Assignment-4

Natural Language Processing and its Application

Report

Neural Machine Translation

Sourav Kumar
20161072

DATASET

For this, I have used my own DATASET because,

1. The dataset given contain around 50,000 parallel sentences (English-HIndi), so it is difficult to train on local system. I have tried to run code on GPU cluster of college but it is showing memory limit exceeded and every time i have to reinstall dependencies on my account(allocated space).
2. Sentences given are very large so neural network may not perform better for that case, so i took the dataset in such a way that first it has one letter translation then two so on..
3. I used Google Collab for the training purpose and all python libraries are preinstalled and it provide jupyter notebook access but if i take the whole corpus then notebook was crashing for some reason, so i took around 3000 parallel sentences to train the model.
4. Although time taken for the generation of one epoch is around 30 min, so it was not possible to train for large number of epochs (trained till 10).
5. Hence, the output by the models was not so good (mostly same kind of words are being generated in translation irrespective of the given input sentence). Therefore in order to get the better i output i trained the model using OpenNMT (till 5000 epochs).
6. Also, to overcome the problem of non-occuring word in corpus of given input text for translation, I am using Glove Vectors to generate dictionary.

Model 1: Sequence to Sequence Learning with Neural Networks

This is simple encode decoder based MT tool. To get better output:

1. I have reversed the input string as per suggested by the paper.
2. I have used the keras library for implementing Deep Learning.
3. I have used 2 layer LSTM mode for encoding and decoding purpose.
4. I have used Glove Vectors to generate dictionary of vocabulary of words in that language.
5. I have tried to optimize the parameters using Adam optimizer.

Link to: Code
Link to: Output Annalyse

Model 2: Neural Machine Translation By Jointly Learning To Align And Translate

It is similar to sequence to sequence but in this along with translation to target language we are also trying to generate the alignment by using bidirectional LSTM's (but here i am using GRU instead of LSTM) to get familiar with GRU's functionality. Also this time i used tensorflow library as Keras and tensorflow are two main libraries for Deep learning, so to get familiar with both i implemented both models in different backend for deep learning.

1. Here, I am using Bahdanau attention.

Pseudo code for my implementation is:

Notation:

- EO = Encoder output
- H = hidden state
- X = input to the decoder
- FC = Fully connected (dense) layer

Code:

- score = FC(tanh(FC(EO) + FC(H)))
- attention weights = softmax(score, axis = 1). Softmax by default is applied on the last axis but here we want to apply it on the *1st axis*, since the shape of score is *(batch_size, max_length, 1)*. Max_length is the length of our input. Since we are trying to assign a weight to each input, softmax should be applied on that axis.
- context vector = sum(attention weights * EO, axis = 1). Same reason as above for choosing axis as 1.
- embedding output = The input to the decoder X is passed through an embedding layer.
- merged vector = concat(embedding output, context vector)
- This merged vector is then given to the GRU

NOTE: The shapes of all the vectors at each step have been specified in the comments in the code.

Training

1. Pass the *input* through the *encoder* which return *encoder output* and the *encoder hidden state*.
2. The encoder output, encoder hidden state and the decoder input (which is the *start token*) is passed to the decoder.
3. The decoder returns the *predictions* and the *decoder hidden state*.
4. The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
5. Use *teacher forcing* to decide the next input to the decoder.
6. *Teacher forcing* is the technique where the *target word* is passed as the *next input* to the decoder.
7. The final step is to calculate the gradients and apply it to the optimizer and backpropagate.

Translation Criteria

- The input to the decoder at each time step is its previous predictions along with the hidden state and the encoder output.
- Stop predicting when the model predicts the *end token*.
- And store the *attention weights for every time step*.

Note: The encoder output is calculated only once for one input.

Link to: Code

Link to: Output Annalyse

NOTE: I have saved both the models in my google drive