CHATBOT WITH PERSONALITY : BUILDING SEQ2SEQ ARCHITECTURE USING NUMPY FROM SCRATCH

A PREPRINT

Hao-Chien, Hung
Institute of Information Science, Academia Sinica
Taipei, 10617, Taiwan
hchungdelta@gmail.com

March 7, 2019

ABSTRACT

The whole machine learning model in this paper is writed using Numpy, which is available on Github (see footnote). The basic LSTM seq2seq architecture is adopted in this model, the self-attention mechanism and the layer normalization are applied to enhance the performance and the training efficiency. Furthermore, parallel computing is implemented to speed up training. In this paper, the main purpose is to build a chatbot model that can reply based on the personalities. Instead of only using personality-specific data to train the model, the model is trained with general data with personality label. By this approach, the personality-labeled data can help to mold the personality, the personality-unlabeled data (general data) can be used to boost the linguistic proficiency of the model. Only 880k conversation data are used for training (80k of these are labeled), however, even with relatively small data, the results are reasonably good and indicate the promising future for further developing. [1]

***K*eywords** Natural Language Processing · Deep Learning · Attention Mechanism · Layer Normalization

1 Introduction

Recently, there is an unprecedented progress in natural language processing (NLP), which attributes to the rapidly developing neural network architecture and the computing power increases in the order of magnitude just within these few decades. The dominant learning model in NLP is the encoder-decoder neural network, which is known as seq2seq model, has a multifaceted capability to perform a variety of tasks, such as machine translation, speech recognition, and text generation.

Nowadays, most of the seq2seq models are using long-short-term memory(LSTM)[1] or gated recurrent unit (GRU)[2] as the basic building block, which is less likely to suffer from loss of information compared to the traditional recurrent neural network(RNN). Subsequently, the introduction of attention mechanisms [3, 4] further boost the performance of seq2seq model. More recently, a new attention mechanism called self-attention have been introduced by Ashish Vaswani *et al.* [5], in which a set of key-value pairs are adopted. In this paper, a different version of self-attention is implemented, which will be discussed later.

To train the LSTM seq2seq model is computationally expensive, and it potentially suffers from internal covariate shift as the neural network become deeper, which further slow down the training. Fortunately, layer normalization [6] has been introduced to efficiently solve this problem. Layer normalization normalizes the activities of the neurons and hence mitigates the risk of vanishing gradient problem during training. Compare to batch normalization [7], layer normalization doesn't require a large amount of batch during training and doesn't need to deal with different input

[1] Code is available on Github https://github.com/hchungdelta/Simple_NN_API/tree/master/NN_v.2.1_personality_model

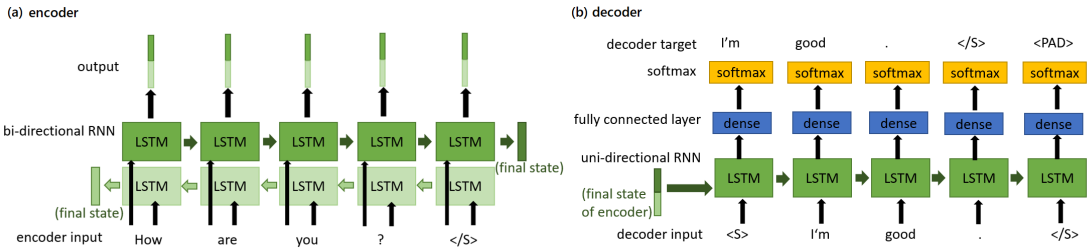

Figure 1: The schematic diagrams of seq2seq model. (a) The encoder, where the Bi-RNN mechanism is used. (b) The decoder, where the uni-RNN mechanism is used. <S>, </S>, and <PAD> represent the start token, end token, and padding token, respectively.

sequence length. Therefore, layer normalization is generally considered more suitable to be combined with LSTM-based model rather than batch normalization.

Recently, although most of the state-of-the-art modern chatbot models can reply to the query with high accuracy, there are still many challenges remain. One of the challenges is lacking personality and identity. It can potentially lead to a discontinuous position for the model, such as different social status and different gender identity. The most simple approach to construct a personality for a model is to extract a certain amount of data that share a similar personality, however, it must at the expense of fewer data and suffer from less accuracy. In this paper, the main purpose is to introduce an efficient approach to train the conversation model that has personalities and in the meanwhile maintain the same level of linguistic proficiency. For the sake of simplicity, only two kinds of personalities are embedded (general and sister), while this model can be applied to train multi-personalities if labeled data are available. The paper is organized as follows. Section 2 presents the architecture and the basic algorithm of this model. In section 3, the results and discussions are presented. Summary and conclusion are included in section 4.

2 Neural Network Architecture

2.1 LSTM

In this paper, the LSTM mechanism [1] is adopted as the basic building blocks of my seq2seq model. The LSTM mechanism is shown below:

$$forget\ gate : hf_t = \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f) \tag{1}$$

$$input\ gate : hi_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i) \tag{2}$$

$$output\ gate : ho_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o) \tag{3}$$

$$candidate\ output : hc_t = tanh(W_{oc}h_{t-1} + W_{cx}x_t + b_c) \tag{4}$$

$$hidden\ state : c_t = hf_t * c_{t-1} + hi_t * hc_t \tag{5}$$

$$hidden\ output\ h = ho_t * tanh(c_t) \tag{6}$$

A number of LSTM form the neural network sequence. In seq2seq model, two sequences are needed, which are known as the encoder and the decoder.

2.2 Seq2Seq Model

For the encoder, the Bi-directional RNN [8] is implemented to process the inputs. For the decoder, since future inputs aren't available, uni-directional RNN is used. A simple schematic diagram is depicted in Fig. 1.

2.3 Layer Normalization

Layer normalization [6] is implemented to improve the efficiency of training. In the early stage, batch normalization [7] is also adopted to train the model. However, the results indicate that it potentially makes the model less stable, since the lengths of input data are usually different, which make batch normalization less suitable for RNN. Therefore, only layer normalization is used in this paper. The mechanism of layer normalization is shown in the following:

$$mean : \mu_d = \frac{1}{d}\sum_{i=0}^{d} x_i \tag{7}$$

$$variance : \sigma_d^2 = \frac{1}{d}\sum_{i=0}^{d} (x_i - \mu_d)^2 \tag{8}$$

$$normalization : \hat{x} = \frac{x - \mu_d}{\sqrt{\sigma_d^2 + \epsilon}} \tag{9}$$

$$scale/shift : LN(x) = \gamma\hat{x} + \beta \tag{10}$$

d represents the depth of the input. i in x_i represents the $i-th$ scalar value in this input. γ and β are both vectors to re-scale the input. As the weights and the biases in LSTM, the parameters of these two vectors are shared among the RNN sequence. The backpropagation is similar to those used in batch normalization [7]. As the following expresses:

$$\frac{\partial L}{\partial \gamma} = \sum_{i=0}^{d} \frac{\partial L}{\partial LN(x)} \cdot \hat{x} \tag{11}$$

$$\frac{\partial L}{\partial \beta} = \sum_{i=0}^{d} \frac{\partial L}{\partial LN(x)} \tag{12}$$

$$\frac{\partial L}{\partial \hat{x}} = \frac{\partial L}{\partial LN(x)} \cdot \gamma \tag{13}$$

$$\frac{\partial L}{\partial \sigma_d^2} = \sum_{i=0}^{d} \frac{\partial L}{\partial \hat{x}} \cdot (x - \mu_d) \cdot \frac{-1}{2}(\sigma_d^2 + \epsilon)^{\frac{-3}{2}} \tag{14}$$

$$\frac{\partial L}{\partial \mu_d} = \sum_{i=0}^{d} \frac{\partial L}{\partial \hat{x}} \cdot \frac{-1}{\sqrt{\sigma_d^2 + \epsilon}} + \frac{\partial L}{\partial \sigma_d^2} \cdot \frac{\sum_{i=0}^{d} -2(x - \mu_d)}{d} \tag{15}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \hat{x}} \cdot \frac{1}{\sqrt{\sigma_d^2 + \epsilon}} + \frac{\partial L}{\partial \sigma_d^2} \cdot \frac{2(x - \mu_d)}{d} + \frac{\partial L}{\partial \mu_d} \cdot \frac{1}{d} \tag{16}$$

After applying layer normalization, the LSTM mechanism now should be rewritten as:

$$hf_t = \sigma(LN_{fh}(W_{fh}h_{t-1}) + LN_{fx}(W_{fx}x_t) + b_f) \tag{17}$$

$$hi_t = \sigma(LN_{ih}(W_{ih}h_{t-1}) + LN_{ix}(W_{ix}x_t) + b_i) \tag{18}$$

$$ho_t = \sigma(LN_{oh}(W_{oh}h_{t-1}) + LN_{ox}(W_{ox}x_t) + b_o) \tag{19}$$

$$hc_t = tanh(LN_{ch}(W_{oc}h_{t-1}) + LN_{cx}(W_{cx}x_t) + b_c) \tag{20}$$

$$c_t = hf_t * c_{t-1} + hi_t * hc_t \tag{21}$$

$$h = ho_t * tanh(LN_o(c_t)) \tag{22}$$

The comparison of training efficiency of the model with and without layer normalization is displayed in Fig. 2.

2.4 Attention mechanism

Dot-product attention mechanism [5] is adopted in this paper, although a little change in algorithm is made. As illustrated in 3. The encoder output) and the decoder output are separated into key parts (k_i for encoder key, d_i for decoder key) and value parts (v_i for the encoder value, O_i for the decoder value). Afterward, the original decoder value and the attention output are concatenated as the final decoder output.

The algorithm is as follows:

$$score_{ij} = s_{ij} = d_i \cdot k_j \tag{23}$$

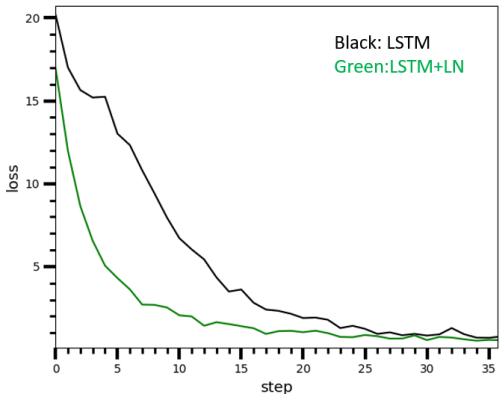

Figure 2: The comparison of the training efficiency of pure LSTM model and LSTM model with layer normalization.

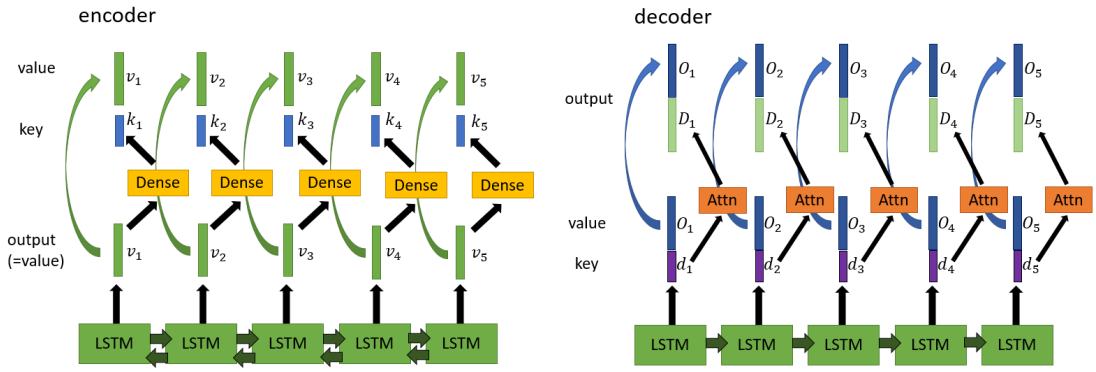

Figure 3: The schematic diagram of the attention mechanism implemented in this paper

$$alpha_{ij} = \alpha_{ij} = \frac{e^{s_{ij}}}{\sum_{x=0}^{x_{max}} e^{s_{ix}}} \tag{24}$$

$$attention\ output_i = D_i = \sum_{x=0}^{x_{max}} \alpha_{ix} v_x \tag{25}$$

$$output_i = concat(O_i, D_i) \tag{26}$$

For the backpropagation:

$$\frac{\partial L_i}{\partial v_j} = \frac{\partial L_i}{\partial D_i}\frac{\partial D_i}{\partial v_j} = \frac{\partial L_i}{\partial D_i}\alpha_{ij} \tag{27}$$

$$\frac{\partial L_i}{\partial d_i} = \frac{\partial L_i}{\partial D_i}\frac{\partial D_i}{\partial d_i} = \frac{\partial L_i}{\partial D_i}\sum_{x=0}^{x_{max}}\frac{\partial}{\partial d_i}\alpha_{ix}v_x = \frac{\partial L_i}{\partial D_i}[\sum_{x=0}^{x_{max}}\alpha_{ix}k_x v_x - \sum_{x=0}^{x_{max}}\sum_{y=0}^{x_{max}}\alpha_{ix}\alpha_{iy}k_y v_x] \tag{28}$$

$$\frac{\partial L_i}{\partial k_j} = \frac{\partial L_i}{\partial D_i}\frac{\partial D_i}{\partial k_j} = \frac{\partial L_i}{\partial D_i}\frac{\partial}{\partial k_j}\sum_{x=0}^{x_{max}}\alpha_{ix}k_x v_x = \frac{\partial L_i}{\partial D_i}[\delta_{ij}d_i v_j - \alpha_{ix}\alpha_{ij}d_i v_x] \tag{29}$$

In this model, since the encoder training data is much shorter than the decoder, only a single-headed attention mechanism is implemented, rather than multi-headed attention mechanism. Fig. 4 compares the efficiency of the model with and without attention mechanism.

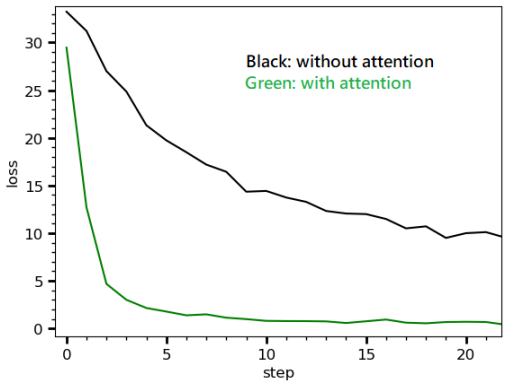

Figure 4: The comparison of the training efficiency of the model with and without attention mechanism.

2.5 Orthogonal initialization

Rather than randomly generate the weight matrices as initial states, in this paper, the weight matrices are generated using the concept called orthogonal initialization. It can potentially help the learning model recognize the difference between each input.

2.6 Parallel Computing

Parallel computing is implemented by mpi4py. In current development, only embarrassing parallel computing is available in this code, since it is effortless to apply to all the learning models without further adjustments. All the nodes share the same weights (W) and biases (b) throughout the training while exposing to different training data. For instance, if four nodes are used for training and the batch is set to be 32, training data no.1 to no.32 will be delivered to node one, no.33 to no.64 will be delivered to node two, and so on. Hence at each step, the total amount of 128 training data are used for training. Afterward, the master node will sum up all the gradients of weights (dW) and biases (db) to optimize the model. As the following equations show.

$$dW_{sum} = \sum_{node=1}^{node_{max}} dW_{node} \tag{30}$$

$$db_{sum} = \sum_{node=1}^{node_{max}} db_{node} \tag{31}$$

The summed gradients of weights and biases will be sent to the optimizer for updating the original weights and biases (See Sec. 2.7).

The schematic diagram (Fig. 5 (a)) visualizes the notion of the embarrassing parallel computing in this code. If one wants to take an average of the gradients to avoid overshooting, just simply divided the learning rate by the amount of the nodes. Fig. 5 (b) depicts the loss-step curves for the same learning model while using a different amount of nodes during training. The plot clearly indicates the parallel computing is useful and efficient. Until now, there is no dramatic deterioration in accuracy found when using parallel computing.

2.7 Optimizer

Stochastic Gradient Descent (SGD), Momentum, and Adaptive Moment Estimation (Adam) optimizer [9] are available in this code. In this paper, only Adam optimizer is used for training since the efficiency of Adam can outperform the other two approaches nearly in the order of magnitude. The learning rate (lr) is set to be 0.0015. ϵ, β_1, and β_2 are equal to $1e^{-8}$, 0.9, and 0.999, respectively. The following shows how the Adam optimizer works, t represents the step during training (here only take the gradients of the weights for example, the same goes for biases).

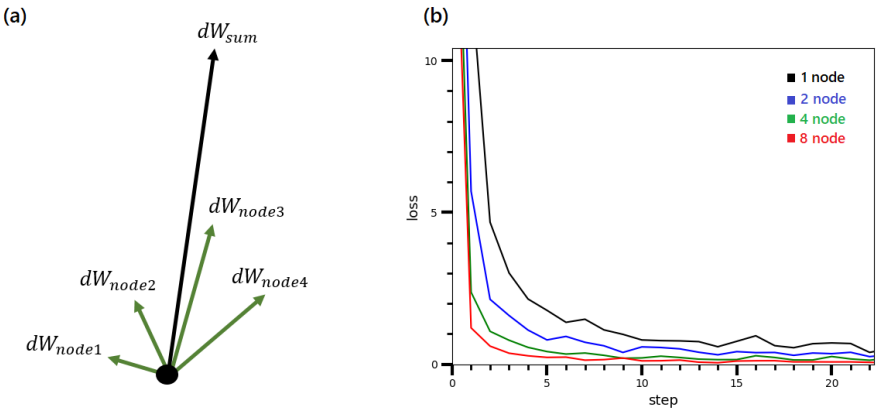

Figure 5: (a) The schematic diagram of embarrassing parallel computing implemented in this model. (b) The comparison of the loss-step curves of the same learning model trained with different amount of nodes.

$$dW_{t+1} = dW_t - lr\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{32}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}\ ,\ \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{33}$$

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)dW_t \tag{34}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)dW_t^2 \tag{35}$$

The gradient clipping [10] is also implemented to help model converge more smoothly. In this paper, the threshold is set to be 1.5. The equation is shown below:

$$dW_{ij} = \begin{cases} threshold, & \text{if } dW_{ij} > threshold \\ dW_{ij}, & \text{otherwise} \\ -threshold, & \text{if } dW_{ij} < -threshold \end{cases} \tag{36}$$

2.8 Word2vec

The word vector has been proved to be more efficient, and serve as a better representation for the words compared to the simple one-hot vector. First, statistic method is used to separate the sentences into words. Afterward, word2vec(using genism) is implemented to generate the word vector. To avoid including too many vocabularies in the dictionary, the threshold is set to be 50 (the lower limit of the number of occurrences), below which the vocabularies are labeled as "unknown". The depth of the word vector is 200 units. A few hundred MB news data was provided to train the word vector, as it turns out that nearly 41k vocabularies are over the threshold and be included in the dictionary.

2.9 Personality label

General and sister's personalities are labeled in this paper, personality vectors are embedded in the decoder. Since there only exist two types of personalities, general and sister's personality vectors are set to be a zero vector, and one vector, respectively. The depth of personality vector is 100, which is concatenated with decoder output and attention output, before entering the dense layer. As depicted in Fig. 6, P_{emb} is abbreviation for personality embedding. It is noteworthy that the personality embedding technically act as an additional bias outside the LSTM sequence.

2.10 Model

The neural network architecture used in this paper is illustrated in Fig. 6. The hidden units in LSTM are set to be 800 (for biRNN it is 1600 in total). The attention depth is 200 (depth of k_i and d_i). The input length and output length are both 20. The batch size is 32 during training. With only 1 CPU, it cost nearly a month to train the model. The results and discussions are detailed in the next section.

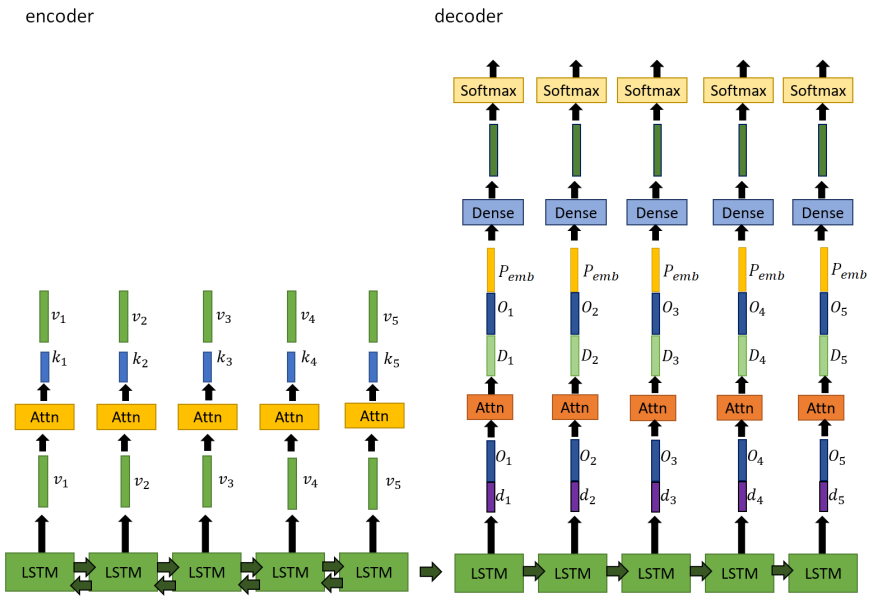

Figure 6: The neural network architecture used in this paper. P_{emb} is abbreviation for personality embedding.

3 Results and Discussions

Since there is no universal standard to judge the performance of this kind of task-specific model, The evaluation of the performance of this model must be based on human judgment. In this section, the performance of the two models is exhibited. The first one is proposed model in this paper, which is training with 800k general-labeled data and 80k sister-labeled data. The second one is a simple version, only training with 80k sister-labeled data.

Three criteria are considered to judge the performance of the model, the first is the grammar of the sentence, whether the sentence is grammarly correct or not. The second criterion is the consistency of the output and the input, to see whether the model can generate relative content based on the input. The third criterion is whether the model can reply a sentence based on its personality.

The same input sentences are used to examine the performance of these two models, the results of the first model are displayed in Fig. 7. The outputs based on general personality are on the right, those based on sister's personality are on the left. The results of the second model are shown in Fig. 8. One can find that the proposed model (first model) has higher accuracy since it has more training data. Using personality embedding, the sister's characteristic can be extracted from the model, a certain "stereotype" can be constructed. It is quite surprising that by using this approach, the personality-specific conversations can share the same linguistic proficiency as in general case and even enhance the personality trait. This model obviously outperforms the second model evaluated based on these three criteria.

As mentioned in Sec. 2.9, the personality embedding act as an additional bias outside the LSTM sequence, which would be a little bit counterintuitive, but the results indicate that the performance of the model in which personality embedding is concatenated within LSTM sequence are not ideal. To optimize this model, and also to have a full understanding of phenomenon, further investigation and development are required.

4 Conclusions

In this paper, a conversation model based on personality is introduced. Although the personality embedding in this paper just act as a additional bias outside the LSTM sequence, the training efficiency and the performance are pretty surprising. It guarantees that personality-based conversation model to be a fruitful field for further investigation and research.

The whole machine learning algorithm is written in Numpy, it is available on Github.

`https://github.com/hchungdelta/Simple_NN_API/tree/master/NN_v.2.1_personality_model`

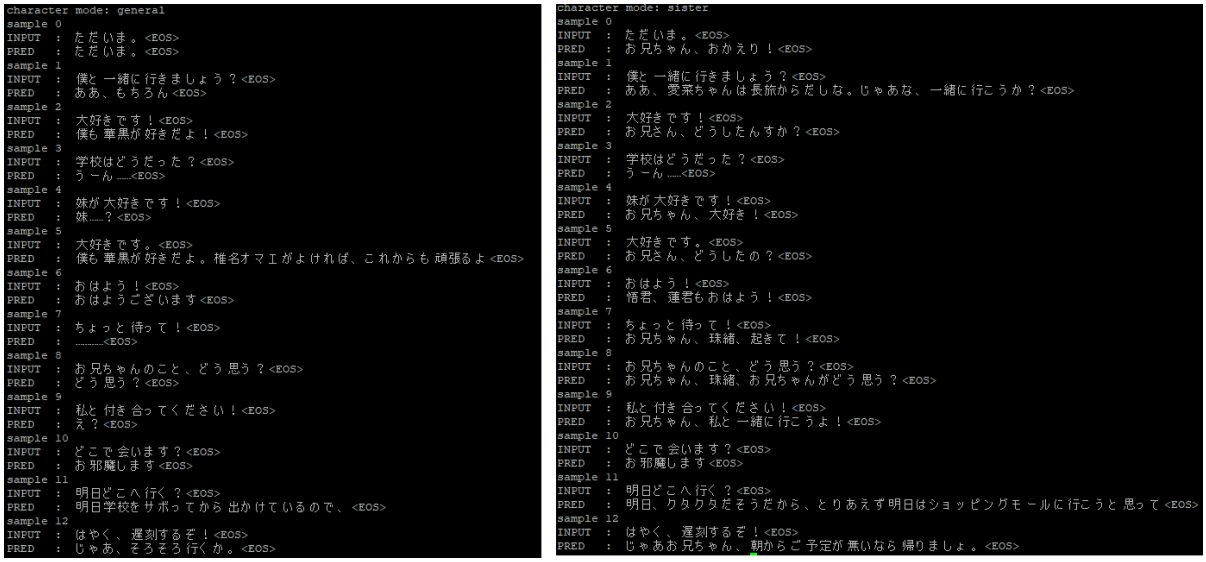

Figure 7: Some samples to demonstrate the performance of the model (Using 800k general-labeled data plus 80k sister-labeled data). Right: general personality, Left: sister's personality.

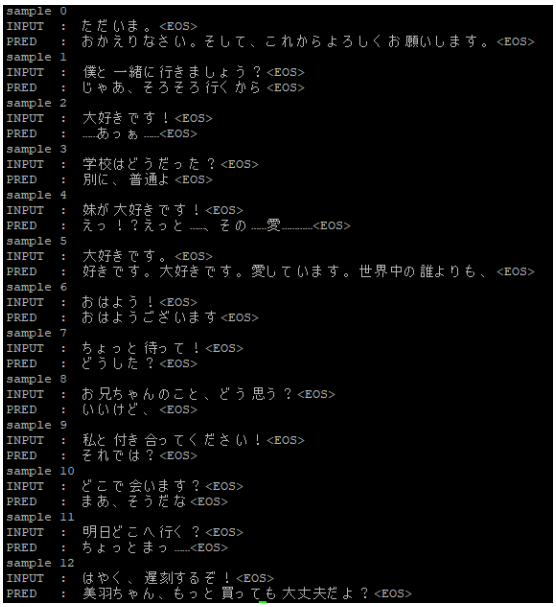

Figure 8: Some samples to demonstrate the performance of the second model (Using 80k sister-labeled data).

References

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long-short term memory. pages 1735–1780. Neural Computation 9(8), 1997.

[2] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoderapproaches. In *eprint arXiv:1409.1259*, 2014.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *eprint arXiv:1409.0473*, 2014.

[4] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *eprint arXiv:1508.04025*, 2015.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones Jakob Uszkoreit, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *eprint arXiv:1706.03762v5*, 2017.

[6] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. In *eprint arXiv:1607.06450*, 2016.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *eprint arXiv:1502.03167*, 2015.

[8] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. volume 45. IEEE TRANSACTION ON SIGNAL PROCESSING 11, 1997.

[9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *eprint arXiv:1412.6980*, 2014.

[10] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *eprint arXiv:1211.5063*, 2012.