Sequence to Sequence Modeling for User Simulation in Dialog Systems

Paul Crook, Alex Marin

Microsoft Corporation, Redmond, WA U.S.A.

pacrook@microsoft.com, alemari@microsoft.com

Abstract

User simulators are a principal offline method for training and evaluating human-computer dialog systems. In this paper, we examine simple sequence-to-sequence neural network architectures for training end-to-end, natural language to natural language, user simulators, using only raw logs of previous interactions without any additional human labelling. We compare the neural network-based simulators with a language model (LM)-based approach for creating natural language user simulators. Using both an automatic evaluation using LM perplexity and a human evaluation, we demonstrate that the sequence-to-sequence approaches outperform the LM-based method. We show correlation between LM perplexity and the human evaluation on this task, and discuss the benefits of different neural network architecture variations.

Index Terms: dialog systems, sequence-to-sequence modelling, user simulators

1. Introduction

Simulated users have long been used to enable both offline training and evaluation of spoken dialog systems (SDSs) [1, 2, 3, 4, 5, 6, 7] in the absence of large numbers of users, or tolerance of users to poor performance during the initial training phase. With newer training methods allowing rapid bootstrapping of SDSs [8], or training directly from data, the need for user simulators for training has somewhat declined. However, they retain an important role in evaluation, especially of industrial systems [9, 10]. An enduring advantage of simulated users is that they generalize from static log data, which can allow offline testing of SDSs that differ from those on which the data was collected. Our long-term aim in pursuing this work is the ability to offline measure incremental improvements of the dialog engine and check for regressions.

Previous simulated users in the literature usually have inputs and outputs that operate at the dialog act level [1, 5, 11] instead of natural language (NL). Typically, they are addressing the functionality of the dialog manager in exclusion of other components. To this end, data-driven simulated users require human annotation of log data with both system and user dialog acts and, for agenda based [12, 5] simulated users, the manual design and construction of some internal simulated user state, to track against the associated agenda. Agendas address an important problem in SDSs training: without some form of context tracking to ensure consistency, the dialog manager can end up dictating both the conversation *and* the simulated user's requirements, *e.g.* everyone gets a down-town Italian restaurant.

With the advent of deep-learning it is possible to train sequence-to-sequence (*seq-2-seq*) models as simulated users that track an external agenda [11]. It would seem likely that this can be extended to also learning their own internal state. Furthermore, if such models are trained to operate with NL input and output, then no additional human annotation would be required to train the models from log data. This paper presents some initial steps in this direction, with sequence-to-sequence models trained as NL-to-NL user simulators with no additional human annotation. We also explore the effect of adding some basic dialog context as input to these models.

The remainder of the paper is laid out as follows. Section 2 describes related work, section 3 describes the models, section 4 the automatic and human evaluations, and section 5 concludes with a brief discussion and future work.

2. Background

Many previous approaches, including for agenda-based systems, operate at the dialog act level [1, 5], requiring human annotation of user dialog acts. Even those that interact with SDSs at the level of NL or speech [2, 7] have required manual curation of data and often hand crafting of interaction statistics. Asri et. al. [11] present a *seq-2-seq* user simulator model that takes into account context. However, this model operates at the dialog act level rather than the NL-to-NL level. Other similar *seq-2-seq* models have been recently proposed. Our architecture is, for example, very close to that proposed by Serban et. al. [13], however their target is chat-oriented dialog systems whereas ours is task-oriented user simulation.

We are not aware of any published work on *seq-2-seq*, NL-to-NL user simulators. The contribution of this paper is the initial investigation of such models.

3. Models

Given the general success across a wide set of tasks and the simplicity of design of the *seq-2-seq* model proposed by Sutskever at al. [14], we adopt this model as our starting point for our investigation with one initial modification, see solid outlined boxes in figure 1. The basic *seq-2-seq* model consists of an encoder (solid boxes labeled GRU in figure 1) that consumes the input sequence, in this case the words of the system prompt, and generates a fixed size summary vector. This vector is then fed into a decoder (labeled LSTM) that generates an output sequence, in this case the words of the user response. Recurrence in both encoder and decoder is lossy, with some information earlier in the sequence being lost as encoding/decoding progresses. In this application, it is important that the decoder has in mind the summary vector that represents the system prompt (and later the system prompt and context) for each step of generating output words, especially for long user utterances. Thus, we introduce connections from the summary vector to each time step of the unrolled decoder. Within the Keras framework [15], this is easiest achieved by inserting an additional dense layer between the encoder and decoder. Additionally, in future work, it is proposed to make this layer recurrent over dialog turns, *i.e.* capture context information in the summarization layer. A sim-

We would like to thank Zhalleh Feizollahi for invaluable help in annotating our human and simulated user dialogs.

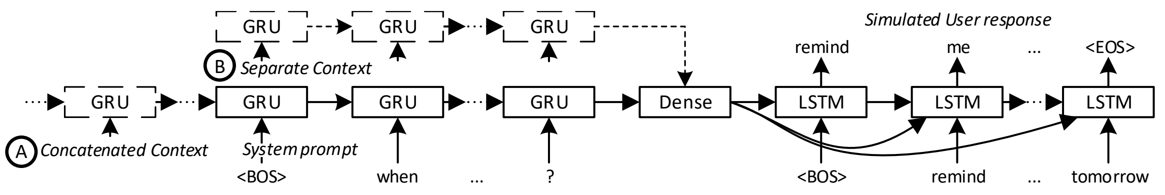

Figure 1: *Architecture of seq-2-seq models with encoders (GRU) and decoder (LSTM) unrolled over 'time' (word sequences).*

ilar intermediate layer was adopted by Serban et. al. [13] in construction of chat-bot dialog systems.

In this paper three models are tested: *No-Context*, *Concatenated-Context* and *Separate-Context*. The *No-Context* model, drawn with solid outlined boxes in figure 1, has a single input encoder that receives only the NL system prompt. *Concatenated-Context* also has a single encoder but it is extended to included context information which is prepended ahead of the system prompt; label A in figure 1. The third, *Separate-Context*, has an additional encoder for context (label B) that is independent of the system prompt encoder. The output vectors of these two encoders are concatenated as input to the intermediate dense layer.

Multiple GRU units [16] make up the 1 or more encoders. Input words to GRU layers are embedded using a pre-trained embedding. The decoder consists of multiple LSTM units [17] whose inputs, for each time step, are the summary vector of the NL system prompt (plus context in the latter two models), and an embedded vector representing the user simulator's previous word in its response. Input words to the LSTM are also embedded using a pre-trained embedding. The LSTM layer's output is connected to a layer of softmax units, with one softmax unit per word in the user simulator vocabulary. The softmax units are trained with reference to 1-hot vectors of the words in the target utterance; thus, they learn a distribution in the target word space. This distribution is sampled from at evaluation time to generate new utterances, with the likelihood of each word being selected being proportional to the softmax unit activation.

GRUs were selected for the input layer due their relative simplicity compared with the LSTM. LSTM was selected for the output layer based the de-linkage between its internal state and its output, this was thought likely to complement the external recurrent loop that is feeding back selected words to the next time-step. No experimentation was done to determine if this arrangement was optimal compared to any other, *e.g.* we didn't investigate the use of GRUs or LSTMs for both input and output recurrent layers. The intermediate dense layer uses rectified linear units (ReLU) [18].

The embedding layers were trained separately from the main model using the GSIM implementation of Word2Vec [19] and the complete corpus of either system or user responses captured in the log data. The *seq-2-seq* models in this paper used the following dimensions: embedding vectors of length 100, 512 unit GRU layers, dense ReLU layer and LSTM layer, softmax layer and user vocab size (words) 13,768, system vocab size 11,507 words.

4. Experiments

4.1. Data and Evaluation

We perform experiments using data from the *reminder* domain of the Cortana personal digital assistant. The reminder domain allows users to set reminders by voice or typed input, using a custom reminder text message and based on a variety of triggers, such as at specific times or upon arrival at specific locations. A confirmation-and-correction phase of the dialog allows users to change previously-provided information. An illustrative example is shown below.

User: Remind me to call Ann tomorrow at 4PM to cancel page.
Cortana: Alright, remind you to Call Ann at 4PM tomorrow. Is that right?
U: No.
C: OK, should I change the reminder, date, or time?
U: The reminder.
C: What's your reminder?
U: Call Ann, the cancer patient.
C: Ok, I'll remind you to Call Ann, the cancer patient, at 4PM tomorrow. Sound good?
U: Yes.
C: Got it.

Log data consisting of approximately 40,000 sessions (136,000 turns) is split 60-20-20 into training, development, and evaluation sets. All data was sessionized and automatically annotated by the Cortana runtime with system-side dialog acts. Additionally, the user side utterances were automatically tagged by a domain-matched intent and slot tagger [20], pre-trained using non-overlapping training data.

For this specific task Cortana supports 60 unique system dialog acts, covering individual *request* dialog acts for each task parameter, task-level and parameter-level *inform* dialog acts, as well as more specialized flavors of *inform* to cover business requirements. The NLU schema contains 14 slots.

Performance of the simulator user models is measured offline, using automatic evaluation, as well as online with each simulator interfaced to Cortana.

The offline, automatic evaluation is intended to assess the performance of a particular simulator model in aggregate. Liu et al. [21] discuss the ineffectiveness of other unsupervised metrics, such as BLEU [22], METEOR [23], and ROUGE [24], in particular in the context of evaluating the dialog system responses which may or may not have strong surface form overlap with the ground truth. In evaluating user simulators, the problem is compounded by the fact that ground truth may not even be available, especially when the simulator must generate a relatively open-ended slot value (such as the text of a reminder). Thus, instead of evaluating each generated utterance against a set of references, we use an aggregate evaluation approach, inspired by language modelling methods. Given a simulated user model, we compute the perplexity of the evaluation set data against a language model trained on utterances generated by that simulated user model. To generate data, we use the development set sessions as templates[1]. That is, for each system-side utterance (and associated real user context utterances, if any) in each session, we generate a new user-side utterance using the simulator model. This approach does not yield new complete (and coherent) dialogues, since the generated user utterances may diverge, and the succeeding system utterances remain fixed, rather than adapting to the simulated user utterances. However, given a good simulated user model, we expect that the generated data, in aggregation, will approximate the behavior of real users (also in aggregation). The perplexity

[1]Note this is the only use of the development set, it is not used for model tuning.

measure thus allows us to evaluate how well the simulated user data matches data obtained from real users.

The online evaluation interfaces the user simulator with the publicly available version of Cortana in Windows 10 for PC, Anniversary Update. This version was not restricted in any way: Cortana could misinterpret the simulated user and respond with any supported task [20] or with web results. As Cortana's interaction differs for speech vs. text input, each user simulator utterance is first converted to speech using the Microsoft Cognitive Services, Bing Speech API TTS engine [25]. The Cortana responses are captured in real-time from the Windows 10 event logs and are used as input to the next execution of the simulator, together with the previous simulator-generated utterance (if any) as context. A session continues until either Cortana terminates the interaction (by setting a reminder for the simulated user or reaching a terminal state in the attempted task) or until the simulated user model generates a special token that terminates the session. The generated sessions are evaluated in aggregate by a human annotator. The annotation scheme was designed in collaboration with a computational linguist. Each session is assigned two scores on a scale of 1-5, indicating naturalness and discourse-level cohesion, respectively. Naturalness (is this a grammatically correct and understandable dialog) is expected to correlate with perplexity, while discourse cohesion captures how natural the conversational flow is (e.g. did the simulated user answer system prompts reasonably).

4.2. Baseline: LM-based simulator

As baseline, we use a natural language simulator built using language modelling techniques. LM-based user simulators are a popular approach for building simulated users, in particular for dialog system evaluation tasks [4]. A common approach in LM-based simulation is to model the dialog as a sequence of user and system actions that alternate [26]. In this paradigm, the user action a_u can be predicted from the previous system action a_s, as $p = P(a_u|a_s)$.

The text generation is done using a generative LM approach, similar to the class LM approach described by Oh and Rudnicky [27]. We treat each user action a_u as a separate class for LM generation. Given the strong system-directed aspect of our domain, we simplify this model further, tying the user act distribution such that $P(a_u|a_s) = 1$ if $a_u = a_s$ and 0 otherwise. In other words, we use all user utterances in the training data that follow a particular system dialog act a_s to train the corresponding LM for generating the user act a_u.

Unlike in [27], we use a hierarchical generation approach. That is, we first train a template LM for each class, where the text covered by any NLU slot is collapsed to the corresponding slot tag name. Then, for each slot tag, we train a separate LM dedicated to just producing text that should fill this slot. This allows for different dialog acts that use a particular slot to share the corresponding slot LM, while each dialog act still uses a separate LM trained using template sentences. At decoding time, we first generate a template given the appropriate dialog act class. Each slot tag contained in the template is replaced with text generated from the corresponding slot tag LM.

All LMs are trained using the training set data. The data is first tagged using the external, domain-matched slot tagger [20]. The word sequences covered by slot tags are collapsed to each respective slot tag label to generate the templatized strings for training dialog act-level LMs, and extracted to train the slot tag-level LMs. We experiment with different LM orders for both the dialog act-level LMs and the slot tag-level LMs. All LMs are trained using the SRILM toolkit [28].

Table 1: *Comparison of LM-based and Sequence-to-Sequence simulators without any context modelling, using LM perplexity of the evaluation set as the automatic performance measure.*

LM Source	**Sim Configuration**		**Perplexity**
training transcripts	n/a		13.5
dev transcripts			15.2
LM Sim	**LM Order**		
	Dialog Act	**Slot**	
dev generated	5	5	83
	5	2	111
	2	5	85
	2	2	117
Seq2Seq Sim	**Context Type**		
dev generated	no context		17.1

4.3. Experiment: LM vs. Sequence-to-Sequence Simulators

In the first experiment, we compare the performance of the baseline LM-based simulator against the sequence-to-sequence simulator without any context. We use the offline, automated LM perplexity-based measure. For each simulator configuration, we first use the trained simulator to generate utterances using the dev set sessions as templates. For comparison, we also include the perplexity of the eval set user utterances given the real user utterances in the training and dev sets, to show the effect of using more data as well as using the same amount of real user data vs. simulated user data.

Results are shown in table 1. The first two rows in the table indicate the perplexity of the eval set given language models trained on human user transcripts. The small difference in performance between using the training vs. the development set as LM training shows the impact of the corpus size. All the LM-based simulators exhibit significant performance degradation; the utterances generated by these simulators are not sufficiently similar to the true human transcripts to predict the eval set; in general, performance is better when using longer-span context in the Slot LMs, even though the available data to train some of the slot LMs is limited; we hypothesize this is primarily due to the general text saved in the *reminder text* slot. The sequence-to-sequence simulator recovers most of the performance lost by the baseline LM simulators, yielding only a small performance degradation relative to the human transcript-based evaluation.

4.4. Experiment: Comparison of Different Sequence-to-Sequence Architectures

Next, we compare the effect of different context modelling approaches in our sequence-to-sequence architecture. For each model configuration (*None* if using no context, with the previous user-side utterance *Concatenated* to the system utterance as input, or with the previous user-side utterance used as a *Separate* input layer), we train a corresponding model for up to 200 epochs. We select the model configuration corresponding to the lowest loss function value for evaluation. The model selected for the *None* configuration was also used in table 1.

Results are shown in table 2. Over all, the perplexity numbers obtained from each simulator configuration are very similar, with no significant differences in the results. We observe that the training objective for the *None* configuration appears to converge more slowly than either context-aware configuration.

Table 2: *Effect of context modelling on Sequence-to-Sequence simulators, as evaluated using evaluation set LM perplexity.*

Context	Epoch	Perplexity
None	160	17.1
Concatenated	60	17.4
Separate	80	17.4

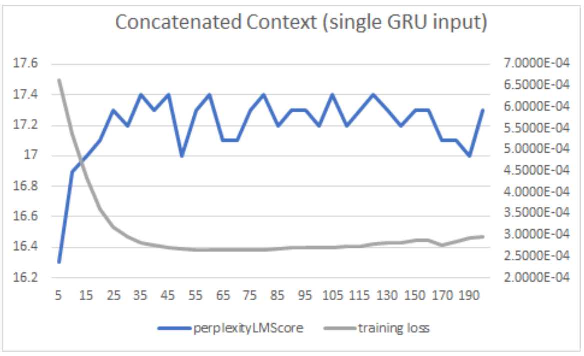

Figure 2: *Eval set perplexity and training loss at each epoch.*

We perform two additional studies to evaluate the correlation between the training objective and the perplexity measure. Figure 2 plots the training loss together with the corresponding model's eval set perplexity, as evaluated every 5 epochs during the training process. We use the *Concatenated* context model configuration (other architectures display similar behaviour). We see that the loss function drops smoothly until around 60-80 iterations; the improvement then stalls, with the loss function displaying only minor fluctuations in later iterations. The perplexity, on the other hand, shows optimal behaviour at the first evaluation of the model (epoch 5); performance after that is essentially flat, with insignificant fluctuations.

Given that the perplexity score was optimal as early as iteration 5, we perform an analysis of the perplexity scores computed for each of the first 10 epochs of training of each model architecture. Results are shown in figure 3. The trends are similar for all three architectures, with the first epoch yielding substantially higher perplexity scores than the rest, and the scores flattening after the first five epochs. Even though the loss function continues to improve over a much longer period of time, the performance of the user simulators stabilizes much faster. The *Separate context* model generally has lower perplexity than the other architectures, though the differences are not significant.

We additionally perform a live evaluation of the best simulator model from each model architecture, as described in section 4.1. For each model architecture, we generated 100 sessions by interfacing the simulator with the live Cortana system, and selected 100 sessions from the offline evaluation set. We had a single human judge score all sessions. The average scores for each simulator model configuration, as well as the human sessions, are shown in table 3. Differences between the *No context* configuration and the human user sessions are significant at $p = 0.1$; other differences are not significant.

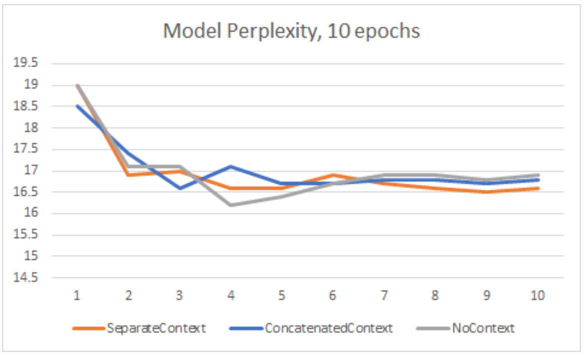

Figure 3: *Eval set perplexity for each model architecture as measured at each of the first ten epochs of training.*

Table 3: *Human evaluation of the three sequence-to-sequence user simulator architectures compared to human user data*

User Type	Context	Naturalness	Cohesion
Human	n/a	4.56	4.52
Simulated	None	4.31	4.33
	Concatenated	4.45	4.52
	Separate	4.41	4.41

Table 4: *Task success during online evaluation.*

Statistic (% of total)	Human Sessions	Simulator Context None	Concat.	Separate
Success	62.9	56.5	58.2	63.7
Cancelled	1.9	4.3	5.2	8.0
Abandoned	33.3	30.4	29.8	23.9
Other	1.9	8.8	6.8	4.4
Goal Changes	6.6	5.2	7.5	14.2
ave. #turn pairs	3.2	2.8	3.1	3.5

Table 4 shows some additional statistics regarding the human sessions as compared to the live sessions recorded from each simulator. The *Separate context* model best matches the successful completion rate of the human sessions; it also has a much lower abandonment rate than the *No context* and *Concatenated context* models. The *Other* category includes various conditions, such as Cortana losing focus or incorrectly ranking a user query as reminder; these are also lowest in the *Separate context* configuration, suggesting that the dialogs better match system expectations. Interestingly, the *Separate context* configuration has a much higher rate of goal changing than the others; anecdotally, it appears that many of the extra goal changes are due to repeated goal-changing in a single session.

5. Conclusion

In this paper, we introduced a class of NL-to-NL simulated user models based on sequence-to-sequence architectures, and showed that they significantly outperform a simulated user based on language modelling techniques. These models are trained from log data, with no further human annotation, allowing them to be automatically trained and deployed.

All three *seq-2-seq* architectures examined perform similarly and closely match human user data when hand-evaluated for naturalness and discourse cohesion. This matches the LM perplexity measure results that also show little difference between the models. The *seq-2-seq* models approximate the human distribution over Complete, Cancelled, Abandoned, Goal Change and average dialog length, showing that these models learned not only to reproduce natural sounding dialogs for this domain but also to match the general statistics observed in real user data. Both points are important when using such models to evaluate dialog system improvements.

Future work will focus on additional architectures for better modelling context, and apply the methods to additional domains and more sophisticated dialog systems, *e.g.* by allowing more user initiative in the interactions.

6. References

[1] K. Scheffler and S. Young, “Corpus-based dialogue simulation for automatic strategy learning and evaluation,” in *Proceedings NAACL Workshop on Adaptation in Dialogue*, 2001.

[2] R. López-Cózar, A. De la Torre, J. C. Segura, and A. J. Rubio, “Assessment of dialogue systems by means of a new simulation technique,” *Speech Commun.*, vol. 40, no. 3, pp. 387–407, May 2003. [Online]. Available: http://dx.doi.org/10.1016/S0167-6393(02)00126-7

[3] H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira, “Human Computer Dialogue Simulation Using Hidden Markov Models,” in *Proc. ASRU*, 2005.

[4] K. Georgila, J. Henderson, and O. Lemon, “User Simulation for Spoken Dialogue Systems: Learning and Evaluation,” in *Proc. Interspeech*, 2006.

[5] J. Schatzmann and S. Young, “The hidden agenda user simulation model,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 17, no. 4, pp. 733 – 747, 2009.

[6] S. Keizer, F. Jurek, F. Mairesse, B. Thomson, K. Yu, and S. Young, “Parameter estimation for agenda-based user simulation,” in *in SIGDIAL 10: Proc. of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, 2010.

[7] J. Gtze, T. Scheffler, R. Roller, and N. Reithinger, “User simulation for the evaluation of bus information systems,” in *2010 IEEE Spoken Language Technology Workshop*. IEEE, 2010, pp. 454–459.

[8] M. Gasic and S. Young, “Gaussian processes for pomdp-based dialogue manager optimization,” *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 22, no. 1, pp. 28–40, Jan. 2014. [Online]. Available: http://dx.doi.org/10.1109/TASL.2013.2282190

[9] S. Möller, R. Schleicher, D. Butenkov, K.-P. Engelbrecht, F. Gödde, T. Scheffler, R. Roller, and N. Reithinger., “Usability engineering for spoken dialog systems via statistical user models,” in *In First International Workshop on Spoken Dialogue Systems Technology (IWSDS)*, 2009.

[10] T. Scheffler, R. Roller, and N. Reithinger, “Speecheval: A domain-independent user simulation platform for spoken dialog system evaluation,” in *Proceedings of the Paralinguistic Information and its Integration in Spoken Dialogue Systems Workshop (IWSDS)*, 2011, pp. 295–300.

[11] L. E. Asri, J. He, and K. Suleman, “A sequence-to-sequence model for user simulation in spoken dialogue systems,” *CoRR*, vol. abs/1607.00070, 2016. [Online]. Available: http://arxiv.org/abs/1607.00070

[12] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. Young, “Agenda-based user simulation for bootstrapping a pomdp dialogue system,” in *Proc. NAACL*, 2007, pp. 149–152.

[13] I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau, “Building end-to-end dialogue systems using generative hierarchical neural network models,” in *Proc. AAAI*, 2016. [Online]. Available: http://arxiv.org/abs/1507.04808

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

[15] F. Chollet *et al.*, “Keras,” https://github.com/fchollet/keras, 2015.

[16] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[17] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, November 1997.

[18] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Frnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814. [Online]. Available: http://www.icml2010.org/papers/432.pdf

[19] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *CoRR*, 2013.

[20] R. Sarikaya, P. Crook, A. Marin, M. Jeong, J.-P. Robichaud, A. Celikyilmaz, Y.-B. Kim, A. Rochette, O. Z. Khan, X. Liu, D. Boies, T. Anastasakos, Z. Feizollahi, N. Ramesh, H. Suzuki, R. Holenstein, E. Krawczyk, and V. Radostev, “An overview of end-to-end language understanding and dialog management for personal digital assistants,” in *Proc. SLT*, 2016.

[21] C. W. Liu, R. Lowe, I. V. Serban, N. Noseworthy, L. Charlin, and J. Pineau, “How NOT to evaluate your dialogue system: An empirical study of unsupervised evaluation metrics for dialogue response generation,” in *Proc. EMNLP*, 2016.

[22] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL ’02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: http://dx.doi.org/10.3115/1073083.1073135

[23] S. Banerjee and A. Lavie, “METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgements,” in *Proc. ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization*, 2005.

[24] C.-Y. Lin, “ROUGE: a Package for Automatic Evaluation of Summaries,” in *Proc. Workshop on Text Summarization Branches*, 2004.

[25] “Microsoft Cognitive Services Bing Speech API TTS,” https://www.microsoft.com/cognitive-services/en-us/speech-api, accessed: 2017-03-21.

[26] W. Eckert, E. Levin, and R. Pieraccini, “User modelling for spoken dialogue system evaluation,” in *Proc. ASRU*, 1997.

[27] A. H. Oh and A. Rudnicky, “Stochastic natural language generation for spoken dialog systems,” *Computer Speech and Language*, no. 16, pp. 387–407, 2002.

[28] A. Stolcke, “SRILM – An Extensible Language Modeling Toolkit,” in *Proc. ICSLP*, 2002.