CPSC 8430 Homework 2

Model (Decoder Only):

Data Preprocessing:

Loaded training label data and created a frequency dictionary out of the words found in the captions. Then the preprocessor **limits captions lenght** to a set max and drops words after that max. After the words are dropped, the preprocessor **filters out** any word that appeared **less than 10 times**. The captions were then converted to their index counterparts and finally into a “one hot encoding.”

Attention Layer:

The attention layer is simple. It takes encoder output from the dataset that has been run through an LSTM and does batch matrix multiplication with hidden state output from an GRU that was fed the encoder outputs. The goal here is to try and highlight a certain part of the frame through use of weights applied to the encoded feature vector from the CNN. This is done by having the LSTM learn were to look for any given sequence of frames.

Forward Layer:

The forward layer starts off by feeding in the CNN encoder output provided by the dataset into a **linear layer** that reduces the features. The linear layer output is then fed into an **SELU** and directly through a **dropout layer**. From the dropout layer the processed encoder frames are input into the **LSTM layer**. The hidden state outputs are collected from the LSTM layer and then pass to the **GRU layer** where the hidden state outputs are also collected. The forward layer then enters a caption prediction loop.

If the layer just started the loop a <BOS> is set as the previous input. Otherwise, if the loop has already been running, the forward layer decides whether to use **teacher forcing** based on a probability (random number lower than set number). If true, the ground truth will be given to the model for that word in the caption. This prevents a series of wrong guesses from over penalizing the model’s loss/weights. If teacher forcing is not used, however, the current word in the caption along with the prior predicted words are set as the input.

Before the current frame can be set as the decoder input, the hidden state outputs from the LSTM and GRU layers are sent to the **attention layer** to produce a set of “highlighting” weights which are applied to the current decoder input frame. With the applied weights the caption (predicted) and the current frame are combine and fed through the GRU previously used. Another **linear layer** and a **softmax** are then used to pick the best word choice for the caption so far. This is repeated until the maximum caption length is reached. Note that the caption can be filled with padding, so the actual sentence length does not need to be the maximum length.

Finally, the loss of the forward layer is calculated based on the difference between the ground true and the predicted word. Loss is only calculated once for each word in the

caption during any given loop. This is to prevent over penalizing the model for mistakes it cannot correct.

Testing and Validating Layers:

The testing and validating layers are very similar to the forward layer with a few functional differences. The testing layer is designed to use the **beam search algorithm** and produce an actual human readable sentence.

The validating layer is an exact copy of the Forward layer. It was made to be used to check epoch loss without updating weights of the model.

Training and General Setup:

The training loop for the model used both the forward layer and the validating layer. To setup the epoch loop the model was first initialized and then loaded onto the GPU (for this assignment a RTX 2060). For an optimizer I chose to go with the **AdamW optimizer** with weight decay as it was suggested online for general speedup and performance improvements.

Within the epoch loop a generic learning setup is used where the gradients are zeroed, the model is fed all data in the training dataset, loss is added up for a status update after each epoch, and the loss of each datapoint is backpropagated to the model. Finally, at the end of each epoch a total loss for the epoch and a **validation loss** (called “betterLoss” in the code) is reported. Based on the validation loss the model is either saved or the code continues until a new best loss is calculated.

For actual training I experimented with a few different hyperparameters including a **hidden node multiplier** which adjusted the input and output feature sizes. I ended up settling on 512 as anything under did not perform as well and anything over was too costly to training time.

For **batch size** I stuck with 64 as it fit well into the training data, only omitting a few samples per epoch.

Epochs were quite troublesome because 40-50 epochs produced consistently good results but anything past that i.e., 100-350 produced inconsistent results. The best model I created happened to be 200 epochs, but I might have gotten lucky.

Learning rate was set to 0.001 and consistently worked for the model, I have little to no problems with the gradient overshooting.

Teacher forcing rate was set to 0.06 and did not seem to change the outcome by much went increased or decreased slightly.

Beam path number was decided through empirical experimentation with my best model. I concluded that 4-5 paths gave the best possible bleu score.

CPSC 8430 Homework 2

Interesting Techniques:

The three main interesting techniques used in this homework are the beam search algorithm, the attention layer, and teacher forcing.

Beam search is simply keeping a record of n possible paths given n best choices at each word in a caption. This allows a model to possibly increase performance if a predicted word was not the best choice and prevents the whole caption from being ruined. The key point is that the combination of best words does not always provide the best caption.

Along with beam search, the attention layer was suggested in the homework writeup. Attention helps a model by highlighting important parts of a frame at every step in a caption.

Lastly was teacher forcing, as explained in the forward layer section, it helps the model by stepping in and asserting a correct word in a caption sequence to prevent long chains of incorrect guesses.

Performance:

The model performance peeked at about 0.69-0.697 with the bleu scoring metric. Training time took about 80-100 minutes and the test took under 3 minutes.

Model Requirements:

To run the trained model on a dataset four files are needed. Two of the files come included in the GitHub repository (testModel.py and hw2_seq2seq.sh). The other two files (modelBest and DatasetPreload) will be downloaded by the shell script hw2_seq2seq.sh when it is run.

The syntax for running the shell script is **./ hw2_seq2seq.sh dataDir results.txt**

dataDir must be the path to the test root folder of the dataset and nothing more. It also is expected to have the same file structure as MLDS including a named label file called “training_label.json”.

results.txt can be any name but the .txt file extension is not added by default.

This program was tested on a Palmetto Cluster node but should work on any Linux system that is correctly setup. The program also requires that a compatible GPU be present (one with at least 6GB of memory and compute capability 6.0 or higher). For my test I used a Tesla V100.

Palmetto Node: qsub -I -l select=1:ncpus=8:mpiprocs=8:ngpus=1:gpu_model=v100:mem=62gb:interconnect=hdr, walltime=4:00:00

GitHub Repository: https://github.com/beamreaching1/Deep-Learning