Text Generation with Recurrent Neural Networks

Andrew Brusso
Computer Science
Michigan Technological University
Houghton, United States

***Abstract*—A research project exploring techniques for using Recurrent Neural Networks for the problem of Natural Language Generation, a key problem in ongoing cutting edge Natural Language Processing research.**

***Index Terms*—text generation, natural language processing, machine learning, supervised learning, recurrent neural networks**

I. Background

The problem of Natural Language Generation (NLG) is by no means new, despite the most visible applications such as autocompletion and virtual assistants only appearing in the past decade. Some trace the history of the more general domain of Natural Language Processing (NLP) as far back as renaissance philosophers such as Descartes and Leibniz in the seventeenth century, who were proposing early ideas for machine translation between different languages using encodings. Others trace the history back to Alan Turing's paper on Computing Machinery and Intelligence [13], which proposed a way of testing the effectiveness of a NLG technique by pitting it against a human participant.

In a practical sense, the first actual application of NLP is probably the Georgetown experiment in 1954, which proposed a technique for translating a number of Russian sentences into English. The experimenters made the bold claim that in "five, perhaps three, years hence, interlingual meaning conversion by electronic process in important functional areas of several languages may well be an accomplished fact" [5]. The first chatbot in 1964, ELIZA, might be considered the first true application of NLG. These early techniques in Natural Language Processing and Generation were almost entirely rule or template based approaches, and this remained the case well into the early 1980s. A rule would be written for how to respond to text in a particular format, and would be used to map an input text to a reasonable output text. For example a rule might look like "I am interested in *" and the rule might map to an output that reuses the wildcard such as "Yes, * is very interesting". The complexity of rules and templates improved considerably over time, often in parallel to improvements in compiler and language theory which represented similar problem approaches. In the 1980s, early machine learning concepts were being proposed and applied in limited scopes using probabilistic approaches (such as n-gram models), but the problem of general generation and translation remained elusive into the early 2000s. Probablistic models were improving, but they were often heavily supervised, and mostly used handwritten text with machine learning backing the selection process of the response or result text, rather than generating entirely novel text.

A short publication by Mikolov et. al [6] in 2010 pointed towards the potential of Recurrent Neural Networks (RNNs) for NLP problems, with promising results compared to existing techniques that focused on n-gram sequences. The first breakthrough into more novel text generation happened in 2011 with a landmark paper by Sutskever, Martens, and Hinton [8] that proposed using RNNs to learn how to generate text without hand written rules. As this idea expanded further, it developed alongside advances in RNNs, first applying the concept of Long Short-Term Memory (LSTM) and then Gated Recurrent Units (GRU) to deal with issues of exploding and vanishing gradients while training the network via Backpropogation Through time (BPTT). This advancement showed the initial potential of the approach by showing that RNNs with LSTMs could capture complex aspects of sentence structure and grammar (although poorly initially). Alex Graves in 2013 showed with the explosion of deep learning, that stacking more hidden layers combined with memory units, and using larger datasets, could generate surprisingly good structure through simple approaches [3]. Shortly after, Sutskever, Vinyals and Le [9] were working on the problem of machine translation and proposed the Seq2Seq architecture as a powerful deep learning tool to capture context in generative models, with promising results that soon made their way into google translation architectures. An explosion of different approaches and results soon occurred focused on improving contextual awareness [10], dialog systems [14], emotional components [7], and more aspects of generated text. Open AI released the first commercial all purpose text generator, GPT-2 in 2019, and with it threw down the gauntlet. Google's Deepmind team responded in kind in January 2020 with their conversational agent Meena [1], and with that the NLP field has become more lucrative than ever.

II. My Research

A. Getting a basic model

For my research, I wanted to explore the concept of using RNNs for text generation, but I also wanted to be realistic with what I would be able to accomplish in the limited time I would have to work on the project. Realistically, I didn't expect to do something completely novel, or to have results comparable to the current cutting edge approaches, because there are teams of people with significantly more experience and time working on them. What I did want to accomplish was learning more about

the problem of text generation, what some of the historical solutions to the problem looked like, what some of the more modern solutions looked liked, and the direction that the cutting edge research is moving in. Instead of jumping straight into trying to build a Seq2Seq architecture of my own, and likely struggling with it the whole term, I decided to start with a more simple RNN that I know I could feasibly implement, and then work my way towards understanding some of the nuances of more complicated Seq2Seq architectures.

I started things off by reading and understanding the original 1997 paper that proposed using an LSTM to address the vanishing/exploding gradient problem [4], as well as the foundational RNN papers [8] [3] for modern NLG with deep learning. With those papers in mind, I started off building a basis using a tutorial on text generation with Tensorflow [2], to build what I will refer to as my *basic* model. This tutorial utilizes a text generation approach based on a many to one RNN architecture, where the input of the model is a sequence (in the article the sequence length is 100 characters), and the output is a single character predicted to occur as the next character in the sequence. The method in this tutorial does work, but I found that the performance I was getting with my output didn't seem to align very well with the results they showed–my end results were very repetitive, and lacked significant structure. With this tutorial, it was also taking a very long time to train the network (I was training overnight and finding it still running in the morning). I eventually followed the Tensorflow guide for using GPU acceleration [11], to instead train on my GPU. This did speed things up quite a bit, but I was still running into memory bottlenecks. With this model in my back pocket, I started branching out to find other approaches that might work better.

B. Making an Improved Model

After having a model that sort of worked, I happened across the official Tensorflow article on text generation [12], which I utilized to augment my network to try and address some of the key issues I was seeing, and create what I will refer to as the *improved* network. There are four significant changes from this article that I implemented into my network, which caused valuable improvements. The first was using the Tensorflow Dataset class, which does a much better job of memory management during training than the basic model, because it buffers the amount of the dataset that is kept in memory to make sure it doesn't max out the memory of the system. The second change had to do with the method of inputting and outputting the training sequences into and out of the network, which the basic model was doing with a technique called one-hot encoding. With one-hot encoding, every character of the training data is encoded into a sparse vector, where every value in the vector is zero, except the specific index corresponding to the particular character being encoded, which has a value of one. Because of the sparseness of these vectors, the number of parameters the network needs to learn is forced artificially high. The Tensorflow article introduced me to the concept of using an embedding layer, where instead of explicitly encoding the vectors, the network is trained to learn a significantly more efficient representation automatically, which reduces the overall dimensionality of what the network learns (and also has the advantage of grouping characters that are more similar together). The third change, which was the most important with respect to the results, was using a temperature adjustment to inject randomness into the network output. Initially, the selection of the output character of the network was being done using the argmax function, which takes the networks softmax output, and blindly chooses the character that the model is most confident in. It may seem counter intuitive, but always choosing the value the model is most confident about is what causes looping behavior, because once an input sequence starts generating the same output as it is receiving as input, the output makes its way back into the input and eventually creates cyclical text. The idea with temperature is that we sample from a probability distribution based on the network's confidence, so that there is some probability that other less confident values can show up in the output. The temperature itself is used to bias the probability distribution that is sampled from, where small temperature values bias the distribution more towards argmax-like behavior, and large values bias it closer towards a uniform distribution. The fourth change is that the basic model treated the problem as a many to one sequence problem, while the Tensorflow article treats it as a many to many sequence problem. In the article, instead of the output only being the single desired character to be predicted, it shifts the entire sequence forward a character, and tries to train so that it predicts the next character for every single character in the sequence. This is only used for the training, and then for actually running the network for generation only a single character is predicted from the network at a time, but this training approach seems to significantly strengthen the connections that the model ends up learning.

	Characters	Words
Facebook	519,898	93,526
Discord	780,503	137,887
Essays	259,593	44,447
Total	**1,559,994**	**275,860**

Fig. 1. The distribution of characters and words in the dataset that I generated

C. Gathering a dataset

Up until this point, I was using the works of Shakespeare and Alice in Wonderland [1] as my data sources for training and testing the network. This was when I started gathering data from my own data sources, to build out a dataset that I felt could be representative of my own writing style. There was three data sources I pulled textual data from: Discord, Facebook Messenger, and a collection of school essays and

[1]I think it is funny that Alice in Wonderland shows up often in the NLG literature, because if your model generates nonsense output you can always fall back on claiming it is because it was trained on nonsense (as Alice in Wonderland is classified as the foremost example of the genre literary nonsense).

reports I have written. For discord, I had done previous work with the discord python api, and had a server that I had a significant number of messages on, so I pulled all the messages I had sent from this server to comprise the discord data set. For Facebook Messenger, I had downloaded a full extract of my profile, so I wrote a number of scripts to pull just the messages that I had sent from this extract, which comprised the Facebook Messenger data set. Finally, I combined digital copies of all of the essays and reports that I had written since Middle School, which comprised the Essay and Report dataset. Together, these datasets are quantified in Fig. 1, and comprise the majority of digitally written text that I have personally written dating back to middle school, so I suspected they would give me the best chance of building and training a model that generates text reasonably close to how I write. To simplify the dataset a bit, I converted all text to lowercase, and removed a lot of special characters that might negatively impact the results by introducing unnecessary additional classes to the vocabulary (such as $, #, &, * and others).

As a note, due to the personal nature of the dataset, I decided not to include the full dataset itself with my project submission, just the trained models and the Shakespearean and Alice in Wonderland corpuses that I trained some of my models on. There is not anything particularly special about the dataset I built though, other than that it consisted of text that I had personally written, any large corpus of text could reasonably be used to make a similar dataset. The structure of the dataset was one line per message that I had sent for the messages, and one line per paragraph for the essays I had written. When the model generates new line characters, the implication therefore is that everything after the new line is a distinct message or thought.

D. Creating a platform for testing

After having a few models and datasets to build off of, I wanted to create some type of system for trying out the different models I had built and comparing how they perform qualitatively. To this end, I pulled from some previous work I had done with the Django web framework to start building a page for quickly selecting a model and the parameters for it (see Fig. 2). The first step of this process was exporting the models I had made so far, and keeping track of all the necessary files to utilize the models, which includes a model.h5 file that completely describes the model, and a token-map.json file which I export to retain the token-to-number mapping used by the model to convert from text to numbers the model can understand (since for any given source corpus this mapping may change).

The page is a fairly basic single page app built with Django and some ancillary technologies to make it work as intended. After selecting a specific model and the parameters to use when running it, the page dispatches a request to generate text using the specified parameters. The requested page then uses Tensorflow to load the specified RNN model, and then based on the provided parameters will generate text using the loaded model. To make things portable, I developed the page within a docker container, so that the code can easily be built and deployed in any environment for use. With this important infrastructure built out, I was now more comfortable getting more adventurous with my model and trying some more significant changes to it with the knowledge that I had the old models available and still functional.

E. Trying new things

Now that I was more comfortable with being able to change my model, I started making lots of small tweaks to the structure of the network. I had seen a lot of the source literature using Bidirectional RNNs (BRNNs), so I initially tried using those in place of the normal RNNs I had used. BRNNs are the same as regular RNNs, but they pass information backwards in time as well as forwards in time, meaning that connections going backward could be utilized as well, implying additional information capture. What I failed to realize while I was trying this out is that the usage of the BRNNs was specifically in the Seq2Seq infrastructures for the Encoding layers. I was surprised to find that BRNNs were negatively impacting the results of the network, and spent some time trying to find out why it seemed like other's were using them successfully but I was not. What I found is that due to the way BRNNS are implemented in Tensorflow, the training it was doing wasn't actually training on the correct pieces of the sequence that I wanted it to. If I switched back to the many to one architecture I might be able to easily utilize BRNNs, but at an even more significant training cost, which would have problematic.

I also tried out using a word based model instead of a character based model, which I will refer to as the *word* model. In a character based model, the fundamental unit that the model is being trained on and told to predict is an individual character. In a word based model, this unit is shifted to instead be individual words. From the literature, the general consensus seems to be that a word based model generally yields better results overall, but is more difficult to train for the same set of text. In a character based model, if you limit yourself to an ASCII character set then you limit the number of characters you have to be able to predict correctly to be 256. With a word based model, the number of classes explodes, because every word is a possible class that you need to predict, which for my model meant tens of thousands of classes that need to be trained for. There is a bit of give and take with the comparison between the approaches as well, because if your corpus isn't diverse enough then it might do a good job of learning individual characters, but a poor job of learning a wide range of words, or vice versa depending on the way words and characters end up being distributed. The character based model is often considered more impressive and flexible, because it has to learn not only sentence structure and grammar, but how words are built up using prefixes, suffixes and roots. Due to the temperature adjustments, the character based model necessarily makes mistakes within words that, for a really good model, might give it away. At the same time, the word based model is incapable of spelling things wrong unless they were spelled wrong in the corpus text, which is its own

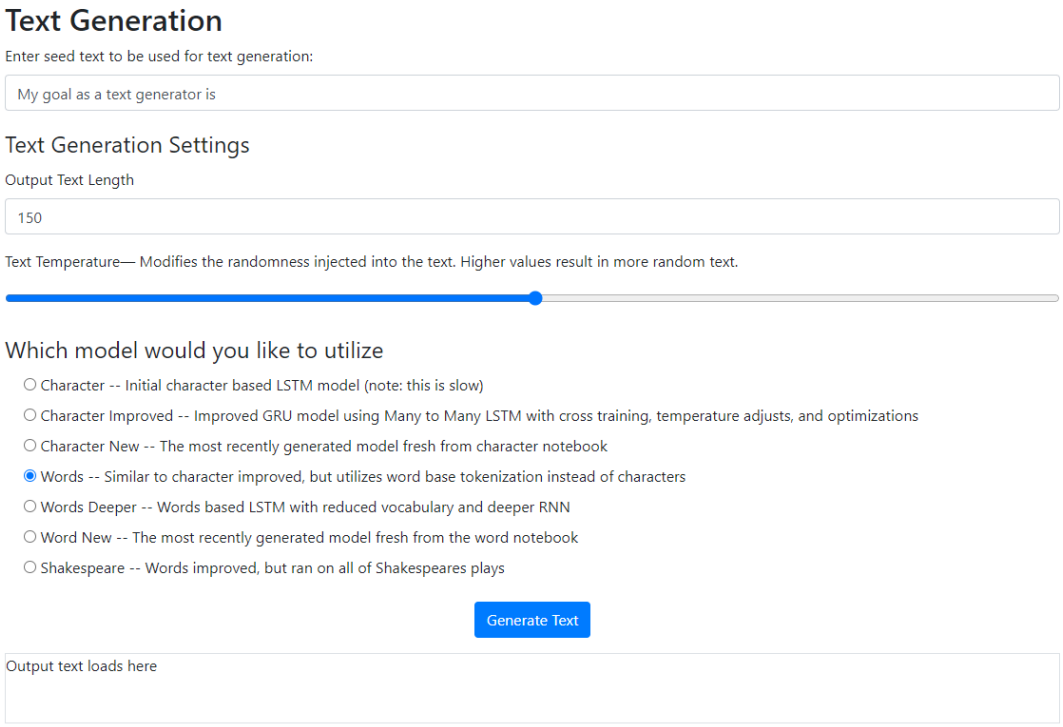

Fig. 2. The page I built for testing and comparing the models dynamically.

form of a give away, as humans do often spell things wrong. I initially had significant issues training my word based model, which lead me to wondering why the model was considered more powerful overall. Eventually I pinpointed the problem as being a combination of having too many parameters in my overall network, and using sequences that were much too long to successfully capture sentence structure well. I found that with words, sequences of lengths close to 20 words long did a good job of capturing sentence structures, whereas longer length sequences started having issues, since the more sentences you include in your training examples the more unrelated information you start to capture unintentionally.

F. My Results

In the literature, a BLEU (bilingual evaluation understudy) score is often used to assess results quantitatively for generative models. A BLEU score is typically calculated by taking a source text and translating it using one or more humans as well as a machine translator. The human translated text is called the reference text, and the text generated by the machine translator is called the candidate text. The BLEU score is used to calculate how different the candidate translation is from a set of reference translations. The more n-grams (sequences of n words) that appear in the candidate that also appear in the reference sequences, the higher the BLEU score will be, and the closer the candidate is considered to the reference text. I could have calculated a BLEU score for my models, but they are normally reserved for the domain of machine translation. Technically, most of my models are trained as machine translators, but when they actually are generating text they are treated as sequence classifiers. This makes the value of calculating a BLEU score a bit dubious, until I've transitioned to using a Seq2Seq model where the model will be translating from a context sequence to an output sequence. What the literature also commonly does, which I think is much more important for where my project is currently at, is show qualitative results of the models. This helps to show the progression of the models over time with examples that are more clearly understandable and apparent to a reader. BLEU scores are a great tool, and have been shown to match well with human intuition, making them very useful for comparing two texts that are already pretty good. For the results that I've attained thus far, I don't think a BLEU score is necessary to show the differences between my models, and my models aren't really well suited for calculating a BLEU score.

Qualitatively, Fig. 3 shows a comparison between the different models I have created so far. The character models (basic and improved) were asked to generate 400 characters, and the word models were asked to generate 200 words. All models were given the same seed text of "My goal as a text generator is", and the models with temperature adjustments had their temperatures tweaked to values that generated text that was appropriately random while retaining structure.

Looking first at the basic model, it is immediately apparent why the temperature adjustment is needed. The model loops after a few words, so the output isn't too interesting. What is interesting though is the way the model is generating things, because it is selecting each character one at a time, meaning

Model	Output Example
Basic	My goal as a text generator is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of the stuff is the problem that is the problem of
Improved	My goal as a text generator is the same as the same as well, in the right exists of an and are aracing because of the test in the card for the first solution to the problems and change deals in the world. when the first important process the most accident is a bit more enough to fix something that is a bit more to think it has a bath of the problem though. is the problem that the most set of each other progress of volume that
Word	My goal as a text generator is a lot of nonsense is a lot of the upper body than the past of the agriculture, and then i dont think you could be just a lot of important countries and dont be able to create a the same way that the system is probably a bit of the way to get some weeks and the mind is going to be a bit of a them from the past for the universe, and i think it is a good idea and something in the book, but i dont think you have a lot of the bond
Word (Shakespeare)	My goal as a text generator is a woman in the crown and in the grave of your grievous son. grumio my sovereign is my name. i am not in the honour and the very father to a man. petruchio i have for the blushing story of the part of a world in thy worm of troth. juliet but i pray you, sir, sir, i am a world with the part of their knave of the world of death, for you be a commons must have much that he will be a time, but i am a bond of love.

Fig. 3. Example outputs from the various models when provided with the seed text “My goal as a text generator is”

that it has a sense of where spaces belong, and what a word looks like. This may seem insignificant, but it shows that the model is learning some key information about the way words and sentences are structured. The improved model is where this starts getting a bit more interesting, because with the temperature adjustments in place the sentences have significantly more variance to them, and things like periods start showing up in the text at fairly reasonable intervals. The words don’t seem to be approaching anything close to actual sentences in the character model though, they are pretty clearly incoherent strings of words. A final thing to point out is that this model starts spelling things incorrectly, such as “aracing”, this is an interesting case where if the temperature causes a weird character to be injected, the model still tries to correct for it by turning it into something it thinks is reasonable.

The word models is where things become even more interesting, because reading the text generated with the word model, you can see glimpses of proper grammar and sentence structure happening. The sentences are very clearly run on sentences, and there isn’t anything completely intelligible, but it has now figured out how to use conjunctives, and it is displaying some evidence that it understands what a verb and a subject is. On some of the text that was being generated by the word model I noticed quirky things with the way I write that were clearly showing up in the generated text. Things such as starting messages with the phrase ”Yeah, . . .” or responding with ”mhm” followed by a new line and then an actual sentence of response. There were also instances where the model built URLs by itself, mostly links to youtube videos, which did not actually exist but were convincing. The final model is the same word model but was trained using a large set of shakespearean dialogues instead of my own dataset. The change in dataset starts to paint a picture of the strength of the model, and makes me suspect that if I can improve and expand the dataset more the model may get stronger as a result. We can see with the Shakespearean dialogue that the sentences are much more structured, and the dialogue format of the text is displayed very clearly. The line lengths are clearly much

shorter than the text generating using my own dataset, which shows it has some understanding of how long lines in the source text are. This helps show that the model has reasonably good transfer learning. These results make it obvious that there is room for improvement, but the direction of improvement isn't entirely clear at this point.

G. Future Plans

I think this problem is quite fascinating, and I do have plans to expand and improve my solution, partially because of my own interest, and partially because I think that I've learned and will continue to learn a lot by iteratively improving this model. There are still a few open questions for myself that I would like to answer. The first is whether or not the reason my current model does not produce more coherent sentences is simply because my dataset is too small. To answer this, I plan to explore a larger public dataset to test my model on, and see how it performs with longer training times and more data to train using. Second is whether I can adapt the data I have to a Seq2Seq model, and start approaching a model that is context sensitive. The reason I did not move forward with a Seq2Seq model yet is because Tensorflow does not have a Seq2Seq primitive, and I felt that focusing my time on trying to build something myself would be a poor use of time rather than focusing on gathering my presentation materials and report. My dataset currently only contains text that I have written, but I suspect I can pull the messages I was responding to in a reasonable manner to build a dataset that is closer to a machine translation problem. For example I could use timestamps and the previous few messages from others to be fed in as the context of my model, with the assumption that in most conversations I am probably replying to things that were recently said by others. This would significantly reduce the overall size of my dataset, but would provide better context. I think what I will find is that my actual problem is in fact with my dataset, and that if I had a larger dataset my model would perform much better overall. The output of a Seq2Seq model really shouldn't be much better overall than my existing model at generating coherent text, which leads me to believe that my model itself could still be improved on with respect to generating text, or that I've hit an impasse and my problem is unlikely to be solved without a larger dataset. One datasource that I hadn't pulled from for my dataset was emails that I have sent, and if I were to explore that further it might expand my dataset a lot, so that may be something else I try as well.

III. Conclusion

I wanted to explore techniques in NLG and NLP through the idea of generating text with a corpus of text that I had written. When I set out my initial goals for this project, included in the goals were creating a dataset of text I had written, building a model that would generate text based on this corpus, iterating on this model, building a web frontend that could interact with this model, and eventually adding contextual awareness into the model. Even when I set the final goal of contextual awareness, I was aware that this was a big ask for the amount of time that we had and the amount of data that I had available. I don't think my failure to meet this particular goal is reflective of a failure on my part, but speaks to the overall difficulty of the problem I was exploring.

I did successfully build a reasonably large corpus of text that I had written, with the earliest samples dating back 10 years and the most recent from weeks ago. I did successfully build an RNN model trained on this dataset, and it does generate text using words from me own lexicon, and captures some of the hallmarks of my writing style, but not fully coherent or convincing text. I did successfully build a page that can be used to experiment with all the models I had built, and compare how well they perform. From this perspective, I think I accomplished what I had set out to realistically accomplish. I still think this is an interesting problem, and that it would be neat to have better results that are closer to realistic sentences, so this will probably remain a toy project that I work on to try and improve, and to work towards getting a Seq2Seq model working. I would like to move my understanding much closer to the cutting edge work being done by Google's Deepmind and OpenAI, because I think the future of NLP and NLG is very exciting, and I'd be interested in watching it closer than simply from the sidelines.

References

[1] Daniel Adiwardana et al. *Towards a Human-like Open-Domain Chatbot*. 2020. arXiv: 2001.09977.

[2] Jason Brownlee. *Text Generation With LSTM Recurrent Neural Networks in Python with Keras*. Machine Learning Mastery. 2016. URL: https://machinelearningmastery.com/text-generation-lstm-recurrent-neural-networks-python-keras/.

[3] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *CoRR* abs/1308.0850 (2013). arXiv: 1308.0850.

[4] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80.

[5] W. John Hutchins. "The Georgetown-IBM Experiment Demonstrated in January 1954". In: *Machine Translation: From Real Users to Research*. Ed. by Robert E. Frederking and Kathryn B. Taylor. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 102–114. ISBN: 978-3-540-30194-3.

[6] Tomas Mikolov et al. "Recurrent neural network based language model". In: *Proceedings of the 11th Annual Conference of the International Speech Communication Association, INTERSPEECH 2010* 2 (Jan. 2010), pp. 1045–1048.

[7] Xiao Sun et al. "Emotional Conversation Generation Based on a Bayesian Deep Neural Network". In: *ACM Trans. Inf. Syst.* 38.1 (Dec. 2019). ISSN: 1046-8188.

[8] Ilya Sutskever, James Martens, and Geoffrey Hinton. "Generating Text with Recurrent Neural Networks". In: *Proceedings of the 28th International Conference on*

Machine Learning (ICML-11) (Jan. 2011), pp. 1017–1024.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215.

[10] Jian Tang et al. "Context-aware Natural Language Generation with Recurrent Neural Networks". In: (2016). arXiv: 1611.09900.

[11] *Tensorflow - Use a GPU*. tensorflow, 2020. URL: https://www.tensorflow.org/guide/gpu.

[12] *Text generation with an RNN*. tensorflow, 2020. URL: https : / / www . tensorflow . org / tutorials / text / text_generation.

[13] A. M. Turing. "Computing Machinery and Intelligence". In: *Mind* LIX.236 (Oct. 1950), pp. 433–460. ISSN: 0026-4423.

[14] Tsung-Hsien Wen et al. "Multi-domain Neural Network Language Generation for Spoken Dialogue Systems". In: (2016). arXiv: 1603.01232.