CpSc 8430 - Deep Learning
Project 2 Report

Giovanni Martino
giovanm@clemson.edu

Introduction

The code for this project can be found at the following repository:
https://github.com/rein5/cpsc8430-deep-learning

The folder *hw2/hw2_1/* contains the code to train and test the model, as well as the final files for trained model and vocabulary.

There is also a script that can be used to evaluate the model on a given test dataset.

Data Preprocessing

A dataset is computed from the provided data, and it consists of a set of (X, Y) pairs, where:

- X: is a sequence of 80 feature vectors (of size 4096), corresponding to 80 frames in the input video
- Y: is a sequence of words for a caption corresponding to X, one-hot encoded over a vocabulary of words

Since for any given video we can have multiple captions, each of these is mapped to a separate (X, Y) example in the above dataset.

The vocabulary is determined by counting how many times each word appears in the training set, and then selecting the set of words that have at least **3** appearances. With such a design decision, the vocabulary size ends up being **2879**.

The captions are also preprocessed by padding punctuation with white spaces, and getting rid of any characters outside of: a-z, A-Z, ".", "?", "!", ",".

The vocabulary also contains the following special tokens:

- <BOS>: beginning of sentence
- <EOS>: ending of sentence
- <PAD>: used to pad a sequence of words to a given size
- <UNK>: used to represent a word not in the vocabulary

The captions are constrained to a maximum length of **20** characters (they are also padded to this length, if shorter).

Model Architecture

The model is made up of an encoder and a decoder, both of which are implemented with GRUs. GRUs were chosen over LSTMs since they are simpler and offer faster training. Both encoder and decoder GRUs have a hidden size of 640.

The model utilizes dropout for some of its layers for regularization. It also employs an **attention mechanism** to more strongly condition the decoder on certain steps within the input sequence, at each output timestep.

Additionally, the model uses an embedding layer to map between one-hot word encodings and dense embeddings, and an initial fully-connected layer to preprocess the feature vectors of the video frames, before these are fed into the encoder.

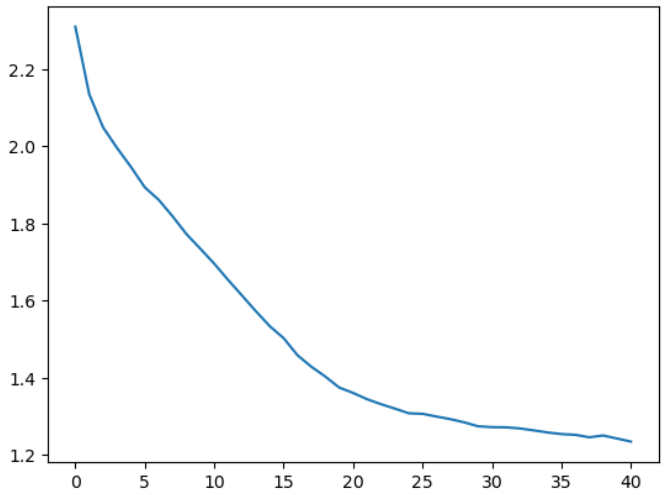
The standard decoding loop feeds in the output from the previous timestep as the new input, to predict the next word in the sequence (with the initial input word being the <BOS> token). On a few random timesteps we also use a **teacher forcing** mechanism (schedule sampling), which feeds in the actual output word at the previous time step instead of the predicted one. This helps improve model performance.

At test time, we also have the option of generating captions using **beam search**, instead of the simpler **greedy search** scheme. While with greedy search we pick the word with highest probability at each timestep, with beam search we keep track of the best *k* prediction sequences up to the current timestep, updating these at each step with *k* next word candidates, and again picking the best k. At the final step, we pick the overall best sequence. This can improve model performance, but also increases inference time, since we are running the model multiple times for each inference.

Training and Test Results

The model was trained for 40 epochs, on the provided training set (processed into 23,618 distinct examples).

The training was performed with an Adam optimizer, with a learning rate of 10^{-3}. Here is a plot of the training loss:



Evaluating the trained model with greedy search, on the test data, gives a BLEU@1 score of: **0.6687365272322527**

Using beam search (beam width = 2), we get a BLEU@1 score of: **0.6996315140815459**