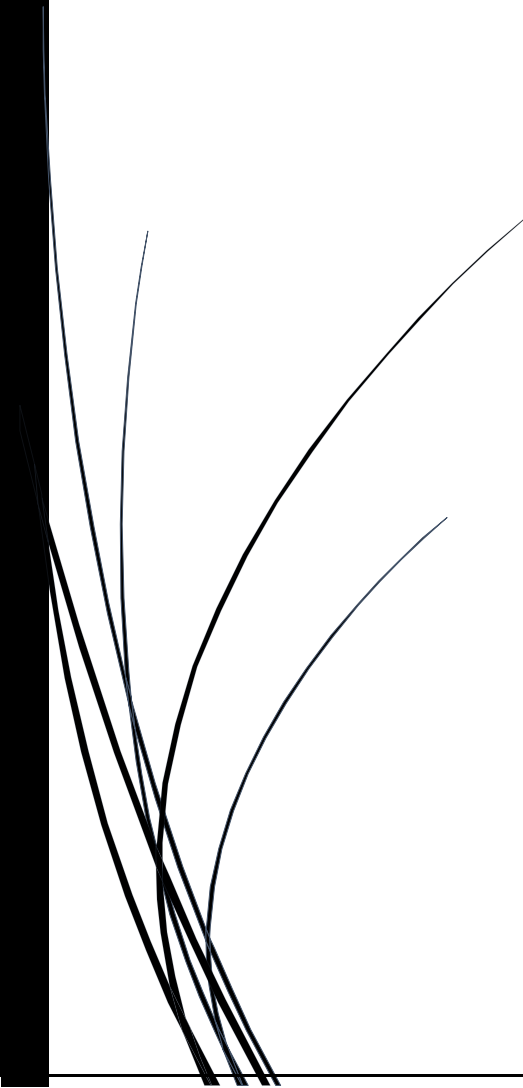


4/3/2020

HOMEWORK 2

Submitted by Foram Joshi



CPSC 8810: Deep Learning
CLEMSON UNIVERSITY

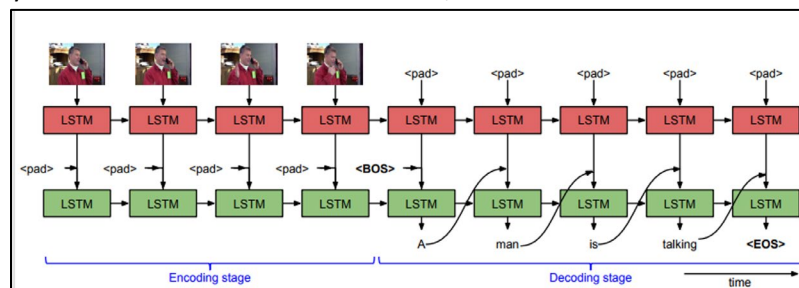
Things Done & learned

Loading and Preprocessing the Data

- 1) Loading & Preprocessing the data
 - `def getFeatures(filename)`: Gives you the features from JSON file
 - `def getLabels(index)`: Gives you the Label to corresponding feature given the index of it from JSON file
 - `def getIndicesFromSentence(sentence)`: Gives you an array that converts the words of sentences to their indexes
 - `def getSentenceFromIndices (index)`: Gives you an array of index that converts to words of sentence
 - `def sample_minibatch()`: This function makes mini-batches default to batch size
- 2) Creating 2 dictionaries, `word_to_count_dict`: maps all words in training labels into an index
`word_to_index_dict`: maps all indexes to words
Added BOS,EOS,<pad>,<unk> in these dictionaries.
- 3) We cannot feed the input data straight to the model. We need to split the sentences into words and then representing those words as number and eventually using one-hot vector. The words are uniquely indexed in the dictionaries.
- 4) The cleaning of the input is done by removing spaces and special characters. We then normalize the sentence to lowercase.

Network Architecture:

- 1) Creating the graph
 - Short video(Pre-processed video frames) : Input X- Placeholder
 - Caption that depicts the video: Output Y- Placeholder
 - Padding the input: `shape= [batch_size, hidden_units]`
 - Word embedding: `shape= [vocab_size,embedding_size(which is number of hidden_units)]`
 - Creating 2-layer LSTM cells with 128 hidden units,



- 2) Best Test Bleu Score: ≈ 0.76

Model used	
Training Epochs	15000
LSTM Dimension	128
Batch size	128
Learning Rate	0.001
Optimizer used	AdamOptimizer
Vocab Size	Min count > 3

The whole model is a 2-layer LSTM structure, these layers are internally sub-divided as encoding and decoding stage.

Encoding Stage:

1) Input of LSTM1 is pre-processed video frames of size [Batch Size, 4096] per timestep for 80 frames.

2) Output of LSTM1 is sent as the input to LSTM2 combined with embedding vector of the special word <pad>.

Decoding stage

3) Input of LSTM1 is a zero-tensor of size (batch_size, 4096) per time step.

4) Input to LSTM2 is a combination/concateration of the output of LSTM1 at that time step, and output of LSTM2 at previous time step.

5) The output of LSTM2 at every timestep is a tensor of size (batch_size, hidden_units). This output is then multiplied with the transpose of the word embedding matrix to determine the logit, of size (batch_size, vocab_size).

6) The argmax along the columns of the logit is appended into a tensor variable to determine the final output sentence. At the end of the sequence generation, this will be a tensor of size (batch_size, sequence_length)

7) The word embedding of output word is looked-up in the embedding matrix, this is a lower-dimension representation. This will be passed into the LSTM2 input at the next timestep.

7) The loss per time step is calculated by using the Softmax Cross-Entropy loss of the logits with respect to the labels at that timestep. All these losses are summed up over the output sequence length to derive the total loss per batch.

8) Adam optimizer is used to minimize the above loss

Training the model:

The model trains for 5000 epochs (it generally converges to a good validation score within the first 500 iterations). After every 2 iterations, the current bleu test score is generated. The best bleu score generated yet on the test dataset is logged, and whenever a better score is generated, the model parameters are saved. This is a greedy way of achieving the highest possible BLEU score on the test dataset (may not be the best approach, but worked for me.)

Testing the model:

The evaluation process begins by checking the model output and the BLEU score. Each pair of sentences will be feed into the model and predicted words will be generated. We will then take the argmax of the values and find the correct index to the value.