Natural Language Generation of *The Office* Using a Neural Machine Translation Context

Samuel Kwong
Department of Computer Science
Stanford University
Stanford, CA 94305
samkwong@cs.stanford.edu

Winton Yee
Department of Computer Science
Stanford University
Stanford, CA 94305
winton@cs.stanford.edu

Abstract

Advances in deep learning and neural networks in recent years have led to impressive advancements in text generation. However, text generation by neural networks is still often incomprehensible to a human reader beyond a single sentence. This problem is especially noticeable in film dialogue scripts. In a script, each line of dialogue is made by a particular character, with the next line of dialogue often being a response to that speaker from another speaker. Our goal in this paper is to improve the flow of exchange between characters in the script dialogue, namely, when exactly characters choose to speak and what they say in response to each other. We propose a natural language generation model using a neural machine translation context. Words spoken to a certain character in a script are treated as source sentences, and their responses are target sentences. Using this unique model, we are able to significantly improve on coherence and humor when compared to Recurrent Neural Network language models.

1 Introduction

Natural language generation is currently a very difficult problem to model. Human language is extremely complex, and there are many considerations to account for when trying to simulate the way a human talks. It particularly difficult to simulate how humans speak to each other. Different humans speak in different ways, so there is no one model that will be able to perfectly replicate the way all humans speak. Human dialogue also has many different aspects and features: a sentence may contain information about a context and a main subject that is subtle and hard to track. When trying to generate language, this is a problem, since it is difficult to capture the entire meaning of the previously generated text when generating new, relevant text. Language models have been used often in the realm of natural language generation, as they attempt to predict the probability of the next word of the text, based on previous words. In this paper, we explore generating new text based on scripts of the U.S. television show, *The Office*. For our baseline tests, we use simple word-based and character-based RNN language models. We found that these models often failed to capture the general sentiment of the previous text, and thus would just continually generate nonsensical text unrelated to what had come before.

For this reason, we decided to attempt a new approach: use neural machine translation for our natural language generation task. For this, we adapted the framework of translation to fit our task. We modified our dataset such that words being spoken to a character would be treated as the source sentence, and the words that a character says in response are treated as the target sentence. For example, imagine that in the script, Jim says to Pam, "Hi Pam," and Pam responds, "Hi Jim." We then treat "Hi Pam" as the *Spoken to Pam* language and "Hi Jim" as the *Pam* language. Afterward, we set up neural machine translation networks for each main character in the show, to translate to that character's "language." We hope that neural machine translation would help us in many areas

where a simple language model could not. In neural machine translation, it is possible to capture "many-to-one" and "one-to-many" alignments using attention, and this feature could simulate certain words triggering others in response. For example, if someone said "Pam" to Jim, that might trigger a long sentence by Jim, in a "one-to-many" alignment. Or, a sentence describing hatred may trigger Michael to say "Toby," in a many-to-one alignment. In a similar vein, by passing the same sentence across many different "languages" (characters), we hoped that we would be able to represent some kind of long-range dependency simulating a long conversation about the same topic.

In addition to our neural machine translation models, we also used a language model to choose the sequence in which characters speak. This model would take in a sequence of characters who had spoken previously, and output a prediction for who should speak next. For example, if the previous speakers were "Dwight," "Jim," and "Dwight," our language model may recommend that "Jim" speak next, simulating a back-and-forth conversation.

In our research, we found that a few others also had come up with the idea of generation-as-translation before us, but their end goals were in a different scope. Ehsan *et al.* use translation for generation, but motivate it as rationalization of an agent's actions, rather than purely generation for coherence's sake. Ferreira *et al.* also use translation as generation, but their "source" language consists of Abstract Meaning Representations, rather than other generated sentences themselves. With our model, we are unique in proposing that we chain together translations of the "same" sentence to create a coherent dialogue across the text.

In summary, our natural language generation model is an end-to-end network using neural machine translation models. A language model determines the next speaker, based on the previous speakers. We then take the previous sentence (the "translation" output into some character's "language") and translate it into our chosen speaker's "language." We repeat this process until we have generated our dialogue.

2 Related Work

Work in the area of natural language generation is extremely diverse. We used two models as our baseline: PyTorch's default word-based language model [1] and Andrei Kaparthy's character-based Recurrent Neural Network [2]. These two models used very simple Recurrent Neural Network structures to model language and served as our jump-off point for our own model.

Ehsan *et al.*'s paper on rationalizing internal states of machine agents to natural language using neural machine translation interested us due to the similarity of their proposed idea to our own [4]. In their paper, they used this context to get a machine agent to explain why it performs a certain way while playing a game of *Frogger*. Their training corpus is entirely human-generated. Humans complete an agent's task while writing down their reasons for acting that way. They then use an encoder-decoder Long-Short Term Memory RNN to encode the internal state representations of the agent, and decode to natural language. For evaluation, they asked humans to evaluate how satisfied they were rationalizing agent's motivations and explanations compared to baseline models that only stated utility or actions. They found that humans far preferred the rationalizing agent's explanations compared to the baseline models.

As stated before, this paper is similar to our own idea in that it is tackling the problem of NLG in an NMT context. This validated to us that our approach may work as well. However, we innovate in that our work is same-language to same-language translation. This paper's translation is more obvious: from an internal state to human language. Our translation, however, occurs on a much more abstract level: from person-to-person.

Ferreria *et al.* similarly try NLG using machine translation [5]. In their paper, they attempt to produce natural language from Abstract Meaning Representations (AMRs). AMRs represent sentences in the form a graph, where nodes represent concepts and edges represent relationships. For their translation models, they use a simple phrase-based Bayesian model and a basic GRU RNN. Interestingly, they found that their phrase-based model often outperformed the neural translator among different evaluation metrics (BLEU, METEOR, and TER). Ferreria *et al.*'s paper also validated that translation may be useful for producing text.

Once again, we found that our contribution would be quite unique. In both the Ehsan *et al.* and Ferreria *et al.*, both add another layer of abstraction to their source sentences in translation. Thus, their

source sentences in fact look different and have a different grammar and structure when compared to their target sentences, similar to how one language is very different when compared to another language. However, for our model, we have no extra abstraction between our source and target sentences. They are both in English, and we are trying to "translate" between one person's speech and another's. The translation we are doing is less literal: it is not from one representation to another, but from one sentiment to another.

3 Approach

3.1 Baseline

Our baselines include a word-based RNN and character-based RNN.

3.1.1 Word-based RNN

We used a word-based RNN from the PyTorch example library as our first baseline model [1]. It first embeds words using an encoder. Then, it feeds it into a multi-layer LSTM RNN, GRU RNN, or Elman RNN with $tanh$ or $ReLU$ non-linearity. For our baseline experiments, we used only the LSTM RNN and GRU RNN since we believed they would give us the best results. Finally, a decoder decodes the output of the RNN. This trains the model to predict the next word of the script. Below is an example of a character-based RNN. Notice how it is incoherent:

```
Oscar: Can I make you a day off between us? [Michael sets over]
Andy: I'm gonna have some attention. They good? One, two, he Halpert is that
stuff,  and it's been good enough to be earbud] Plus I have a couple of little
meet, a few office.
Michael: Toby sounds good.
Pam: [leaving the cigarette approaches from Michael's desk] Hey!
```

3.1.2 Character-based RNN

Additionally, we used a character-based RNN referencing Andrej Kaparthy's implementation [2]. This model first encodes each character into a one-hot vector and then feeds each encoding into the RNN on each step function call. On each step call, the RNN assigns confidence to whichever character may come next, based on the training data. Through parameter updates over time, the weights change in the direction of the gradient. During testing, every time a character is fed into the RNN, we obtain a probabilistic distribution of what characters are most likely to appear next, and this is how we repeatedly sample new characters to generate text. Below is an example of a character-based RNN. Notice how, like the word-based RNN, it is incoherent:

```
Jim: I have ever stolen.
Dwight: Oh, okay. Yeah. Get off the sy-ceat and then you guys do it!
Jim: Mmm hmm.
Erin: [laughs up] Frankling this thing. Okay?' [Pam jumps] Great, my carsons.
Andy: [on a stap fake desk robust may] Air... Sign to get out of the real.
```

3.2 Our Model

While our baseline models effectively generate new text, for a dialogue-based script, the output would be nonsensical to the audience. This is because characters seem to not be talking to each other but rather spitting out arbitrary phrases. These phrases lack any relation to what was said previously in the conversation, and overall, the characters fail to contribute to the flow of the conversation. Thus, we face two challenges: 1) having characters talk one-by-one in a sensible order (there should be at least some back-and-forth exchange i.e. Dwight and Jim talking to each other, instead of only instances like Dwight, then Jim, then Pam, then Michael, then Ryan, and so forth) and 2) having characters speak in a style they would normally speak in the show. We address these issues by exploring a novel approach with multiple parts.

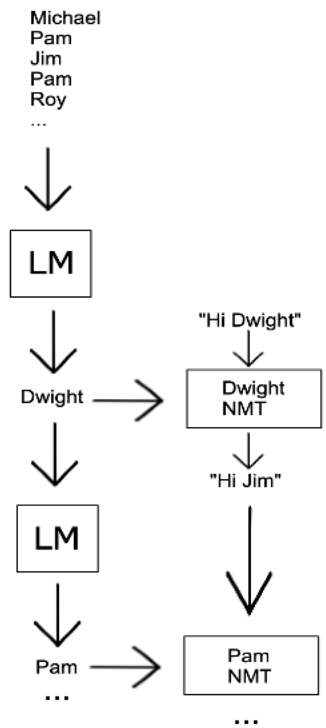

Figure 1: Architecture of our model.

The model we use is an amalgam of models: one model for determining which character speaks next and twenty-five separate models trained specifically for each of the twenty-five characters in our dataset. The leading motivation to use an individual model for each character is to ensure that our system generates text relevant to each character. As a result, our overarching model is trained end-to-end. We first train a word-based RNN language model that generates who is the next speaker for each line of dialogue, treating each speaker's name as an entry in our vocabulary. Then through a novel approach, we explore using a form of NMT to generate the dialogue text of one speaker after another. Each NMT model uses a character-based convolutional encoder to derive the word embeddings for our raw input text from the particular speaker's line of dialogue. Then an RNN encodes the line of dialogue as a sequence of encoder hidden states. Another RNN produces decoder hidden states by operating over a target sentence and using attention on the encoder hidden states. Finally, a softmax linear layer informs our model the most likely target word for each decoder timestep. The twist is that, in training time, each model's source sentence is the previous speaker's line of dialogue, while the target sentence is the line of dialogue the current speaker has said in the dataset. Thus, when it comes time to generate output, each model's generation of text is dependent on what the previous speaker said. In effect, every line of dialogue is a translation from one line to the next, supplying a flow of exchange between speakers previously rare in other NLG models.

In our architecture, we make additional adjustments to the NMT models to effectively generate interesting text. With a vanilla NMT translation model, a sample source and target is shown below:

```
Dwight: Question. Will you be seeing Jan when you're in New York?
Michael: No, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no,
no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no,
no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no, no,
no, no, no, no, no, no, no, no, no, no, no, no, no, no, no.
```

In fact, for the vanilla NMT, most of a single character's responses would be the sample phrases repeated over and over. For example, many of Michael's target translations were a long string of

"no"'s. This is due to the fact that the NMT model is trying to predict the best possible translation. Since it is not easy for the model to see how one question might translate to a response, it defaults to simply predicting the most popular words to try and minimize its loss. To account for this, we introduced a random sampling based on a probability distribution. We applied a softmax function to the scores of translation hypotheses, and then randomly selected one based on the probabilities.

After this step, we found that our generated text would still always be the maximum possible length. This was because model was still trying to minimize its loss by putting all words it thought might be included in the target sentence. To avoid this, we automatically added an end of sentence token to the text if it contained a period, question mark, or exclamation mark. With these changes, we are able to produce much more realistic responses. A sample source and target from our final NMT translation model is shown below:

```
Dwight: Question. Will you be seeing Jan when you're in New York?
Michael: No, you know what?
```

4 Experiments

4.1 Data

We found our dataset on Reddit, posted by u/misunderstoodpoetry [3]. It contains every line from every episode of the U.S. television show "The Office," 59909 lines in total. Lines include the speaker and stage directions. As the data was originally in CSV format, we wrote a script to convert it to text files. We had two versions of the for two different use cases, the translation task and the language modeling task. The translation text files contained all the information, including the speaker of a line and the line itself. The language modeling text files contained only the order of names in which characters spoke, since we just wanted to use the language model to predict the next speaker.

4.2 Evaluation Metric

We use perplexity (PPL) to evaluate the sample scripts generated by our baseline models, as in common in language modeling tasks. Perplexity measures how well our model is able to predict the next word of the text. Formally, perplexity is defined as follows:

$$\prod_{t=1}^{T}\left(\frac{1}{P_{LM}\left(x^{(t+1)}|x^{(t)},...,x^{(1)}\right)}\right)^{\frac{1}{T}} \tag{1}$$

where T is the length in words of our predicted text. We also used perplexity to evaluate our own language model for generating sequences of speaking characters.

For the NMT part of our model, though, we decided that perplexity was not a good measure of our objective. Our task is to generate completely new text, not simply match exactly the gold standard text. Reiter [6] states that "Metrics are of interest if they can approximate or predict the above kinds of hypotheses. In other words, we ultimately care about whether an NLG system produces high-quality texts, not what its BLEU score is. BLEU (etc) scores are only valuable if they can reliably predict the result of testing the hypotheses we care about (utility, etc)." Perplexity is still useful for comparing our baseline language models to each other, since they have similar structures in architecture, and we could empirically compare how good they were at predicting the next word. When comparing the generated text of our own model to the baselines, however, we decided to evaluate based on human judgment. We found that perplexity was not ultimately a good indicator of what was a "high-quality" text.

However, there are also other details of our generated scripts that we want to be able to evaluate, such as humor. We plan to also use human judgment to evaluate scripts on both humor and coherence. Humor is a measure of how funny the script is. Coherence is a measure of the overall structure of the script (how much it "makes sense"). In each of these categories, humans will evaluate the scripts on a scale of 1 to 5.

4.3 Experimental Details

After hyperparameter tuning, we selected the best hyperparameters for the baselines and our own model. The following are the best models we found in our experiments:

1. Character-based LSTM RNN, with 3 layers
2. Word-based LSTM RNN, with 2 layers, embeddding size 650, hidden units per layer 650
3. NMT Language Generation

4.4 Results

Table 1: Perplexity Scores of RNN Models

Model	Score
Char-based	104.45
Word-based	251.76

No matter the hyperparameters, our results with the baseline models leave much room for improvement. The models are able to output scripts, but they are mostly nonsensical. There are good and bad qualities in both character-based and word-based RNNs. Character-based RNNs are able to easily close brackets, but often do not produce real English words. On the other hand, word-based RNNs only produce English output, but the grammatical structure is often poor. Additionally, there is no relationship between what one person says and what the next person says; the lines are all mostly individual. The sequential nature of RNNs makes it difficult for us to model long range dependencies.

For our own model, we individually trained one NMT model for each of 25 character from *The Office*. We found that the perplexity of these models were extremely high. We include the perplexity for the models of the top three most frequent characters below:

Table 2: Perplexity Scores of NMT Models

Character	Score
Michael	2628.17
Dwight	23525.16
Jim	2628.17

However, as we stated above, perplexity is not the best measurement for NLG, and not our main text evaluation metric. In our case, the reason the NMT models have such a high perplexity is due to the fact we introduced sampling based on a probability distribution during translation, rather than always selecting the most likely word. As stated above, we did this so our NMT models would produce varied and interesting text, rather than always trying to produce the most "correct" translation.

Our model has two parts: the language model for generating sequences of characters and the NMT models. Our language model proved to simulate sequences of characters well. Our final language model for sequences had a perplexity score of 3.49, and in our text, we often saw the model would output the same two characters going back a forth, simulating a dialogue between two people.

As stated above, we evaluated the NMT models (and our model and its generated text as a whole) using human judgment. We received 26 human evaluations each for the text generated by the char-based model, the word-based model, and our NMT generation model, in the domains of coherence and humor. Our results are shown below, as well as evaluation of actual scenes from *The Office* for comparison:

We found that humans found our scripts to be significantly more coherent and humorous than our baseline models. While the baseline models seemed to only output nonsensical text, our model was able to generate text that was coherent (except for a few grammatical mistakes) across multiple lines of dialogue. Additionally, we found that our model was able to improve on the humor of the baseline model as well, due to the fact that people were actually able to understand what they were reading better. Thus, humor was easier to find in the readable text.

Table 3: Huamn Evaluation Score of Models

Model	Coherence	Humor
Char-based	1.31	1.27
Word-based	1.23	1.15
NMT-language	3.27	3.08
Actual scenes	5.00	4.12

Below are some examples of text generated by our NMT natural language generation model.

Example 1

```
Dwight: I don't think you can do this to you down.
Jim: I'm going to you?
Dwight: I can do my aphrodisiac.
Jim: I think you should have to go back to her.
Dwight: I don't want to be standing to work?
Jim: I'm not gonna do it.
Dwight: I can get it. You know what? You just want to be a bad or time you
ever learned off how you were private salivating that Hold a witness.
Jim: I'm going to coffee.
```

Example 2

```
Jim: I think you should have a second?
Ryan: Did you get a drink of a thing.
Pam: I don't think you should go it.
Michael: Excellent!
Pam: I mean, I don't think we should have a second?
Michael: Mmm.
Pam: You don't have cash.
Michael: No, no. You know what, that is that?
Jim: No.
Michael: Definitely.
```

5 Analysis

Our model is successful in many ways. We found that, for one, it was able to much better capture long-range dependencies within a dialogue, mentioning things that had been brought up a few lines ago. One instance of this is in our second example, above. A "second drink" is mentioned by Pam and Jim. This is due to the nature of our NMT model. Our model essentially translates the "same" sentence over and over again, into the languages of different characters. Because of this, some encoding of the sentence that continually gets passed along through the models may mean some certain thing to a specific model. Thus, when that encoding reaches another model, it may trigger a similar response a previous one.

Additionally, our model is good at capturing general sentence structure. Questions are followed by answers, and dialogues between two individuals naturally occur. The word-language model is good for producing these dialogues, since it picks up from the training data that two people often speak back and forth to each other. Attention and alignment from the NMT models may also help for producing quality responses. They may notice that source questions are often translated to target answers.

One area for improvement is the general grammar of our generated text. The text loosely resembles English grammar, but it is far from perfect. This poor grammar may be a byproduct of the randomness introduced during the translation phase. Longer training, or an addition of a module to regularize grammar may help in this aspect. We additionally noticed a concern that we were just copying n-grams from the original scripts. However, analysis showed that we never copied anything longer than a 2-gram, validating that our text was indeed new.

6 Conclusion

In this paper, we explored the intersection of natural language generation and neural machine translation in producing plausible *The Office* scripts, where dialogue seems to have flow of exchange similar to real *The Office* scripts. From initial baseline tests, we noticed that word-based and character-based RNNs, although already quite effective at text generation, are rather poor at producing dialogues for film scripts.

We have shown through the results of our multiple-step model, however, that our proposed architecture noticeably outperforms previous works in generating coherent dialogue scripts. We were able to attain more believable text conversations, in the style of their respective speakers, by addressing the main issue of conversation: that speakers must indeed *speak* to each other. Producing a whole block of text for story generation is one task. But producing back-and-forth dialogue is another, with its own challenges being supporting a flow of exchange between speakers. Overall, it is key to for the model to attain a sense of understanding of who exactly the model is responding to and what the current topic matter is.

Acknowledgments

Thanks to Kim *et al.* for NMT model reference [7].

Thanks to our mentor Anand Dhoot for his guidance.

References

[1] Pytorch. (2019). Pytorch Examples. [online] Available at: `https://github.com/pytorch/examples/tree/master/word_language_model`

[2] Kaparthy, Andrej. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. [online] Available at: `karpathy.github.io/2015/05/21/rnn-effectiveness/`

[3] Every Line From Every Episode of "The Office." [online] Avaiable at `https://www.reddit.com/r/datasets/comments/6yt3og/every_line_from_every_episode_of_the_office_us/`

[4] Ehsan, U., Harrison, B., Chan, L., and Riedl, M. (2017). Rationalization: A Neural Machine Translation Approach to Generating Natural Language Explanations. [online] Available at `https://arxiv.org/pdf/1702.07826.pdf`

[5] Ferreira1, T., Calixto I., Wubben, S., Krahmer E. (2017). Linguistic realisation as machine translation: Comparing different MT models for AMR-to-text generation. [online] Avaiable at `https://pdfs.semanticscholar.org/60e6/cf2f76da04dc71e3174079972da070a3722e.pdf?_ga=2.42208869.1509382020.1552848569-2145297773.1552848569`

[6] Reiter, E. (2017). How to do an NLG Evaluation: Metrics. [online] Available at `https://ehudreiter.com/2017/05/03/metrics-nlg-evaluation/`

[7] Kim, Yoon. (2016). Character-Aware Neural Language Models. [online] Available at `https://arxiv.org/pdf/1508.06615.pdf`