Recurrent Entity Networks for Persona-based Dialogue Generation

Mitchell Kaysen
University of Texas at Austin, Department of Computer Science
mkaysen@cs.utexas.edu

Abstract

Chit chat dialogue based on a persona requires the selection of relevant information from the encoded persona statements. This entails the usage of some form of memory network in order to access the encoded persona. In this paper, I used a modified Recurrent Entity Network, but any memory network would probably do. I present a basic seq2seq model augmented with this memory network and evaluate its performance on the PersonaChat dataset. I also evaluate several other models for comparison using BLEU score and perplexity. I find that the architecture provides some improvement in the perplexity, but can be outperformed by other architectures. I also find that adding attention in general hurt the BLEU score of the model.

1 Introduction

The PersonaChat dataset (Zhang et al., 2018) is a chit-chat dataset for dialogue generation. The dataset consists of a conversation of multiple dialogue turns, as well as associated personas for each of the participants. Each persona consists of a series of statements about that participant from the first person perspective. Each dialogue turn has both the actual response, as well as other candidate dialogue responses. The conversations were produced by having participants converse while acting out their given persona. The goal of these personas, and the dataset in general, is to thus provide a more relevant and useful context on which models can base predictions (Zhang et al., 2018).

Zhang et al. (2018) present two main tasks for the dataset: generative models and ranking models. The generative models are language models, where the objective is to predict the next token based on the given context. The objective of the ranking models is to rank the suitability of various predetermined responses. This is why each dialogue turn contains a list of potential candidate responses in addition to the actual response. For the purposes of this project, I only evaluate generative models, not the ranking models, and I do not use any of the additional candidate responses.

The generative model task can be represented and trained as a sequence to sequence (Seq2Seq) problem (Cho et al., 2014). The persona and the partner's utterance together are the given source for each training example. For the baseline Seq2Seq, the source sentence is encoded into a fixed length context vector, which is then passed to the decoder (Cho et al., 2014). However, Luong et al. (2015) showed that directly consulting the past memory states using an attention mechanism can significantly improve performance on machine translation.

Memory augmented neural networks such as the Neural Turing Machine (Graves et al., 2014) use an attentional mechanism controlled by a recurrent controller to access an external memory. The Recurrent Entity Network (EntNet) is a memory network first developed by Henaff et al. (2016) for question-answering tasks. In general, it can be viewed as a bank of recurrent networks with an associated key for each network. However, the architecture as specified in the original paper is only built for single inputs, not sequences (Henaff et al., 2016), so it required modification before it could be used feasibly for this project.

2 Recurrent Entity Network

2.1 Original formulation

Originally, Henaff et al. (2016) used a simple encoding scheme where the embeddings of the original sequence are multiplied by a learned positional mask. This result is then summed to produce the input to the EntNet itself.

To update each of the key-value pairs, the model first takes the dot product similarity of the input with the key and value, which is then passed through a sigmoid to obtain a gating value. The key, value, and input are all fed through respective feedforward layers as well as an activation function to produce an update term. The new value associated with the key is obtained by adding the product of the gating value and the update term to the previous value for that key. A normalization is applied on the final values.

2.2 Modifications

Because of the limitations of the original design in handling longer sequences, I replaced the update scheme with an attention layer. The EntNet consists of a bank of key-value pairs, each of which is meant to represent separate entities (Henaff et al., 2016). I add the keys and values of the EntNet together to produce a query matrix Q. I then perform attention using Q on a matrix M of memories to encode, and obtain a context matrix C. I produce the new values for the EntNet's recurrent states by adding the previous values as a residual connection to the context matrix C and performing a layer normalization (Lei Ba et al., 2016). This is essentially the global attention mechanism (Luong et al., 2015), where the queries are the current memory values of the EntNet. This attention mechanism allows for the simple encoder (Henaff et al., 2016) to be replaced with the encoder of a Seq2Seq model, with the matrix M of memories to encode being the hidden states of the encoder over the source sequence.

The output module of the original EntNet needed next to no modification, as it is also equivalent to an attention mechanism. The only alteration I made was to use scaled dot-product attention (Vaswani et al., 2017) rather than just the dot-product.

3 Seq2Seq Modules

I tried several different model architectures to contrast with the modified EntNet architecture. To observe the differences in performance between models based on additional functionality, I adopted a modular approach.

3.1 Encoder/Decoder

The core of every model was a basic encoder-decoder architecture using a GRU (Cho et al., 2014). I adopted this approach, rather than a non-recurrent approach such as that in the Transformer (Vaswani et al., 2017), for two reasons. The first was that I no longer had to encode the position in any way outside of the GRU itself, as Vaswani et al. (2017) did. The second is that I would no longer need to mask the input in any way to avoid the model looking into the future (Vaswani et al., 2017).

For the non-EntNet models that incorporated a persona, the persona was simply concatenated to the front of the utterance before being passed into the encoder, as detailed by Henaff et al. (2016). However, for the EntNet, I used the decoder GRU to encode the persona embeddings before inputting them into the EntNet itself. The intuition behind this was to hopefully learn a shared representation between the model's generated utterances and the persona sequence, the persona being statements about the self in the first person perspective. I did not perform any concatenation between the persona and the source sentence for the EntNet.

For all models, I used an embedding layer whose weights were loaded in from GLoVE.

3.2 Output Layer

For the output to every model, I initially tried to use a simple feedforward layer from the hidden dimension to the number of classes, over which I applied log-softmax to obtain the final log-probabilities. However, the size of any such layer is substantial for large vocab sizes, and presented a significant challenge because of the limitations on both batch size and speed.

I was able to rectify this problem by replacing the feed forward layer and log softmax with an Adaptive Log Softmax layer (Grave et al., 2016). This module was already implemented in pytorch, so I did not have to make my own implementation. After switching to this module, I observed significant improvements in both compute time as well as required space, and was able to run far higher batch sizes. The adaptive log softmax splits the traditional linear layer into several linear layers of increasing size. Each produces the softmax for a subset of the distribution, allowing the layer to save on both time and space by only computing the necessary layer (Grave et al., 2016). This did require that the tokens be sorted by order of rank, but that was trivial. I also had to decide index cutoffs for the different subsets. I describe this more in detail in Subsection 4.1.

3.3 Attention Layers

For both the EntNet modifications and the models that used an attention layer, I implemented Scaled Dot-Product Attention as described by Vaswani et al. (2017). I used the same input to produce the keys and values from respective feedforward layers. I also used a feedforward layer to transform the queries. After each attention layer, I added the queries in as a residual connection then performed a layer normalization (Lei Ba et al., 2016), which is implemented in pytorch.

4 Training

4.1 Dataset Preprocessing

I adopted a word-level tokenization scheme for this dataset. Although this did allow me to use pre-trained GLoVE embeddings for the models' embedding layers, there were a couple problems with this approach that likely hurt performance. The primary issue is typos, as these are considered unique tokens, and both receive a random embedding weight while also artificially increasing the number of potential classes, which is very problematic when using a traditional linear and log-softmax output layer.

To use the Adaptive Softmax layer provided with pytorch, it is required that the classes be sorted in order of frequency, with the highest frequency tokens being assigned the lowest indices. This is trivial to implement. However, there is the issue of padding tokens, as the frequency of those is entirely batch dependent. This is because I only padded up to the maximum length sequence per batch to save on memory. Considering that the padding token would always fill up the rest of the sentences, leading to a very high incidence even if the length to be padded were to be small, I decided to simply make the padding token the first index. My assumption is that it will always be at least be top 3 or 4 even with a very small batch size. I believe that the larger the batch size gets, the frequency of the padding token will probably become higher than any other.

For the cutoffs of the Adaptive Log Softmax layer, that is, the subset of indices handled by each part of the layer, I looked at the distribution of the dataset overall. Because I used word-level tokenization, the vast majority of unique tokens in the corpus are either proper nouns or typos. Most tokens in the corpus only have a frequency of one or two. Thus, the earlier cutoffs were the most important. I decided to make the cutoffs 7 and 21 initially, then I also added a cutoff at index 500 for good measure. This was just a guess based off of looking at the frequencies. I added the cutoff at 500 to save further on computational time and memory by trying to reduce the number of tokens processed by the largest sublayer, the tail. Though this was a rough estimate, it worked extremely well, and allowed me to run substantially larger batches and significantly cut down on training time. I did not feel it needed significant hyperparameter tuning, so I left it as is.

4.2 Optimizer and Learning Rate Scheduling

I used an AdamW optimizer (Loshchilov and Hutter, 2017) and I used the Cosine Annealing With Warm Restarts (Loshchilov and Hutter, 2016) scheduler, both provided by Pytorch. I used AdamW (with $lr = 1e - 3$) because the models were prone to overfitting, which AdamW helps to counteract (Loshchilov and Hutter, 2017). Some sort of scheduler also proved necessary to help the model converge. I tried multiple different schedulers, including StepLR and ReduceLROnPlateau, but this particular scheduler gave the best results, and in less epochs.

4.3 Models Tested

I evaluated five different models:

1. Seq2Seq baseline, no persona
2. S2S baseline, with persona
3. S2S with persona and attention
4. S2S with EntNet
5. S2S with EN and auxiliary loss

These different models all used the encoder-decoder as the core of the architecture, as well as GLoVE for the embedding layer. They were all trained independently, with no transfer learning scheme. For the baseline Seq2Seq, I simply did not concatenate the persona to the source utterance.

I also applied a self attention layer (Vaswani et al., 2017) to the encoder's hidden states in the attention model and EntNet models, as well as performing a query on the EntNet's recurrent state using the encoder hidden states. The results of any

Model	**PPL**	**BLEU**
S2S no persona	190	2.39
S2S persona	213	2.62
S2S persona + attn	154	1.59
EntNet	172	1.14
EntNet + Aux	170	1.22

Table 1: Results for each of the models. All models used GLoVE with $d = 100$ and a hidden layer with $d = 200$. The EntNet used 20 blocks.

attention layers, both to access the EntNet and to access the source sentence encodings, were added with the query before applying a layer normalization (Lei Ba et al., 2016).

The final EntNet I trained involved adding an auxiliary decoder that would attempt to reconstruct the persona sentence from the EntNet recurrent state. I used the same output layer as in the rest of the model to produce the loss.

5 Results and Analysis

5.1 Perplexity

Somewhat expectedly, both the Seq2Seq with attention and the EntNet based model were able to achieve much better perplexity than the baseline seq2seq models.

5.2 BLEU

The BLEU scores for the models are rather confusing. The attention based model and EntNet models did much worse than the baseline models, which is not what I would expect given the perplexity is essentially the inverse. This is likely because the networks are much smaller than what are used to achieve state of the art, and are also not particularly deep. Multiple stacks of attention layers, as well as training for more iterations, would probably cause the attentional models to outperform the baseline, while the baseline stagnates at some point.

5.3 Qualitative Analysis

Again, it was surprising that the models that had the best BLEU scores had the worst perplexity. Qualitatively, looking at the output of the baseline models reveals extremely safe sentences composed of the most frequent tokens. The other models produced some words that were less common. This indicates that the baseline models are outputting more confident probabilities for the wrong tokens, leading to higher perplexity. The BLEU score can likely be explained by the grammatical and frequent nature of the output, making the baseline models more likely to achieve good scores.

In my opinion, this dataset requires a different automated metric for scoring the relevance of the generated utterance to the persona statements.

5.4 Overall Performance of the EntNet

It's possible that I overengineered the final model a bit. However, there is definitely a noticeable drop in perplexity when the EntNet is introduced, so it does improve performance compared to the baseline. One of the issues that probably hampered its performance in relation to the attention-only model is that the attention-only model has direct access to all of the encoder's hidden states, including those for the persona. The EntNet based model, in the meantime, can only access the persona's representation in the EntNet. Considering that the number of blocks I assigned to the EntNet for these experiments was only twenty, it's highly likely that there is information being lost when it compresses the persona sequence into memory. That, or there are simply not enough training examples to adequately determine which information is most relevant, but I do not think that this is the case.

This project mainly served to highlight the disadvantages of the EntNet when compared to simple attention and other memory augmented networks. The EntNet requires parameters for the keys and values. The EntNet also struggles to use extremely large memories because every recurrent state is checked or changed on update, requiring the values to be copied and saved for every update to properly pass a gradient. It also requires several additional parameters for the query and input modules, making it larger and slower. At the same time, it does not achieve the same performance that simple attention does.

5.5 Auxiliary Loss

The auxiliary loss that attempts to minimize the reconstruction error of the persona seemed to help, primarily in the BLEU score. It lead to a slight drop in perplexity as well, but I don't know if the difference is enough to be significant. Its addition seemed to allow the model to learn more grammatical output, which is one benefit of using the decoder to encode the persona. However, the addition of the auxiliary decoder module and loss slowed the model down significantly at test time, for some reason.

5.6 Possible Improvements

I think evaluation on the dataset would be improved if an additional metric could be developed to measure memory recall for a given persona. Such a metric should also allow for measurement of long term dependencies between sentences in other conversational datasets.

The next steps I would take if I were to continue working on this project would be to try transfer learning, possibly using QA tasks, to train the memory module. I would also adopt a different tokenization scheme, such as a character-level CNN or a sub-word tokenization. I believe that the word-level tokenization scheme probably hurt performance. The prevalence of typos and proper nouns inflated the vocabulary quite a bit, and makes a word-level scheme less appropriate for this dataset. I also would experiment with more hidden sizes and number of blocks for the EntNet.

References

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv e-prints*, page arXiv:1406.1078.

Edouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. 2016. Efficient softmax approximation for GPUs. *arXiv e-prints*, page arXiv:1609.04309.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural Turing Machines. *arXiv e-prints*, page arXiv:1410.5401.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. 2016. Tracking the World State with Recurrent Entity Networks. *arXiv e-prints*, page arXiv:1612.03969.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv e-prints*, page arXiv:1607.06450.

Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic Gradient Descent with Warm Restarts. *arXiv e-prints*, page arXiv:1608.03983.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled Weight Decay Regularization. *arXiv e-prints*, page arXiv:1711.05101.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation. *arXiv e-prints*, page arXiv:1508.04025.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *arXiv e-prints*, page arXiv:1706.03762.

Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. 2018. Personalizing Dialogue Agents: I have a dog, do you have pets too? *arXiv e-prints*, page arXiv:1801.07243.