End-To-End Natural Language Generation Using LSTM-based Networks

Geetanjali Rakshit, Kevin K. Bowden, Panagiotis Karagiannis and **Juraj Juraska**
Baskin School of Engineering
University of California Santa Cruz
{grakshit,kkbowden,pkaragia,jjuraska}@ucsc.edu

Abstract

Natural language generation lies at the core of generative dialogue systems and conversational agents. This paper studies two statistical generators based on Long Short-term Memory neural networks. Both networks learn from unaligned data and jointly perform sentence planning and surface realizations. This is a promising new approach for natural language generation, since it eliminates the need for hard-coded heuristic rules and hence is easily generalizable to new domains. We perform post processing on the output from the neural network architecture, to make the utterances more readable and natural. We report our results after experimenting on the E2E NLG Challenge dataset.

1 Introduction

With the continuous efforts to develop general artificial intelligence, there has recently been a substantial amount of research done in the fields of natural language processing (NLP) and generation (NLG). It gave rise to popular digital personal assistants in smartphones, such as Siri or Cortana, as well as separate devices with the sole purpose of offering the services of such personal assistants through interactive communication, such as Amazon Echo (powered by the Alexa assistant) or Google Home. The capabilities of these conversational agents are still fairly limited and lacking in various aspects, ranging from distinguishing the sentiment of human utterances to producing utterances with human-like coherence and naturalness.

In our work we focus on the NLG module in task-oriented dialogue systems (see Fig. 1), in particular on generation of textual utterances from structured *meaning representations* (MRs). An MR describes a single dialogue act in the form of a list of pieces of information that need to be conveyed to the other party in the dialogue (typically a human user). Each piece of information is represented by a slot-value pair, where the *slot* identifies the type of information and the *value* is the corresponding content (see Table 1). *Dialogue acts* (DAs) can be of different types, depending on the intent of the dialogue manager component. They can range from simple ones, such as a *goodbye* DA with no slots at all, to complex ones, such as an *inform* DA containing multiple slots with various types of values.

Table 1: An example of a meaning representation (MR) and a corresponding utterance.

MR	*inform*(name[**The Golden Curry**], food[**Japanese**], priceRange[**moderate**], kidsFriendly[**yes**], near[**The Bakers**])
Utterance	Located near **The Bakers**, **kid-friendly** restaurant, **The Golden Curry**, offers **Japanese** cuisine with a **moderate** price range.

The objective of the language generation phase of a spoken dialogue system is to produce a syntactically correct utterance from a given MR, which is the outcome of the previous phase (dialogue management). Aside from being correct, the utterance should express all the information contained in the MR, and the formulation should be as natural as possible. The process of assembling an utterance from an MR is often performed in two stages: *sentence planning*, which determines the structure of the utterance (order in which the information is presented, number of sentences, etc.), and *surface realization*, in which the actual words are plugged in and the utterance is given a linear form.

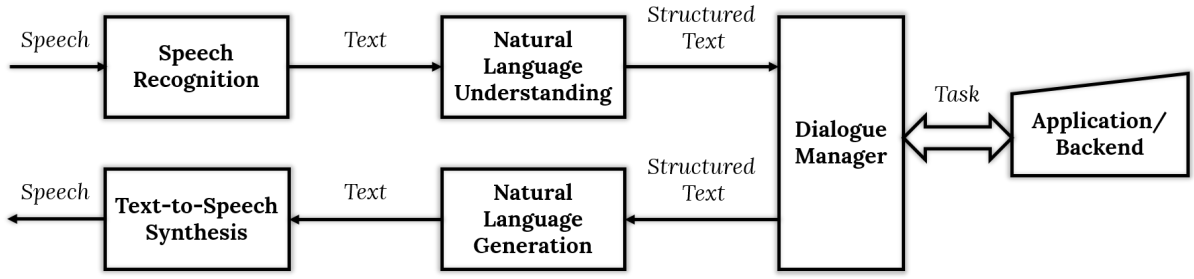

Figure 1: Architecture of a spoken dialogue system.

2 Related Work

Researchers have taken various statistical approaches to solving the language generation problem over the years. One of the first notable efforts was done by **?** who augmented a handcrafted generator with statistical reranking. The generator produces candidate utterances based on hard-coded grammar rules, and then an n-gram language model, trained on a corpus of data, reranks the candidates and selects the best among them. **?** took the idea one step further by incorporating a trainable sentence planner, which was later improved by **?** by making it better adaptable to different domains. Another line of research is focused on learning the parameters of the model through optimizing an objective function. An example of such a method is using reinforcement learning to train a language generation policy, such that the expected reward is maximized (**?**). Nevertheless, all of the above heavily rely on handcrafting parts of the model, which results in a decreased flexibility and portability to different domains, as well as limited variability of the utterances.

In order to avoid the inconvenience of developing grammars or rules, and maintaining them, new approaches emerged attempting to train the language model as a whole on a corpus of data. The earlier works among them used semantically aligned data to train the language models (**??**). The alignment provides valuable information during training, but requires significant additional effort in properly annotating the entire dataset. Therefore, it is feasible to be performed on small datasets only, and preferably by annotators with appropriate linguistic knowledge. On the other hand, the extra information enables the model to learn faster from a smaller dataset, which may be an important factor in certain domains where data is hard to obtain.

More recently, methods were developed that can do without any annotations in the data and often use an *end-to-end* approach to training, performing the sentence planning and the surface realization simultaneously (**?**). Later systems trained on unaligned data successfully utilized LSTM recurrent neural networks paired with an encoder-decoder system design (**???**), or other concepts, such as imitation learning (**?**). These generation models, however, typically require greater amount of data to be able to learn, due to the lack of semantic alignment. Nevertheless, even this issue has recently been mitigated by extending the standard encoder-decoder architecture with an attention mechanism and reranking (**?**).

These models, however, still lack in several departments as far as the lexical correctness and the naturalness of the produced utterances is concerned. As **?** point out, the typical use of delexicalization of MRs and utterances can lead to poor sentence planning overall, which depends on the actual values. Moreover, the authors argue that the above end-to-end approaches might not be able to learn advanced language concepts, such as the aggregation of slots that share the same value in the utterance.

In our work, we develop an LSTM-based encoder-decoder language generation model, which we train on a large unaligned dataset in the restaurant domain, as described in **?**.

3 Data

In this project our primary dataset is defined by 50K instances of MR/utterance pairs in the restaurant domain (**?**). This is by far the largest dataset available for this task, that is also within the restaurant domain. Contrary to the data collection description provided in **?**, we noticed that there exist two additional slot types, namely *kidsFriendly* and *children-friendly*. We consider these slot types to function as alternatives to the *familyFriendly* slot type. By performing an exploratory data analysis, we notice that there exist 10 distinct

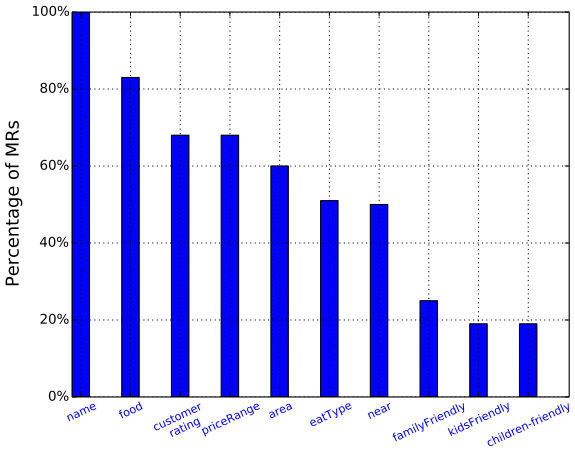

Figure 2: Distribution of slot types in the dataset.

Table 2: Average number of sentences in the reference utterance for a given number of slots in the corresponding MR.

Number of Slots	Number of Sentences
3	1.09
4	1.23
5	1.41
6	1.64
7	1.84
8	1.99

slot types with the most frequent being *name* and *food* (Fig. 2).

Analyzing the data further, we observe that the number of slots in each MR ranges between three and eight, whereas the majority of MRs consist of five and six slots (Fig. 3). Moreover, we notice that even if most of the MRs contain many slots, the majority of the corresponding utterances are expressed using one or two sentences (Table 2). Finally, a typical utterance is approximately 100 characters long.

3.1 Data Expansion

In our initial experiments, we only used the MRs with 3 slots to train our model. There were 13,787 such samples in total. While the entire training set consisted of 42,060 samples, it was still too small to produce reasonable results. We, therefore, expanded our training set by permuting the slot ordering in the MRs (**?**). So basically, we created 6 different orderings for every sample that had exactly 3 slots, 24 different orderings for every instance of 4 slots, and so on for the longer MRs. This, naturally, yielded an enormous

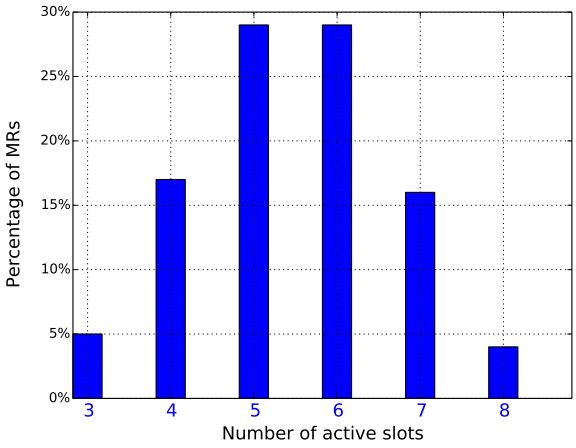

Figure 3: Percentage of MRs containing a given number of slots.

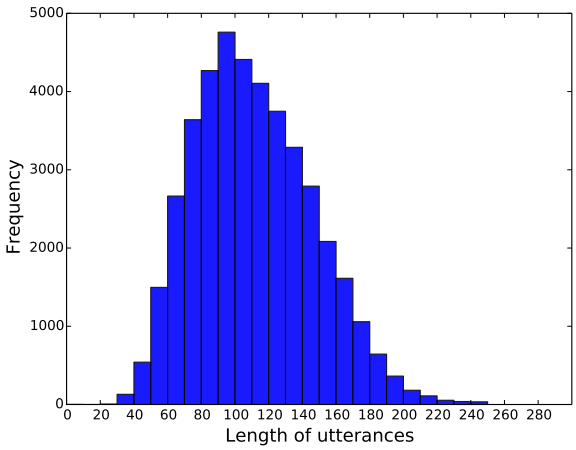

Figure 4: Number of characters in utterance.

dataset. Hence, while using all the permutations for instances of size 3 and 4, we sampled 100 random permutations for higher slot counts, i.e. from 5 to 8. We have thus created our augmented training dataset with 3,439,065 samples.

4 Methodology

4.1 Neural Model

The recurrent neural network (RNN) is a natural extension of the feed-forward neural network (NN) to be used with sequential input. In an RNN, the output at each time step directly affects the output at the next time step and indirectly at all future time steps. Therefore, given a sequence of inputs $(w_1, \ldots, w_n)$, the RNN computes a corresponding sequence $(u_1, \ldots, u_n)$ from the following pair of equations:

$$\mathbf{h_t} = \sigma(\mathbf{W^h h_{t-1}} + \mathbf{W^w w_t})$$

$$\mathbf{u_t} = \mathbf{W^u h_t}$$

Nevertheless, as we see from the formulation of RNNs, it is unclear how to map a sequence onto a sequence of a different size when the alignment of inputs and outputs is not known in advance. A common approach is discussed in **?**, where an RNN is used to map the input sequence to a vector of a fixed dimensionality, and in turn, that vector is mapped to the output sequence using another RNN. However, this approach suffers from the well known problem of the vanishing gradient in RNNs when trying to learn long dependencies (**?**). In order to tackle these problems, we focus on Long Short-Term Memory (LSTM) neural networks, which are better at learning dependencies over many time steps (**?**).

In this project we consider the state-of-the-art methods for natural language generation. We model this problem using LSTM networks in order to generate stylized responses that contain the natural variations of the human language. Following the approach suggested by **?**, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. We employ two LSTM networks (see Fig. 5): the *encoder* LSTM transforms the input sequence to a single fixed-length *context vector*, which forms an intermediate representation of the input. The *decoder* LSTM then infers the target sequence from the intermediate vector by peeking at it at every time step (using it thus as additional input information). In general, there exist several different flavors of LSTM networks, nevertheless, in this paper we compare the results obtained by two different architectures on the same generation task.

First, we explore the network suggested by **?**, which is governed by the following equations:

$$\mathbf{i_t} = \sigma(\mathbf{W_i w_t} + \mathbf{U_i h_{t-1}}) \quad (1)$$

$$\mathbf{f_t} = \sigma(\mathbf{W_f w_t} + \mathbf{U_f h_{t-1}}) \quad (2)$$

$$\mathbf{o_t} = \sigma(\mathbf{W_o w_t} + \mathbf{U_o h_{t-1}}) \quad (3)$$

$$\widetilde{\mathbf{c}}_\mathbf{t} = tanh(\mathbf{W_c w_t} + \mathbf{U_c h_{t-1}}) \quad (4)$$

$$\mathbf{r_t} = \sigma(\mathbf{W_r w_t} + \mathbf{U_r h_{t-1}}) \quad (5)$$

$$\mathbf{d_t} = \mathbf{r_t} \odot \mathbf{d_{t-1}} \quad (6)$$

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \widetilde{\mathbf{c}}_\mathbf{t} + tanh(\mathbf{W_d d_t}) \quad (7)$$

$$\mathbf{h_t} = \mathbf{o_t} \odot tanh(\mathbf{c_t}) \quad (8)$$

where $\mathbf{W}, \mathbf{U}$ are the parameters to be learned, $\mathbf{d_t}$ is a one-hot representation of the dialogue act, and $\mathbf{i_t}, \mathbf{f_t}, \mathbf{o_t}, \mathbf{r_t} \in [0,1]^n$ constitute the input, forget, output and read gates, respectively. Notice, that the equations that distinguish this model from standard LSTM flavors are (5)-(7). More specifically, the newly introduced control cell (5-6) affects the generation process by manipulating the dialogue act features, in order to make the produced utterance represent the intended meaning.

The aforementioned structure consists only of a single LSTM cell, nevertheless, the proposed architecture can be easily extended to a deep neural network by stacking multiple LSTM cells on top of each other. As explained in **?**, in order to allow hidden layers to affect the read gate, we simply have to update equation 5 to:

$$\mathbf{r_t} = \sigma(\mathbf{W_r w_t} + \sum_l \alpha_l \mathbf{U_r^l h_{t-1}^l}) \quad (9)$$

where l is the level of the LSTM layer and α_l is a layer-wise constant.

The second LSTM network that this paper explores is also based on the encoder-decoder architecture. This model utilizes equations (1-4) and (8), but updates the final memory using the more standard:

$$\mathbf{c_t} = \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \odot \widetilde{\mathbf{c}}_\mathbf{t}$$

Moreover, in order to force our network to establish longer temporal dependencies between the input and the output, we represent the encoder as a bidirectional LSTM network. An alternative approach that we attempted, was to reverse the order of the input sequence, as suggested by **?**. Nevertheless, through experimentation we noticed that the bidirectional encoder outperformed the latter solution.

In addition, using this LSTM architecture, we developed two models functioning on the word and the character level, respectively. In the word-level model we used pre-trained GloVe[1] word embeddings (with 300 dimensions), whereas for the character-level model we used a one-hot representation of each character. As expected, we have

[1] https://nlp.stanford.edu/projects/glove/

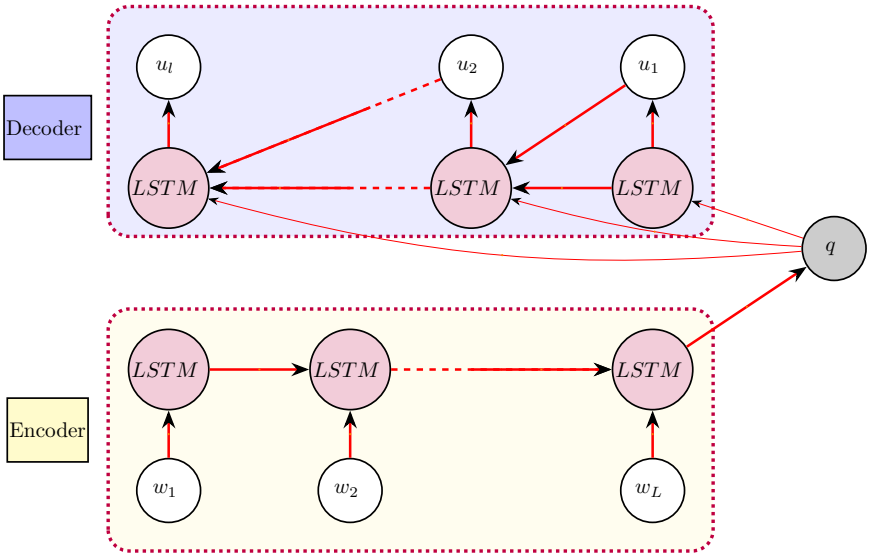

Figure 5: Standard encoder-decoder architecture. Input sequence $(w_1 \dots w_L)$ is mapped to a context vector q. Using q we produce the output sequence $(u_1 \dots u_l)$.

found that the word-level model significantly outperforms the character-level model, since to train the latter we would need significantly more data. The character-level model yielded completely incoherent utterances, failing to even produce single words. Since there was no n-gram overlap between the produced and target utterances, the resulting BLEU score was zero.

The goal of the both LSTM architectures is to compute the conditional probability $P(u_1, \dots, u_l | w_1, \dots, w_L)$ where $(w_1, \dots, w_L)$ is an MR encoded as a sequence of embedding vectors, and $(u_1, \dots, u_l)$ is a sequence of one-hot vectors representing an utterance. The dimensionality of these one-hot vectors equals the size of the vocabulary of the training and test set combined, which amounts to below 2,700. Notice that, in contrast to the vanilla RNN architecture presented earlier, the lengths of the input and the output sequences need not be the same.

4.2 Attention Mechanism

The encoder-decoder architecture described in the previous section has one primary weak spot, and that is the intermediate representation of the input sequence. It serves as the sole input to the decoder, based on which the decoder is expected to produce the target sequence. Encoding a potentially long sequence of dense vectors to a single vector inevitably leads to a substantial loss of information and, subsequently, to poor performance of the decoder.

In order to alleviate the burden on the encoder, **?** proposed using an *attention* mechanism, which allows the decoder to consider different parts of the input sequence at each time step. By providing the decoder with a sequence of context vectors $q_1, \dots, q_l$ instead of a single context vector, it can learn what specific parts of the input sequence to pay attention to, given the output generated so far.

In this extended model architecture, the probability of the output at each time step t of the decoder depends on a distinct context vector q_t in the following way:

$$P(u_t | u_1, \dots, u_{t-1}, \mathbf{w}) = g(u_{t-1}, s_t, q_t) \ ,$$

where s_t is a hidden state of the decoder RNN at time step t, computed as:

$$s_t = f(s_{t-1}, u_{t-1}, q_t) \ .$$

The context vector q_t is obtained as a weighted sum of all the hidden states $h_1, \dots, h_L$ of the encoder:

$$q_t = \sum_{i=1}^{L} \alpha_{ti} h_i \ ,$$

where α_{ti} corresponds to an alignment model parameterized as a feed-forward NN, jointly trained with the entire system. The learned weights indicate the level of influence of the individual words in the input sequence on the generation of the word at time step t of the decoder. The model thus learns a soft alignment between the source and the target sequences.

While the authors of the attention mechanism take the previous time step $t-1$ of the decoder into consideration when training the α_{ti} weights, we started off with a simpler model in which the weights depend on the current time step only. This implementation is clearly inferior and we plan on extending it in the future.

4.3 Delexicalization of MR Slots

In order to increase the performance of our system, we delexicalized our data to allow for more generic utterances, which has been shown to increase performance (**?**). We detected our categorical slots, "food", "name" and "near", and delexicalized them from the utterance. We selected these slots as they are most commonly seen entities and previous work has identified them as the most intuitive slots to delexicalize (**?**). In future work, we intend to experiment with delexicalizing additional combinations of slots. We have also included a relexicalizing tool which replaces our placeholders in the prediction with appropriate values in a post-processing step.

We have also experimented with explicit delexicalization of periods in the utterances, instead of ignoring them. This enables the model to learn to include periods in the utterances, in addition to words. Although it might later prove useful, currently the model produces utterances too short to be split into multiple sentences. Hence, the periods could never be found anywhere else other than at the end of the produced utterances.

4.4 Response Pooling and Error Correction

Empirically, we observed that different slot orderings resulted in better realizations within different contexts. To leverage this finding, we decided to randomly permute the slot ordering in the input MRs. Currently we create 3 variations of each input MR and feed all of these representations into the system to receive a pool of different realizations for the same MR. A point to be noted is that the goal of this permutation is different from our data expansion via permutation mentioned previously - here we are focused on creating variation in the input utterances of our test set while our previously mentioned permutation focused on expanding the size of our training data. Once our system has produced predictions, we perform scoring and reduce each pool to the highest scoring utterance. Equation 10 represents how we scored our utterances.

$$s(u_i) = info(u_i) - errors(u_i) \tag{10}$$

$$errors(u_i) = technical_{errors} - artifact_{errors} \tag{11}$$

We reward an utterance for being more informative. Our informativeness, $info$, is automatically estimated by a direct string overlap of slot values in the utterance. We do acknowledge that due to the variety of ways in which the system can generate the same MR value (e.g. 3 out of 5, 3-star, average), direct string overlap is not the most ideal automatic metric of informativeness. We predict that accounting for semantic overlap instead of direct string overlap is a better metric for automatically scoring the informativeness of an utterance, we leave this refinement to future work.

We penalize an utterance for the number of errors it contains. Errors can be either technical with respect to spelling, grammar, and syntax, or an artifact within our system. An artifact is something that is not detected as a technical error, but we know empirically to be characteristic of a bad utterance. An example of an artifact can be seen in the second utterance of Table 3 where the repeating digits (3 3; 5 5 5) are not technically wrong as they could represent real numerical data, however empirically we know that these recreating numbers within the given context are traits of a bad utterance. Our $technical$ errors are detected by language check[2], a Python wrapper for the open-source proof-reading utility LanguageTool[3] which returns a list of all the technical errors in a candidate utterance. Our $artifacts$ are easily detectable using homegrown regular expression matching.

Finally, we refine our utterance by automatically correcting these known errors using a combination of language check, which automatically performs appropriate corrections, and generic regular expressions substitution. We found that our utterances appeared to be significantly better after the pooling and subsequent correcting of the selected prediction. Table 3 demonstrates different cases in which the scoring and correction algorithms are used. The first and third utterance each have 2 of the 4 possible slot values, hence

[2]https://pypi.python.org/pypi/language-check
[3]https://languagetool.org/

Table 3: An example MR of "food=Fast food", name="The Golden Curry", near="city center", score="3 of 5" and * represents the correct utterance.

Utterance	info	lc	other	score
The Golden Curry serves Fast food food near near *The Golden Curry serves Fast food near	.5	.33	0	.17
The Golden Curry is rated a 3 3 of a of 5 5 5 *The Golden Curry is rated a 3 of an of 5	.25	.13	.4	-.28
The Golden Curry is near near the city centre *The Golden Curry is near the city centre	.5	.25	0	.25

an informativeness score of 0.5. The second utterance also represents 2 slot values, however, due to our current reliance on direct string overlap (mentioned previously), it only achieves an informativeness score of 0.25. Both the first and the third sentence have several technical errors, while the second sentence has both technical errors and artifacts. We notice that the third utterance *The Golden Curry is near the city centre* is scored highest and kept in our list of predictions while the other two utterances are pruned away. This example indicates that our scoring algorithm is valid as it has indeed picked the best possible utterance.

4.5 MR Splitting

As can be seen in the example from Table 3, our model can moderately well represent two-three slots from an MR. We hypothesize that by breaking longer MRs into multiple smaller MRs, we will be able to generate multiple utterances which when recombined will be more coherent and informative than attempting to directly generate a single long MR. Therefore, a final technique which we applied to our pipeline is MR splitting. In MR splitting we take long MRs with many slots and break them into smaller MRs which have at most 3 slots. More specifically, within our data an MR will include the name slot in addition to up to 2 other randomly selected slot values from the original MR. We observed in our training data that the name of the restaurant, in raw form or as a pronoun, was in each of the sentences within a given utterance, hence this slot is present in every combination.

Table 4: An example of the MR splitting results. Note, results also accounting for correcting mentioned in previous section.

MR	food[Fast food], name[The Golden Curry], near[city center], score[3 of 5]
MR-split	food[Fast food], name[The Golden Curry], near[city center]; name[The Golden Curry], score[3 of 5]
Utterance-split	U(The Golden Curry serves Fast food near.); U(The Golden Curry is rated a 3 of an of 5.)
Utterance	The Golden Curry serves Fast food near. It is rated a 3 of an of 5.

Once we've generated predictions for each of these smaller MRs, we do post-processing to recombine them into one utterance. When recombining sentences, we verify that we are correctly displaying punctuation and using pronouns for the instances of the restaurants name after it has been initially stated. An example displaying this process can be seen in Table 4.

4.6 Baseline System

For our baseline, we use a retrieval based method, which looks for the slots in the test MR in the training data MRs and replaces the slot values in the corresponding utterance with those in the test MR.

The results from the baseline system on the test set are quite good, and difficult to beat using our current approach. However, the baseline system fails to deliver in the cases where the matched MR slots in the training instance don't have the slot values in the generated utterance and instead have a synonym, or if they don't have an exact matching.

Some such examples are listed in Table 5.

These problems can be handled better by the introduction of rules, for example, a high rating is the same as a five-star rating, or a high price range is above $£30$, while a low rating and poor rating are one and the same. Also, using regular expressions to match possible patterns in the generated utterance, such as over $£30$, more than $£30$, greater than $£30$ instead of exact matching, can help.

A major shortcoming of the baseline system is that it is not capable of introducing any new kinds of variations in the generated utterances. Also, the system is restricted to the restaurant domain.

Table 5: Baseline System: Error Cases

1	MR	name[The Eagle], priceRange[more than £30], customer rating[low], area[city centre]
	Reference	'The Eagle' is located in the city center. It costs over 30 pounds to eat out there and it has low customer rating overall.
	Generated	In the city center, 'The Eagle' is a cheap restaurant with poor customer reviews.
2	MR	name[The Wrestlers], priceRange[low], customer rating[high]
	Reference	The Wrestlers is a low priced restaurant with a high customer rating overall.
	Generated	The Wrestlers is a low cost option with poor customer reviews.

5 Evaluation

In prior work, researchers have used both automatic and human evaluation for this NLG task. Automatic evaluation includes metrics from the subfields of machine translation and text summarization, such as BLEU, METEOR and ROUGE. These metrics are useful when we have reference utterances for the MRs and can compare the generated ones to them. We primarily report results in terms of the BLEU-4 metric (**?**) and METEOR (**?**).

Evaluating the LSTM model suggested by **?**, we were able to calculate only the BLEU score, since METEOR was not built into their existing codebase. Assessing this model on the E2E NLG challenge data, we obtained a BLEU score of 0.29. This was initially surprising, since, when the same system was tested on the restaurant dataset provided by the authors, it achieved a BLEU score of 0.75. Analyzing the restaurant benchmark dataset provided by **?**, we see that on an average, each MR contains only 2 slots, whereas in our dataset each MR contains approximately 5.5 slots. In order to test whether this difference in BLEU scores could be attributed to the fact that the two datasets contain many MRs with different numbers of slots, we restrict the training and the test set to contain only a fixed number of slots. In general, we see that the BLEU score increases as we increase the number of slots in both the training and the test data, but it is still far from the 0.75 that was obtained with their dataset. Nevertheless, we observe that when we restrict the number of slots, the system manages to perform significantly better (Fig. 6). Furthermore, the data **?** used for testing contains a large proportion of simple MRs, such as the goodbye-DAs, which are repetitive in the dataset and trivial to produce a reference-matching utterance for.

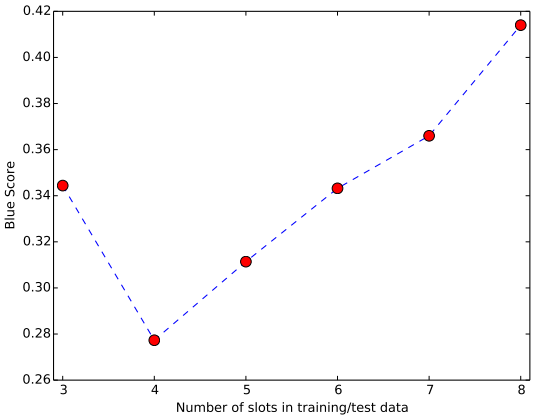

Figure 6: BLEU score obtained for **?** when restricting the number of slots in every MR while training and testing. Restricting the number of slots improves the performance of the system.

Table 6 shows the BLEU scores we obtained as we varied the complexity of our model. We see that the BLEU score achieved by the most complex model (Model 4) is not the highest, whereas this model manages to perform a lot better when we consider the METEOR metric.

Table 6: Training various models for 20 epochs. D, P, R, C, S stand for Delexicalization, Pooling, Reranking, Corrections and MR splitting, respectively.

Model	D	P, R, C	S	BLEU	METEOR
1	—	—	—	0.04	0.14
2	✓	—	—	0.14	0.17
3	✓	✓	—	0.15	0.13
4	✓	✓	✓	0.11	0.20
Baseline	—	—	—	0.12	0.28

6 Conclusion and Future Work

In this paper we have taken promising first steps in creating an end-to-end pipeline, which is capable of generating natural language utterances from their respective meaning representations. We have implemented an encoder-decoder LSTM model and augmented its capabilities with a variety of different post-processing steps. Our model is capable of generating reasonable utterances, but

there are still margins for improvement. In future work, we plan to make our model more robust by exploring the inclusion of an attention mechanism in the LSTM architecture.

Our various extra processing steps had a clear improvement on the overall output of the system. In subsequent models, we plan to explore even more techniques that will allow our model to yield a better performance. Specifically, we are interested in translating the entire training utterance into an appropriate slot value representation. We hypothesize this technique will result in better transitions between slot values from a given meaning representation and work as an automatic method for aligning our currently unaligned data. Creating this semantic alignment is an interesting task in itself and has many advantages, such as allowing for a more accurate metric to automatically rate the informativeness of an utterance. We also plan on incorporating beam search to our decoder in order to efficiently identify the most probable output sequences.

While the dataset used in this project is the largest of its kind, we felt it was best to still do a massive data expansion via permutation of the original data. In the future, we predict it will be advantageous to also synthetically generate data which varies significantly from the original training data. We also strove to create a system which is as generic as possible. Currently we have worked primarily within the restaurant domain. Future work will attempt testing our model within different domains on different datasets to verify its generalizability. Also, we believe that BLEU is not an efficient metric for this task since BLEU uses exact matching of words and does not account for semantic equivalence/similarity.

Finally, once we incorporate all the necessary improvements, we will employ human evaluation in order to assess our model. Human evaluation can take into account more qualitative metrics such as informativeness (the extent to which the utterance contains all the information specified in the dialogue act) and naturalness (how close the generated response is to an actual human utterance), giving a more realistic estimation of the capabilities of our system.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Simard Bengio and Frasconi. 1994. Learning long term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5(2):157-166* .

Merrienboer Cho, Bougares Gulcehre, and Bengio Schwenk. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* .

Ondřej Dušek and Filip Jurčíček. 2016. Sequence-to-sequence generation for spoken dialogue via deep syntax trees and strings. *arXiv preprint arXiv:1606.05491* .

Schmidhuber Hochreiter. 1997. Long short-term memory. *Neural Computation* .

Ioannis Konstas and Mirella Lapata. 2013. A global model for concept-to-text generation. *J. Artif. Intell. Res.(JAIR)* 48:305–346.

Gerasimos Lampouras and Andreas Vlachos. 2016. Imitation learning for language generation from unaligned data. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. The COLING 2016 Organizing Committee, pages 1101–1112.

Irene Langkilde and Kevin Knight. 1998. Generation that exploits corpus-based statistical knowledge. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*. Association for Computational Linguistics, pages 704–710.

Alon Lavie and Abhaya Agarwal. 2007. Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments. In *Proceedings of the Second Workshop on Statistical Machine Translation*. Association for Computational Linguistics, Stroudsburg, PA, USA, StatMT '07, pages 228–231. http://dl.acm.org/citation.cfm?id=1626355.1626389.

François Mairesse, Milica Gašić, Filip Jurčíček, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. 2010. Phrase-based statistical language generation using graphical models and active learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, pages 1552–1561.

François Mairesse and Steve Young. 2014. Stochastic language generation in dialogue using factored language models. *Computational Linguistics* .

Hongyuan Mei, Mohit Bansal, and Matthew R Walter. 2015. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. *arXiv preprint arXiv:1509.00838* .

Neha Nayak, Dilek Hakkani-Tur, Marilyn Walker, and Larry Heck. 2017. To plan or not to plan? discourse planning in slot-value informed sequence to sequence models for language generation.

Jekaterina Novikova, Ondřej Dušek, and Verena Rieser. 2017. The E2E NLG shared task .

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '02, pages 311–318. https://doi.org/10.3115/1073083.1073135.

Verena Rieser and Oliver Lemon. 2010. Natural language generation as planning under uncertainty for spoken dialogue systems. In *Empirical methods in natural language generation*, Springer, pages 105–120.

Amanda Stent, Rashmi Prasad, and Marilyn Walker. 2004. Trainable sentence planning for complex information presentation in spoken dialog systems. In *Proceedings of the 42nd annual meeting on association for computational linguistics*. Association for Computational Linguistics, page 79.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *CoRR* abs/1409.3215. http://arxiv.org/abs/1409.3215.

Marilyn A Walker, Amanda Stent, François Mairesse, and Rashmi Prasad. 2007. Individual and domain adaptation in sentence planning for dialogue. *Journal of Artificial Intelligence Research* 30:413–456.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745* .