Natural Language Processing Assignment 3

Ahmad Chaar, Abhishek Singh, Rebecca Erbanni

April 2019

1 Introduction

The topic of the third assignment is that of a dialogue system: in particular we still use data from the ConvAI challenge to learn to retrieve the next best utterance.
The dataset that will be used is $Persona - Chat$, which is a sum of conversations between crowdworkers that have been randomly paired and asked to pretend to be a certain given persona.
Overall it consists of 164,356 utterances in over 10,981 dialogues, for a total of 1155 possible personas (at training time) with at least 4 profile sentences.
For what concerns the test set, further unseen data was collected for automatic evaluation. The hidden test set consisted of 100 new personas and over 1,015 dialogues.
In the training set each utterance is followed by the correct answer plus N+1 distractors, while of course the test set lacks of the correct answer.
The given task is challenging because it requires both asking and answering questions and maintaining a consistent persona (in fact, common chit-chat models reveal the problem of inconsistent personalities).

2 Data Pre-Processing

The first part of the work consists in normalizing the given text, with the usual transformations which include:

- Converting upper case to lower case
- Separating composed words into separated words
- Transforming english contractions into full words
- Tokenization based on white space

After this, we extract from the train set text lists which would include the following:

- Line Number
- Your Persona
- Other Persona
- Utterance
- Correct Repsonse
- Five variables being a random from among the negative options available

Other pre-processing steps include trimming rare words. Under this approach, we remove words from our h that have total count in the corpus of below a threshold and repopulate the sentences excluding the trimmed words. We experimented with a variety of word count thresholds and settled on three as the most optimal.

3 Retrieval Based Model

3.1 The Model Architecture

The model we're using next is a dual encoder, whose main component is an encoder composed of the following three layers:

- An embedding layer of dimension vocabulary-size x embedding dimension which is initialized using Word2Vec model from gensim and training the model on brown corpus from NLTK
- A LSTM single layer with input-to-hidden weights initialized from a uniform distribution and hidden-to-hidden weights with orthogonal initialization.
- A dropout layer which is applied to the last hidden state of each input sequence.

All the word vectors of the input utterance are fed into the LSTM where the last hidden layer outputs a numerical representation of the input utterance.
This response is labeled and the objective is to minimize the loss given by the Binary Cross Entropy between a score (obtained from the response and the context) and the label.

3.2 Batch Processing

Owing to the huge size of the training data and limited computing power, we struggled to train the model on the entire data. We used a few approaches to begin with. We started using a for loop and passing each context, response pair along with a label and train the model with each context, response pair. This did not seem feasible as one epoch took a long amount of time even with just 50 dialogues. Next we tried to build a dataframe which stores all the context, response and label and pass the context, response and labels from this dataframe to train the model in batches and train it in parallel. Though the training happened a lot faster after the dataframe is created it took a long time to create the dataframe for all the dialogues.

Finally we stuck with picking randomly batches of context, response and label from training data and training for a large number of epochs. This way the computational time is very fast and since batches are picked randomly it helps prevent any bias.

3.3 Building Context Response Pairs

While experimenting with the structure of context response pairs, we had many ideas as to how to feed in the persona of the characters in the conversation.We tried first giving only the persona of the person who is going to reply. We thought this can be useful as the model can pick up on what to reply based on the personality. Later we tried forming the context by adding both personalities and the utterance. This approach rendered in lower loss overall. Hence, we stuck with the latter approach.

3.4 Experimentation with Hyper-parameter

Learning Rate- We experimented with multiple learning rates so that the loss converges and not on local minima. We tried 0.0001, 0.001 and 0.0005. We saw that 0.0001 led to the convergene of loss

Embedding- We tried with embedding of 50 and 100. It did not create a huge impact but size 100 performed a tad bit better

Layer- Layer size was important as to how the LSTM model performed. We tried with 50 and 100 and 100 performed beter than the former

4 Generative Model

After generating the sentences from the pre-processing stage, under the generative model we transformed sentences into tensors. However, we also transpose the batches in order to take advatage of the parallel processing and conserve the sequence through time steps. We also setup functions that handle the padding of sentences in order to have all the batches of a given time step to be of equal length.

4.1 Sequence to Sequence Model

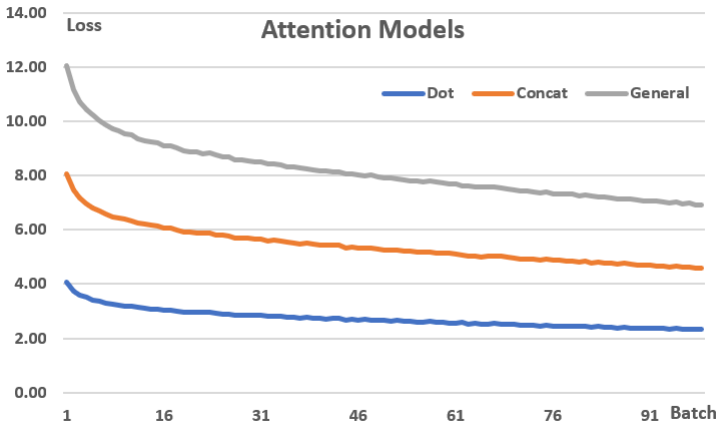
The model consists mainly of a sequence to sequence GRU having bidirectionnal connections:

- An encoder takes in a sequence (being the digitized batch) and outputs a word embedding representation of the word through a feature space. In the end, we get the embedding containing semantic similarity between similar meaning words.
- The decoder generates a response word per word until the EOS token is reached.
- A common problem with this architecture is the information loss. Therefore we used an attention mechanism that allows the decoder to keep in memory certain items of the sequence.

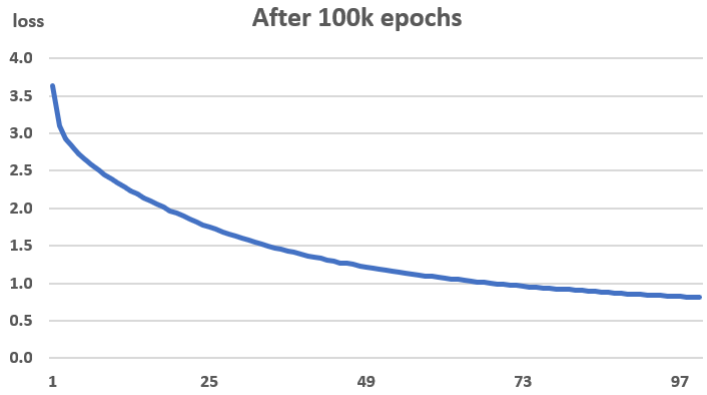
4.2 Experiments:

- In our experiments we used several variations of the attention mechanism, using the "Dot","General" and "Concat". In the end we settled on the concat method which produced the fastest convergence.

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$



- We experimented with several variations of the teacher force ratio. This parameter consists of ratio between the encoder and decoder learning rate. Finally we decided on a teacher forcing ratio of 1 meaning, the encoder and decoder would both have equal learning rates.
- We experimented with several variations of the gradient clipping, but quickly found out that this did not have a significant impact on the training performance. Gradient clipping was used to prevent exploding gradient problem. However, the architecture of the GRU avoids this issue by design (unlike RNN).
- We also experimented with several learning rates for the encoder and decoder. We noticed that a learning rate of 0.0001 resulted in the most consistent convergence in the loss function.
- Finally, in our last experimentation, we used google colab to test the final model on 100,000 epochs and we noticed that the loss continued to converge gradually and the generated sentences improved in quality. We used the best hyper-paramaters configuration obtained with the previous experiments.



Last remarks: Please note we set the iterations of the generation model to be 100,000 and the iterations for the retrieval model at 200,000. These should take around a day to complete on a CUDA device. However, the following variables can be changed to accommodate for other hardware: "nb _epochs" and "n_iteration".

Also note that although the generation model works well, we did not manage to achieve a better than base-line performance on the Retrieval Based Method despite the hyper-parameter tuning.

References

[1] Matthew Inkawhich. CHATBOT TUTORIAL. *Pytorch*, https://pytorch.org/tutorials/beginner/chatbot_tutorial.html

[2] Janina Nuber. https://github.com/Janinanu/Retrieval-based_Chatbot

[3] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao and Dan Jurafsky. Deep Reinforcement Learning for Dialogue Generation *NIPIS*, 2016.