CPSC 8430 Deep Learning

Danish Bhatkar (C15563212) – dbhatka@clemson.edu

GitHub Repository Link

HW2 – Video Caption Generation

Problem Statement

Generate a video caption for an input video using sequential-to-sequential model. The input for the code will be a video, and the output will be a stream of captions describing the actions in the video along with the average BLEU score.
The above is achieved by Recurrent Neural Networks.

Dataset

Provided in class– contains 1450 videos for training and 100 videos for testing along with video features and captions/labels.

Introduction

This project involves building a Sequence-to-Sequence (Seq2Seq) model aimed at generating captions for videos. In this assignment, a Seq2Seq architecture comprising an encoder-decoder model with attention mechanisms, is trained on video data with captions.

The dataset consists of videos, each associated with caption data provided in '.json' format. This dataset is pre-processed and transformed to feed the model, which then learns to generate captions given video features.

Model Architecture

The core architecture of this Seq2Seq model is composed of the following:

The provided Seq2Seq model consists of three main components: the Encoder, Decoder with Attention, and the Seq2Seq framework itself.

1. The **Encoder** processes video features by applying a linear transformation followed by a ReLU activation function, converting the input features into a hidden representation.
2. The **Decoder** utilizes the encoded features and generates captions. It employs an embedding layer for the input captions, calculates attention weights to focus on relevant video frames, and combines these with the embeddings before passing them through an LSTM.
3. Finally, the **Seq2Seq** class orchestrates the training process by connecting the encoder and decoder, allowing the flow of information from video features to generate captions. This architecture effectively captures the sequential nature of the data while leveraging attention mechanisms for improved context awareness.

Model Parameters and Evaluation

- Epochs = 100
- Learning Rate = 0.00005
- Feature Size = 4096
- Size of Hidden State in LSTM = 512
- Optimizer = Adam

- Vocab Size = 3594 (min frequency=3)

Dataset Details

The dataset used in this project consists of two parts:

- Video Features: Stored in .npy format, these files contain the pre-computed features of the videos.
- Captions: The captions are stored in JSON files (training_label.json and testing_label.json), where each video ID corresponds to one or more caption sentences.

Each video feature file is loaded and passed through the encoder, which generates the context vector. The decoder, in turn, generates the caption based on the context vector and attention weights.

Training Process

- Epochs: The training loop runs for 100 epochs.
- Model Mode: The model is set to training mode with 'seq2seq_model.train()'.
- Batch Iteration: For each epoch, it iterates over batches of features and captions from the training dataloader.
- Gradient Reset: The optimizer's gradients are reset at the beginning of each batch using 'optimizer.zero_grad()'.
- Forward Pass: The model performs a forward pass to predict outputs, using 'captions[:, :-1]' as input (excluding the last token).
- Targets Preparation: The targets are prepared by reshaping the captions to align with the predicted outputs using 'targets = captions[:, 1:].reshape(-1)'.
- Loss Calculation: The outputs are flattened for loss calculation, and the loss is computed with the specified criterion.
- Backward Pass: The loss is backpropagated using 'loss.backward()' to compute gradients.
- Parameter Update: The optimizer updates the model parameters with 'optimizer.step()'.
- Progress Monitoring: The loss is printed every 10 steps to monitor training progress, indicating the current epoch and step.

Evaluation

The evaluation function for the Seq2Seq model sets the model to evaluation mode using 'seq2seq_model.eval()'. Within a 'torch.no_grad()' context, it performs a forward pass to generate predictions based on input features and extracts predicted word indices using 'torch.max'. Both predicted indices and reference captions are converted to words via the 'denumericalize' function, filtering out '<PAD>' tokens. The BLEU score for each sample is computed with the 'calculate_bleu' function and accumulated. The function prints reference and predicted captions along with their BLEU scores for inspection. Finally, the average BLEU score is calculated and printed, providing a metric of the model's performance on the test dataset.

The loss obtained at the final epoch is ~0.128 and the average BLEU score is 0.09.