Brad Sanders

Deep Learning HW2

This project attempts to solve the video caption generation task with the use of pytorch and the MLDS dataset. To accomplish this, I utilize a sequence-to-sequence model with a RNN as the encoder and decoder which takes in video features as inputs and outputs a caption for the video. This required me to train a model by first preprocessing the data and then feeding it to the network. Afterwards I would save the checkpoint on test it on unseen data.

To process the data, I set up a class to create a custom dataset to utilize the pytorch dataloader. This class grabs all the videos and captions and relates them to each other by holding same index between each caption and video id in their respective lists. The captions are then used to develop a set of vocab words for the model to use. All captions are padded to the same length of 12 for ease of use. Each caption is tokenized, and each word is encoded to an index value. A dictionary saves the mapping of each word to the specific index. Words are added to the vocab if they hit a certain frequency threshold which is currently set to three. I adjusted this value in order to prevent the model from predicting words that only appeared a few times in the target labels to help improve the accuracy of the model. From here I set up the dataloader to provide a tokenized caption using the index value as a target value alongside the video feature data as the x input.

The sequence-to-sequence model consists of two RNN's. The first is an encoder which takes the video features as input and encodes it to a latent space. This is then used as input to the decoder which produces a one-hot vector containing the index for a word. Both the encoder and decoder use a LSTM layer which uses the embedding to generate an output and hidden values. The output from the encoder is in the size (12, batch size, hidden size) where 12 is the length of the captions. The decoder utilizes a SoftMax layer in order to normalize the values between 0 and 1. To train the model the hyperparameters could be adjusted, these being the optimizer, batch size, number of epochs and the learning rate as well as the inputs to the RNN's being the hidden size and the embedding size. I determined that a smaller hidden worked better for my purposes as it reduced the complexity enough to avoid memory errors. The loss function used is cross entropy which is necessary for classification problems. In the best model the loss started at around 6.3 and dropped to a value of about 3.0 during training.

To evaluate the model the bleu score was used by calculating the number of correctly identified words in the output and then multiplying this value by either one or $e^{1-r/c}$ where r is the length of the actual caption and c is the length of the predicted caption. Bleu scores were averaged across all inputs to determine a final score. After testing it was determined that the model failed to produce any good results. During the training the model would change its predictions however many the value predicted would often lie outside the vocab range and be converted to an unknown token. Despite this during testing the model would always predict 0 and never change despite the input. To fix this I attempted to reduce the size of the captions to 10 and set a minimum caption length to avoid overfitting the pad tokens which had an index of 0. However, this did not change the results and the model still showed no response to the individual inputs. Despite finetuning the various variables, the performance never changed and long training times hindered my ability to produce a functional model.