption. Greedy search uses the highest probability at each step to predict the word, it doesn't consider the sequence to predict the next output. While beam search

considers the sequence with highest probability to predict next word in the caption, number of sequences to be used to predict the next word depends on beam width. Beam search does a lot of computation due to which it is slower compared to greedy search, but beam search as a better performance compared to greedy search which can be calculated using the BLEU score.



The above figure given an idea of how the greedy search works, highest probability word is picked at each position to generate the caption.

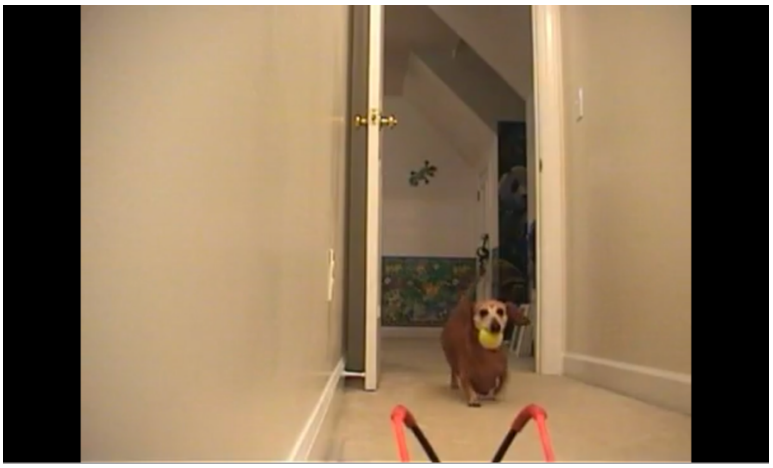


Beam Search example, with width = 2 (Image by Author)

The above figures gives an idea of how the beam search works, we can clearly see that at position1 A and C as highest probability hence its used to generate the sequence at position 2, at position2 AB and AE as highest probability hence it will be selected to generate the word at postion3, at postion3 ABC and AED as highest probability hence it will be selected, finally out of ABC and AED we can pick the one with highest probability, so for the above figure, caption will be AED.

Results

For the above video sample, the caption generated by the model is “Someone is preparing shrimp”



For the above video sample, the caption generated by the model is “A dog is running”.

Using the greedy search, the BLEU score on the testing set is 0.717, using the beam search with a beam width of 4 the BLEU score on the testing test is 0.7260, using beam search with beam width of 2 the BLEU score on the testing set is 0.72639.

Conclusion

I can conclude by stating that if we can use a larger dataset and train the network for longer time may be 1000 epochs, the performance of the model will be better.

Files

hw2_seq2seq.sh <test_data_directory_path> <output_file> - this file can be used to predict the captions for the video samples, it takes in two arguments, <test_data_directory_path> is used to

indicate the directory where the test data is present, <output_file> is a file where the captions predicted by the model will be stored.

model_seq2seq.py – is the file that will be used for training, you might need to edit the paths in the file to point to the directory containing the training samples, post which you should be able to run the file.

Result.txt – contains the predicted captions by the model on the test set.

Requirements
Numpy, pickle, pandas, pytorch, torch vision, pdb