YJTI at the NTCIR-13 STC Japanese Subtask

Toru Shimizu
Yahoo Japan Corporation
toshimiz@yahoo-corp.jp

ABSTRACT

In this paper, we describe our participation in the NTCIR-13 STC Japanese Subtask, in which we develop systems with the retrieval-based method. To retrieve reply texts for a given comment text, our system generates vector representations of both the comment and candidate replies by a 3-layer LSTM-RNN and evaluate distances between the comment vector and the candidate reply vectors, selecting the top-k nearest reply vectors and returning the corresponding reply texts. We also take Theme and Genre into consideration to decide the final ranking. In preparation of the candidate reply texts, we utilize all the comments and replies in the training set of Yahoo! News comments data. Our two runs are based on two different LSTM-RNN models, one trained over Twitter conversation data and the other mainly trained over Yahoo! Chiebukuro QA data. Each dataset has no less than 60 million pairs of text, and we aim to show how effective these combinations of large-scale datasets and large-scale neural models are for developing dialog systems. In addition, we had an assumption that the model of Twitter conversation data would outperform that of Yahoo! Chiebukuro QA data as the task domain seemed to be more similar to conversations in microblog services than social question answering, but the reported results revealed that it was not the case.

Team Name

YJTI

Subtasks

STC Japanese Subtask

Keywords

dialog systems, LSTM-RNN, social question answering, microblog

1. INTRODUCTION

This NTCIR-13 STC Japanese Subtask [11] is about developing dialogue systems which present reply texts to a given comment taken from Yahoo! News comments data. To realize such dialog systems, there are two well known approaches: the generation-based method and the retrieval-based method.

Vinyals and Le [16] proposed a generation-based system using the neural encoder-decoder model [2, 14]. A system built by such a model is supposed to be able to compose responses highly flexibly, sequentially arranging characters or words in the vocabulary freely. But in reality, without countermeasures, this type of implementation tends to lack diversity in responses, often producing trivial or non-committal sentences like "I'm OK" or "I don't know" [7, 8]. Also, training and sentence generation processes of typical, naively-implemented encoder-decoder models suffer from various problems such as exposure bias, label bias, and length bias [17, 12] which make it difficult to generate appropriate, natural responses.

On the other hand, retrieval-based systems using human-written text for candidate responses [9, 12] excel at providing fluent text while they do not have inherent flexibility which generation-based ones have. It is worth to note that the retrieval-based method is free from various problems originating from sentence generation in the generation-based method. Sountsov and Sarawagi [12] discussed a retrieval-based approach as a means to avoid the encoder-decoder model's length bias problem.

In this subtask, we explored the retrieval-based method with neural representation learning. One of our two assumptions was that lack of flexibility in this approach would not become a serious issue as the training set of the Yahoo! News comments data, from which we can choose reply texts, contains roughly 900k comment-reply pairs. The amount of texts would alleviate the problem as long as the system can find appropriate replies in the candidate set. The other assumption was that large-scale neural models trained by large-scale linguistic resources are essential to this kind of tasks. We trained two such LSTM-RNN models trained by two different datasets, Twitter conversation data and Chiebukuro QA data, each containing not less than 60 million pairs of posts. To confirm and compare the effectiveness of these resources (i.e. the models and the data behind them), we submitted two runs, YJTI-J-R1 based on Twitter conversation and YJTI-J-R2 mainly based on Chiebukuro QA. In this work, we demonstrate that a retrieval-based dialog system can be effective and that the combinations of two elements, a large-scale neural model and large-scale linguistic resources for training, is crucial to develop such systems.

2. OUR SYSTEM

Considering our usage of the long short-term memory recurrent neural networks (LSTM-RNNs) [4] and methods to train them, our overall approach can be referred as LSTM-DSSM and is similar to prior work [10] in which LSTM-RNNs are applied to the Deep Structured Semantic Model (DSSM), a model structure for semantic matching in infor-

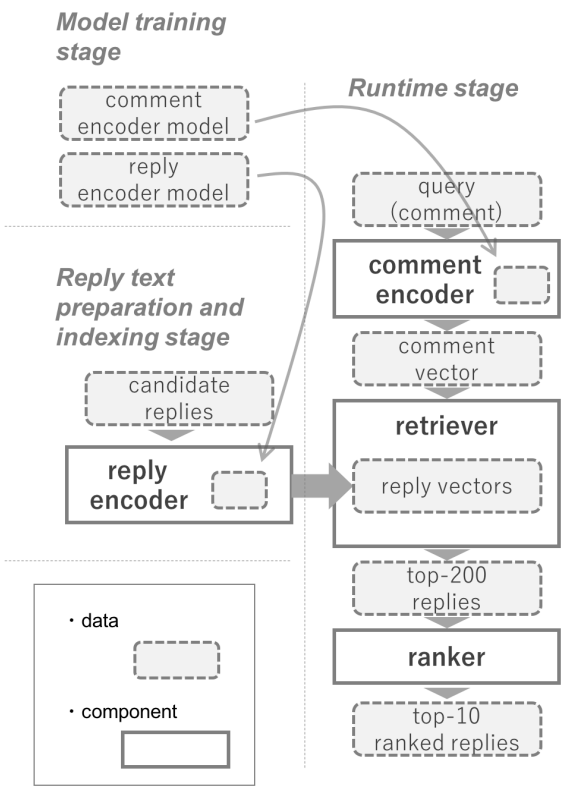


Figure 1: System overview

mation retrieval [5].

Our retrieval-based system has four components: two LSTM-RNN encoder models to generate a vector representations for a given text, a component to retrieve a list of replies on the basis of distances between generated vectors, and a component to determine a final selection of reply texts and their ranking. We call the first two components the *reply encoder* and the *comment encoder*, the third one the *retriever*, and the last one the *ranker*. We first describe the overview of the system into which these components are integrated, and move on to the details of them in the following subsections. Also, the overall system operations are separated into three stages: the model training stage, the reply text preparation and indexing stage, and the runtime stage. We train LSTM-RNN encoder models in the model training stage, prepare candidate reply comments and their search index in the reply text preparation and indexing stage, and then produces a set of up to 10 reply texts for a given comment in the runtime stage.

The overall processing flow to prepare one submission with our system is shown in Figure 1. First of all, in the model training stage, we train two LSTM-RNN encoder models, the comment encoder model and the reply encoder model, using a corpus containing a large number of query-document[1] pairs. The comment encoder model and the reply encoder model are identical in the architecture and almost all the parameters but are different only in parameters of the output layer. The trained models are built into the comment encoder component and the reply encoder component respectively for later use. In the reply text preparation and indexing stage, we produce candidate reply texts from the training set of Yahoo! News comments data containing 894,998 comment-reply pairs and their metadata. When we first checked the data, our impression was that not only the reply side but also the comment side can be used as candidate reply texts. So, we make two entries—one with the comment text and the other with the reply text—from one record in the original training set, copying the metadata to both entries. After uniquing, we obtain 1,167,796 candidate reply texts with metadata. Then, the reply encoder converts all the texts into 1024-element vector representations, and the retriever builds the index of the vectors for approximate nearest neighbor search. In the runtime stage, the comment encoder generates a 1024-element vector representation from a given comment text. Using the vector as a query, the retriever performs search over the indexed candidate reply texts, selecting 200 replies with the nearest vectors from that of the comment. The ranker evaluates the returned 200 replies considering the cosine similarity between vectors and two metadata attributes, Theme and Genre, and finalizes a top-10 ranked reply list. If there are multiple comments to produce responses, the system repeats the runtime stage to each of them.

Comparing YJTI-J-R1 and YJTI-J-R2, the methods are different only in the model training stage, and we describe their details in Sections 2.1.3 and 2.1.4 respectively. The other aspects of the specifications are the same between these two runs.

2.1 Encoder

This section explains the details about the comment encoder and the reply encoder.

2.1.1 Model architecture

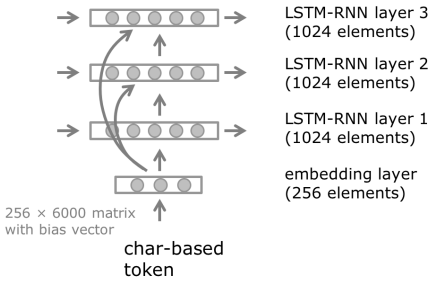
We used 3-layer LSTM-RNN encoders to produce vector representations of text. Figure 2a shows the structure of the LSTM-RNN part of an encoder. The vocabulary size is 6000, and each character token has a corresponding 256-element embedding vector representation. The embedding is fed to LSTM-RNN layers, each of which is the size of 1024. As in Figure 2b, the column of LSTM-RNN runs over given text represented by character-based tokens and then generates a 1024-element vector representation $\mathbf{z}$ through a fully-connected layer called the output layer, which uses a hidden layer vector at the last time step of the LSTM-RNN as the input.

We have two embodiments of such encoders, the comment encoder model and the reply encoder model as shown in Figure 2c. For a comment text Q and a reply text D, we can utilize the cosine similarity between their vectors to measure the similarity:

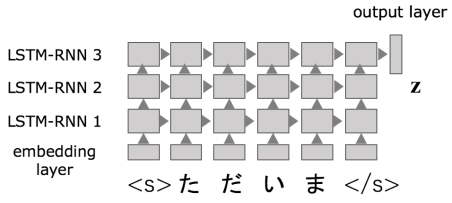
$$R_{\mathbf{\Theta}}(Q, D) = \frac{\mathbf{z}_Q^{\mathrm{T}} \mathbf{z}_D}{\|\mathbf{z}_Q\| \|\mathbf{z}_D\|}, \tag{1}$$

where $\mathbf{\Theta}$ denotes model parameters in the encoder pair. Aiming to learn representations so that vectors of a pair come close to each other, we consider a probability $P_{\mathbf{\Theta}}(D|Q)$ for Q and D being a pair when there are five choices $\{D^1, \ldots,$

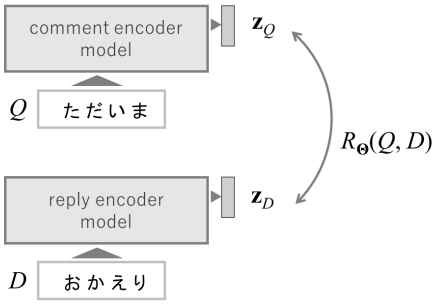
[1]Here, we use "query" and "document" as encompassing terms for other pairs connected in a similar manner such as comments and their replies, tweets and their reply tweets, or questions and their answers.



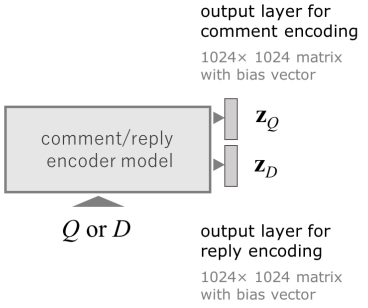
(a) LSTM-RNN model structure



(b) LSTM-RNN encoder to process text and produce a vector representation



(c) Comment encoder model and reply encoder model



(d) Two encoders represented as one combined model

Figure 2: Encoder model overview

$D^5\}$ for actual D:

$$P_\Theta(D_i^k|Q_i) = \frac{\exp(\beta R_\Theta(Q_i, D_i^k))}{\sum_{j=1}^5 \exp(\beta R_\Theta(Q_i, D_i^j))}, \quad (2)$$

which is obtained by feeding values of $R_\Theta(Q_i, D_i^k)$ with $k = 1, \ldots, 5$ into the softmax function. The index i denotes that the pair (Q_i, D_i^1) is the ith record in a dataset, and $k = 1$ always holds for a positive sample. We randomly pick up negative samples $\{D_i^2, D_i^3, D_i^4, D_i^5\}$ on the fly during training. For the inverse temperature coefficient β, we set it to 10 to make the cross entropy loss discussed below large enough. As the right choice among five D^ks is always $k = 1$, cross entropy loss l for the ith pair is written as

$$l_\Theta(Q_i, D_i^1) = -\log P_\Theta(D_i^1|Q_i). \quad (3)$$

Now, we can conduct training to find the optimal parameters $\hat{\Theta}$ minimizing the total loss L for all the pairs in the dataset:

$$\hat{\Theta} = \operatorname*{argmin}_\Theta L_\Theta \quad (4)$$

$$= \operatorname*{argmin}_\Theta \sum_i l_\Theta(Q_i, D_i^1) \quad (5)$$

$$= \operatorname*{argmin}_\Theta \sum_i \{ -\log P_\Theta(D_i^1|Q_i)\}. \quad (6)$$

We used Adam [6] for parameter optimization.

The comment encoder model and the reply encoder model are completely compatible with regard to the model architecture. They share parameters of the LSTM-RNN pipeline consisting of the embedding layer and the three LSTM-RNN layers but differ in parameters of the output layer. From a different perspective, as depicted in Figure 2d, these two models can be seen as one combined model comprising one LSTM-RNN pipeline and two same-sized fully-connected output layers for comment encoding and reply encoding respectively.

Concerning the details of the LSTM implementation, we followed the formulation of Graves 2013 [3], and the resulting total number of parameters in a model became 27.8 million. On a side note, we implemented the models with our in-house codebase supporting experiments using neural networks, which was written on top of Theano [15] and TensorFlow [1] so that a model can be developed, trained, and evaluated on both two frameworks compatibly.

2.1.2 *Datasets*

Table 1 lists datasets we used for training models.

For Twitter conversation data, we first collected 65.1M Japanese reply tweets from the Firehose and then retrieved their corresponding replied tweets. In this process, we filtered out short-lived throwaway accounts and bots with the following criteria.

- The account was at least 50 day old when the tweet was posted.
- The location is set.
- The profile image is set.
- The number of friends is greater than or equal to 10.
- The number of followers is greater than or equal to 10.
- The number of favorite posts is greater than or equal to 10.

name	type	no. of training records
Twitter language model data	posts	100.0M posts
Twitter conversation data	pairs of tweets and their replies	65.1M pairs
Yahoo! Chiebukuro language model data	questions and answers intermingled	202.0M posts
Yahoo! Chiebukuro QA data	pairs of questions and their answers	66.3M pairs

Table 1: Datasets used in the training process

- The source (a channel from which the tweet was posted) is included in our white list.

Satisfying these criteria suggests that the account is operated by a non-spamming human. We prepared the source white list checking the top-150 most frequent sources whether they are safe to include or not, obtaining 108 entries in the resulting list. In the postprocessing, URLs contained in the text were replaced with a fixed short string "[u]" which represent a URL's existence. Similarly, user mentions such as "@NTCIR" were replaced with "[m]" which represents a user mention's existence as these details are not essential to modeling conversations. We prepared Twitter language model data intermingling both sides of pairs into a list and sampling 100M tweets from it.

For Yahoo! Chiebukuro QA data, we used pairs of questions and their best answers posted from Jan. 2010 to May 2017. URLs contained in the text were replaced with a fixed short string "[u]" which represents a URL's existence. Yahoo! Chiebukuro language model data contains some answers which are not the best answers in addition to all the questions and the answers in Chiebukuro QA data.

While both datasets themselves, the Twitter conversation data and the Chiebukuro QA data, are proprietary to Yahoo Japan Corporation and not opened to public, there are ways to obtain equivalent datasets. Twitter's sample stream includes a sizable number of Japanese reply tweets, and researchers willing to build a conversational tweet corpus can collect them and also bulk-retrieve their corresponding replied tweets from a REST API which takes a list of tweet IDs as an argument and returns a list of the corresponding tweets. As for Chiebukuro's questions and answers, Yahoo! Chiebukuro data (2nd edition)[2] containing 16M questions and 50M answers is available for research purposes.

2.1.3 Model training for YJTI-J-R1

For our first submission YJTI-J-R1, we trained the comment encoder model and the reply encoder model over Twitter conversation data including 65.1 million pairs of posts as the training set. The model applied to this subtask achieved an accuracy of 0.835 in a matching task using the validation data to find a corresponding reply tweet of a given tweet from five choices including four negative samples. The accuracy of a similar matching task using the validation set of Chiebukuro QA data was 0.864. It was trained over 2.1 million iterations with the batch size 64, and the number of records the model went through is 135 million, which corresponds to 2.1 epochs of the training data. Table 2 summarizes the above mentioned details.

From our experience in training DSSM-like semantic matching models, the progress tends to be slow and takes weeks or months to reach convergence. To expedite the process, we usually start a training process with language modeling and then move on to a main task. In this case, we started the model training with language modeling using Twitter language model data and continued it for two epochs. When the model is trained as a language model, the LSTM-RNN column has two extra layers as in Figure 3a: the LM readout layer and the LM output layer. It is applied to a character-token sequence as shown in Figure 3b. After that, transferring parameters in the embedding layer and LSTM-RNN layers, we proceeded to the main task, which is DSSM training over the Twitter conversation data.

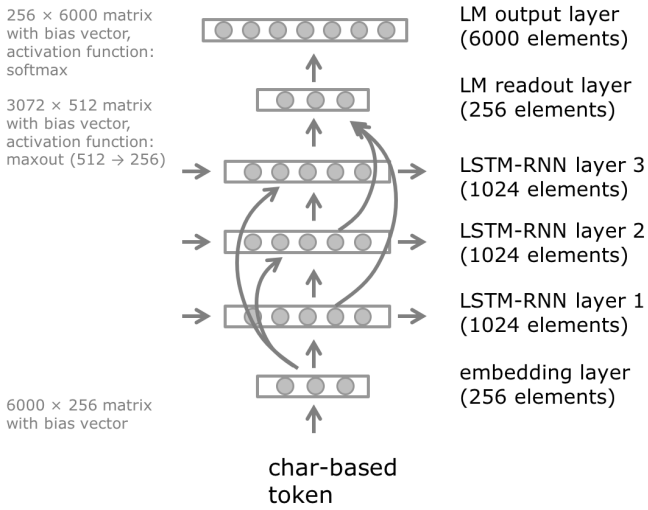
The intuition behind introducing language model pretraining is as follows. In the DSSM model training, the feedback signal is given only from the output layers connected to the last time step of an LSTM-RNN's sequence, and the backpropagation paths tend to be considerably long, spreading over the sequence. That makes credit assignment hard. Additionally, the feedback signals are "highly summarized" in a sense that it only tells the model which direction and how far a vector generated from a whole comment or reply should move. When a model is trained from scratch, it may require a large number of iterations to learn the vocabulary from such summarized signals. On the other hand, with language model training, a model receives feedback signals at all the time steps, and backpropagation paths from weight parameters to their nearest sources of feedback signals tend to be far shorter than those of DSSM training. That arguably leads to easier credit assignment. As the result, a model should be able to quickly learn the vocabulary guided by such "attentive" teaching signals even when it is trained from scratch. In future work, we will investigate the effectiveness of this approach quantitatively.

2.1.4 Model training for YJTI-J-R2

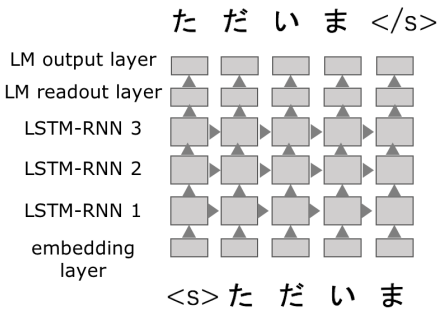
For our second submission YJTI-J-R2, we trained the comment encoder model and the reply encoder model in multitask settings. The main task is to learn an LSTM-DSSM semantic matching model using Yahoo! Chiebukuro QA data including 66.3 million pairs of questions and answers. In addition, there are two auxiliary tasks: one is to learn a language model over questions and answers intermingled, and the other is to learn an LSTM-RNN DSSM model over Twitter conversation data. The latter is identical to the training method for YJTI-J-R1 described in Section 2.1.3. Models in all the three tasks share parameters of the LSTM-RNN layers and the embedding layer, but they differ in parameters of the other parts.

In one iteration of this multitask training, the model goes through four inner iterations of language modeling, two inner iterations for the LSTM-DSSM on Twitter conversation data, and two inner iterations for the LSTM-DSSM on Chiebukuro QA data, as listed below.

[2] http://www.nii.ac.jp/dsc/idr/en/yahoo/chiebkr2/Y_chiebukuro.html



(a) LSTM-RNN model structure for language modeling



(b) Language model to predict the next character for a given input character token

Figure 3: Language model overview

model type	training data	batch size	iterations	records consumed	epochs
DSSM	Twitter conversation data	64	2.1M	135.0M	2.07

Table 2: DSSM training for YJTI-J-R1

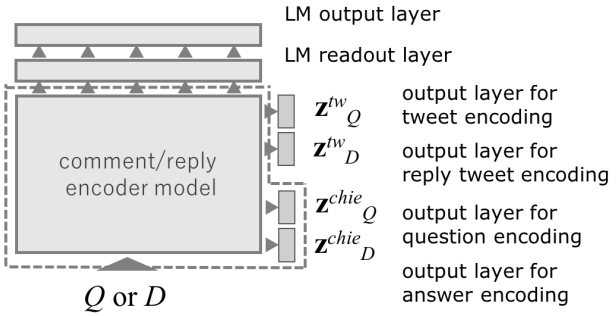


Figure 4: Components for the multitask training as one combined model

model type	training data	batch size	inner iterations	records consumed	epochs
lang. modeling	Chiebukuro language model data	64	2.7M	171.5M	0.85
DSSM	Twitter conversation data	64	1.3M	85.8M	1.32
DSSM	Chiebukuro QA data	32	1.3M	42.9M	0.65

Table 3: Multitask training for YJTI-J-R2

- Language model training using Chiebukuro language model data (1)
- Language model training using Chiebukuro language model data (2)
- Language model training using Chiebukuro language model data (3)
- Language model training using Chiebukuro language model data (4)
- LSTM-DSSM training using Twitter conversation data (1)
- LSTM-DSSM training using Twitter conversation data (2)
- LSTM-DSSM training using Chiebukuro QA data (1)
- LSTM-DSSM training using Chiebukuro QA data (2)

The model was trained over 670k such encompassing iterations. Further details are described in Table 3. Also in this case, we started the model training with language modeling using Chiebukuro language model data and then moved on

to the multitask settings.

This model achieved an accuracy of 0.967 in a matching task using the validation set of Chiebukuro QA data in which the model finds a corresponding answer of a given question from five choices including four negative samples. At the same time, the validation accuracy for Twitter conversation of the same model ended up at 0.759.

The overview of this multitask model comparable to Figure 2d of YJTI-J-R1 is shown in Figure 4. Each DSSM output layer consists of a 1024×1024 weight matrix and a bias vector. In the subsequent stages of YJTI-J-R2, we used parameters in the LSTM-DSSM model of Chiebukuro QA, which corresponds to the area enclosed by the dashed lines in Figure 4, for a comment encoder model and a reply encoder model.

Because of time constraints we had in participation to the NTCIR-13 STC Japanese Subtask, these two pretrained models of YJTI-J-R1 and YJTI-J-R2 were not trained from scratch but taken from existing training pipelines set up for other purposes, for which the configurations and the procedures had been fully recorded for reproducibility. Some unintended differences between pretrained models of YJTI-J-R1 and YJTI-J-R2, e.g. the latter was trained in the multitask settings whereas the former was not, are due to this circumstance.

2.2 Retriever

To conduct search over a large number of high-dimensional vectors of candidate reply texts, we utilize NGT[3], which implements a graph-based approximate similarity search method and is known to be achieving both precision and low latency [13]. In the nearest neighbor search, we use the L2-distance of vectors as the metric while it is practically equivalent to the cosine similarity as all the comment text and reply text vectors are restricted to have the unit length and can be considered as lying on a unit 1023-sphere in a 1024-dimensional space.

In the reply text preparation and indexing stage, NGT generates an index for 1.2 million texts taken from comments and replies in the training set of Yahoo! News comments data. In the runtime stage, the retriever receives a comment vector from the comment encoder, retrieves the 200 nearest replies from the index, and returns a list of them sorted in descending order of cosine similarity. Each item in the list is actually a tuple of four attributes: the comment text id, a list of Themes, the Genre, and the comment text itself.

2.3 Ranker

The ranker generates the ranking of the top-10 reply texts in a tiered manner. Here, we have three tiers each of which contains a partial ranked reply list: `THEME`, `GENRE`, and `OTHER`. When finalizing the ranking, we combine these tiers in the order of `THEME`, `GENRE`, and `OTHER`. As a result, `THEME` always occupy the top of the ranking, then those in `GENRE` follow, and items in `OTHER` fill the remaining slots. Also, a tier has its own scoring scheme, and replies within the tier are sorted in descending order of the score. The order is preserved when tiers are combined.

If the incoming comment text has one or multiple Themes, the ranker checks whether each reply text has any Theme matching with those of the comment or not, and bring matched reply texts into `THEME`. We calculate the score of these matched entries considering both the cosine similarity and the degree of matching of Themes. More specifically, we define the score s_i for ith item in the list as

$$s_i = sim_i^2 \sum_{k \in M_i} idf_k \tag{7}$$

where sim_i is the cosine similarity between the incoming query comment and the ith reply, M_i is a set of indices of Themes that are shared between the query comment and the ith reply, and idf_k denotes the inverted document frequency (IDF) score of a Theme t_k. The IDF scores were calculated from the training set of Yahoo! News comments data for all the Themes found in it $\{t_1, \ldots, t_N\}$. The score (7) can be interpreted as follows: the more and rarer the matched Themes are, the higher we rank the reply while also considering the cosine similarity. As for cosine similarity, we use the squared form of it to make the factor more effective than the linear form. It does not work as intended if the similarity is negative, but as far as we observe, the values are always positive in the current settings. After the score calculation and sorting, we truncate the reply list of `THEME` so as to make the length up to three.

If the incoming comment text has its Genre, the ranker deals with matched replies in `GENRE`. Within the tier, replies are simply sorted in descending order of the cosine similarity score. After the sort, we truncate the reply list of `GENRE` so as to make the length up to three.

The rest of replies go into `OTHER` and are sorted in descending order of the cosine similarity score. Table 4 shows summarized specifications of these three tiers.

Comparing the Theme and the Genre, the number of unique Themes extracted from the training set are 11,538 whereas that of unique Genres are 8. Given that the granularity of Themes is significantly finer than that of Genres, it is safe enough to say that replies in `THEME` have stronger signals for relevance than those in `GENRE` when it comes to metadata matching, therefore `THEME` should be ranked higher than `GENRE`. Nevertheless, while testing over the development set, we observed cases in which replies in lower tiers seemed to be more appropriate though overall the order `THEME`, `GENRE`, `OTHER` seemed to be right. To give replies in lower tiers chances to be included in the top-10 list, we restrict the maximum depth of `THEME` and `GENRE` to three, preventing those in `THEME` or `GENRE` from filling all the available slots, and ensure diversity of the selection criteria for a result set.

In addition to the procedure explained above, we pose a constraint on the final top-10 replies: the length of a comment text must be within a range of 10 to 45. A comment text with less than 10 characters might lack meaningful content. Conversely, a comment text with greater than 45 characters might be too specific to be relevant to a given comment. Lastly, we combine the lists of three tiers as described earlier, apply the text length constraint, unique the comment text, and truncate the items so that up to 10 of them remain in the list. The resulting list represents the system's final output for a given query comment.

The ranker's specifications were determined by trial and error over the development set of the Yahoo! News comments data without any quantitative analysis. Further exploration for more flexible, sophisticated blending and ranking algorithms using data-driven approaches is left as future work.

[3]https://research-lab.yahoo.co.jp/software/ngt/

tier	priority	score	no. of replies
THEME	1st	$sim_i^2 \sum_{k \in M_i} idf_k$	3
GENRE	2nd	sim_i	3
OTHER	3rd	sim_i	(no truncation)

Table 4: Specifications of the three tiers

metric	YJTI-J-R1	YJTI-J-R2
Mean nG@1	0.4322	**0.4893**
Mean $nERR$@2	0.4731	**0.5468**
Mean Acc_{L2}@1	0.1860	**0.2040**
Mean Acc_{L2}@2	0.1490	**0.2030**
Mean $Acc_{L1,L2}$@1	0.6480	**0.7620**
Mean $Acc_{L1,L2}$@2	0.6210	**0.7310**

Table 5: Official STC results of our runs (Rule-1)

metric	YJTI-J-R1	YJTI-J-R2
Mean nG@1	0.4171	**0.4726**
Mean $nERR$@2	0.4544	**0.5288**
Mean Acc_{L2}@1	0.1860	**0.2040**
Mean Acc_{L2}@2	0.1490	**0.2030**
Mean $Acc_{L1,L2}$@1	0.6100	**0.7200**
Mean $Acc_{L1,L2}$@2	0.5750	**0.6900**

Table 6: Official STC results of our runs (Rule-2)

3. EXPERIMENTS AND ANALSYSIS

3.1 Submitted Results

During the participation, we submitted two runs for the subtask.

- YJTI-J-R1: 3-layer LSTM-DSMM model mainly trained over Twitter conversation data
- YJTI-J-R2: 3-layer LSTM-DSSM model mainly trained over Chiebukuro QA data

In Tables 5 and 6, we excerpt the test run results of our submissions. The winning side of each metric is in boldface. Considering that our two runs have almost identical model structures and that YJTI-J-R1 shows a higher accuracy for Twitter conversation matching task, it is surprising that YJTI-J-R2 outperformed YJTI-J-R1 on all the available metrics.

Moreover, as these two runs performed competitively among the submitted runs, we can see that the overall approach we took to develop this system is reasonable.

3.2 Analysis

As for the matching task performances in Twitter conversation data and Chiebukuro QA data, which is to find the counterpart of a pair among five choices, accuracies by the model of YJTI-J-R1 trained by Twitter conversation data were 0.835 and 0.864 respectively, and accuracies by the model of YJTI-J-R2 mainly trained by Chiebukuro QA data were 0.759 and 0.967 respectively. In this comparison, we see a clear contrast in which excels at which as in Table 7. Given that YJTI-J-R2 won against YJTI-J-R1 in the official run, we can infer that Chiebukuro QA data was more effective than Twitter conversation data in this task.

matching task	YJTI-J-R1	YJTI-J-R2
Twitter conversation	**0.835**	0.759
Chiebukuro QA	0.864	**0.967**

Table 7: Comparison of matching task performances

4. CONCLUSIONS

Considering the competitiveness of our systems among submitted runs [11], it is confirmed that a retrieval-based system with a large-scale model trained over plentiful of linguistic resources can be particularly effective. Contrary to our anticipation that YJTI-J-R1 would outperform YJTI-J-R2 as conversational corpora would be more appropriate for dialog systems' training data than social question answering corpora, the results [11] indicate that YJTI-J-R1 based on Chiebukuro QA outperforms YJTI-J-R2 based on Twitter conversation in all the available metrics. This work suggests that social question answering data can be useful to learn models of topic-oriented conversations seen in Yahoo! News comments data.

5. REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *CoRR*, abs/1603.04467, 2016.

[2] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[3] A. Graves. Generating Sequences With Recurrent Neural Networks. *CoRR*, abs/1308.0850, 2013.

[4] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[5] P. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, CIKM '13, pages 2333–2338, New York, NY, USA, 2013. ACM.

[6] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[7] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan. A diversity-promoting objective function for neural conversation models. *CoRR*, abs/1510.03055, 2015.

[8] J. Li, W. Monroe, T. Shi, A. Ritter, and D. Jurafsky. Adversarial learning for neural dialogue generation. *CoRR*, abs/1701.06547, 2017.

[9] R. Lowe, N. Pow, I. Serban, and J. Pineau. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. *CoRR*, abs/1506.08909, 2015.

[10] H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. K. Ward. Semantic modelling with long-short-term memory for information retrieval. *CoRR*, abs/1412.6629, 2014.

[11] L. Shang, T. Sakai, H. Li, R. Higashinaka, Y. Miyao, Y. Arase, and M. Nomoto. Overview of the NTCIR-13 short text conversation task. In *Proceedings of NTCIR-13*, 2017.

[12] P. Sountsov and S. Sarawagi. Length bias in encoder decoder models and a case for global conditioning. *CoRR*, abs/1606.03402, 2016.

[13] K. Sugawara, H. Kobayashi, and M. Iwasaki. On approximately searching for similar word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2265–2275. Association for Computational Linguistics, 2016.

[14] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[15] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[16] O. Vinyals and Q. V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.

[17] S. Wiseman and A. M. Rush. Sequence-to-sequence learning as beam-search optimization. *CoRR*, abs/1606.02960, 2016.