CPSC 8430- Deep Learning

Home work 2

Github repository Link:

https://github.com/im3883/CPSC8430-HW2/tree/main/hw2-1

Model download link:

https://drive.google.com/file/d/1sWlxkEolYYcwXvLyPg6c-uF3GrhjzgOr/view?usp=sharing

Submitted by

Mohammad Imtiaz Hasan (hasan2@clemson.edu)

Video Captioning Code Overview
The code will be used to train a deep learning model in generating captions for video clips. This model uses video features as the input and generates as output the captions describing what is contained in the videos. In short, the whole process involves loading video features with their corresponding captions, training the model on caption prediction, and then model performance evaluation.

Key Components and Functionality:

Dataset Preparation:
Loading and preparation of data will all happen in the first section of the code to be used in training. The features from the video and corresponding captions of these videos will have stored files. Captions are being loaded from a JSON file, while video features are loaded from .npy files.
The task of loading all video features and captions is handled by the Dataset_Creation class. It transforms the text of captions into numerical indices using a vocabulary object. That will become useful when fed into the model. There is also the option to load all video data into memory, which speeds up training, but increases memory usage. If not loaded into memory, video data will be loaded from disk during training.

Train Models:
It provides the class responsible for training the model. It initializes the model, sets up the optimizer-Adam optimizer with a learning rate of 0.001-and utilizes the loss function-Cross-Entropy loss.
The model can run on the presence of a GPU available, otherwise it will run on a CPU by default. The reason is training such large models can be quite computationally expensive, and it really makes it much faster with a GPU.
Actual training occurs within the function train(). It reads in a batch of video features and their corresponding ground truth captions. The model attempts to predict the captions. Comparisons of such predicted captions with the ground truth captions are made, and loss is computed. The weights of the model are adjusted by the optimizer so as to minimize that loss. The model is evaluated using the eval() and test() methods. The eval() method runs the model on a small subset of test data in a loop during training to monitor progress, and the test() method runs on the full test set at the end to evaluate final performance.

Encoder, Decoder and attention layer:
There are two main parts in the model itself: the encoder and the decoder.
The encoder treats the features extracted from a video and converts them into a hidden representation of the video. This is further passed through a decoder, which transforms this hidden representation of a video into a sequence of words for the caption. These different components, therefore, interact both during training and at inference-that is, when this model is generating captions for new videos. The attention layer is used to peek into different

regions of the input. This layer is set into the hidden layer of the decoder function to pick out the important information from the input. The attention layer is built using 3 hidden layers.

Vocabulary and Preprocessing:
The code has a vocabulary class which directs, through a vocabulary object, the transference of words into numerical indices. This is pertinent because deep learning models do not deal with words but with numbers. The vocabulary ensures each of the words in captions maps onto a number and will come in handy in converting the model's predicted numbers into words during the generation of the caption.

Training Procedure:
The training_model() function regulates the flow of the training loop. This function arranges for both training and testing datasets, model initialization, and then launches the training. Training is performed on a multiturn basis-a number describing how many times the model has seen all examples in the dataset. We have trained the models for 20 epochs in this case. Once an epoch is done, the model evaluates its learning.
Once the training is done, the model is saved to a file that would allow it to be reloaded later for use without training from scratch.

Evaluation and Testing:
The performance of the trained model is checked by the BLEU scores, which tell about how similar the generated captions of the model are to the ground truth captions.
testing_model() function allows testing of the model that has been trained; it loads a previously saved model and tests its performance on the test dataset.

Observations:
In this home work, we have built 2 models with varying batch size and dropout rate. The first model uses a batch size of 16 and drop out 0.2 and trained for 20 epochs on the 1450 training videos dataset. After the training is done, we tested the model performance over the 100 testing video dataset. We got a BLEU score of 0.603.
The second model uses a batch size of 32 and drop out 0.4 and trained for 20 epochs on the same training data set and tested on the same testing data set. We got a BLEU score of 0.672 for this model.

Hence we selected the second model for the final testing which is uploaded on google drive for using.