Interim Project Report: E2E NLG

Jackson Zaunegger

March 19, 2021

1 Introduction

The purpose of this project is to run a practical example using Statistical Language Generation. In this project we use the TGen model [2] developed by the Dialogue Systems Group from the Institute of Formal and Applied Linguistics in Prauge, Czech Republic. This system also has a evaluation script that is used to measure the precision of converting the meaning representations to sentences. To test this system we are using the End-to-End (E2E) dataset [1], developed by Jekaterina Novikova, Ondrej Dusek, and Verena Rieser at Heriot-Watt University in Edinburgh, United Kingdom.

I believe that developing a understand of how text generation works is important, because text generation is being more widely adopted. Many websites now have chat bots to help you find products or provide customer support online. In other cases natural language generation is used for creating new content such as ads or sometimes even whole articles, and similar to chat bots NLG provides a backbone for voice assistant technologies. Developing a understanding of how NLG technology works is not only interesting, but it keeps us up to date with changing technology all around us, as well as provides potential employment opportunities as a developer in industry, or as a researcher.

This project was a successful attempt at recreating a running example of using TGen and the E2E dataset. To complete this project we generate a neural model capable of taking in meaning representations of data and convert it to a natural language representation. This process includes converting the data from the E2E challenge to work with the TGen model, using a provided script. That converted data is then used to train the model which is then exported to a file. This exported model, is then used to generate sentences from the meaning representations, which outputs a file of these generated sentences. After a quick reformatting of these outputs, they are used to calculate various metrics to analyze how well this model learns the data and can generate sentences.

2 Data Preparation

To begin this project, I first cloned the existing TGen repository and created a anaconda environment to hold the project dependencies. I then cloned the E2E dataset, inside of the TGen project. Using the pre-made tools from TGen, the E2E dataset was converted into a series of txt files that meet the TGen input requirements. TGen provides a conversion script that automatically completes this process for you. Here is where I ran into my first technical problem, which Mike came up with a useful trick to get around this issue. This conversion file converts the input data, to five separate files. These files include, a file for the lexicalization instructions, another for the original lexicalized meaning representations, another for the original lexicalized references texts, a fourth file for the delexicalized meaning representations, and finally a file for the delexicalized reference texts. The github repository for TGen includes a README file which outlines this process in detail, and was incredibly helpful for completing this step.

When this script is running, in order to convert the meaning representations to a delexicalized form, the names of each restaurant are replaced with the term 'X-name', and the location is replaced with the term 'X-near'. This is because, during the training process the name and location are not so important, but the process of converting the various aspects of each meaning representation are. These aspects include the type of restaurant, the type of food they serve, the rating of the restaurant, the average pricing of that restaurant, and various other aspects. These are used to generate the references texts, which are then later compared to the original references provided. The conversion script also processes the provided references to convert them to a delexicalized form. This conversion step also replaces the location and name of the given restaurant, as well as provides spaces between special characters such as commas, apostrophes, hyphens, and periods. This is so the generated references from the meaning representations match the same format as the originals so they can be compared.

When going to train the model on this dataset, I used the delexicalized data as that is what the TGen system seems to prefer. I thought that if there was a script dedicated to creating these delexicalized forms, they are probably important. From my tests, the delexicalized data worked well, and the regular lexicalized caused errors in the code that would keep it from training at all. This could be due to some sort of technical issue on my side during the implementation, but I was so concerned with completing as much of this homework as possible I did not get much time to investigate training using the lexicalized data further.

3 Training Parameters

When training my model on the E2E data, I used the default parameters outlined by the config file from the TGen project. I did this to establish a baseline understanding of the system and to ensure that it was working. The default settings utilizes a learning rate of 0.0005, with 128 hidden units, a batch size of 20, embedding size of 50, with the Adam optimizer and LSTM cells. This system trained for 20 passes before saving the model. There are several other settings that are set by default, but I just wanted to touch on what I believe to be some of the most important settings. When running the system I passed in the data files for the delexicalized data, which are a result of the data conversion script, as discussed previously.

To train this system, the user provides the config file with the settings they would like to use, as well a file for the meaning representations and a separate file for the references, and the name of the output file. This output file is the model that is created from training stage, which is then used alongside the file that contains the lexicalization instructions and the file for references the user would like to use. This results in a output file which contains the generated reference sentences that are created from the model.

4 Train/Test Split

For splitting the data into multiple sets, I used the default sets that the E2E dataset provides. These include a training set, a development set, and two testing sets, one with text references, and one without them. The training dataset contains 42,064 entries, the development set contains 4,673 entries, the testing set without references contains 631 entries, and the testing set with references contains 4,691 references. Between all of these different sets, there is a total of 52,059 entries. From this I determined that the training set is comprises approximately 81% of the data, the development set comprises approximately 8% of the dataset, while the testing set without references comprises approximately 1% and the set with the references comprises approximately 9% of the dataset.

I just trained the system on the delexicalized training data. Normally I would choose to use a development set to fine tune the model before completing a full training run. I chose not to in this case because I was not altering any of the training settings, and just wanted to prove that the system was actually working as anticipated. It is generally a good idea to use a development set, rather than just relying on the training set to avoid having the computer view a sample multiple times, but this was not too much of a concern for me since I was not altering the training parameters in any way.

5 BLEU Score

BLEU stands for Bilingual EValuation Understudy, and is a measure to compare how closely a generated sentence matches a reference sentence. This scoring metric was developed for translating one language into another, but is also useful for comparing for same language generation. This metric states that an absolutely perfect match would have a score of 1.0, where a absolute mismatch would result in a score of 0. While BLEU is not a perfect metric, it does provide some advantages such as being a relatively simple calculation, it is easy for others to understand how it works, and is language independent. BLEU works by calculating N-grams for the original sentence, and the sentence to be compared and calculates the number of times that n-grams appear in both sentences, or in other words the n-gram precision.

After training the model I calculated the scores using the measure script provided by the E2E metrics repository. To run this script the user passes in the outputs from the text generation step, as well as the original reference samples to compare the accuracy between the two. I calculated these scores on the development and test sets, which in my case the development set really acts like a second test set, as this system has never seen this development set before since I did not use it in my training process at all. I attempted to run the generate sentences from the training set, but it seems to take a significant amount of time, and the partitions I created were killed before it could finish. Because of this I was unable to calculate the scores on the sentences generated from the training data. This is also largely due to the amount of time I spent diagnosing problems with getting the model to run correctly, as well as calculating the scores. I created a simple table to illustrate the scores I got from each set, shown below.

BLEU Scores		
Score Names	Dev. Set	Test Set
BLEU	0.6398	0.6248
NIST	8.2897	8.3066
METEOR	0.4429	0.4411
ROUGE	0.686	0.6746
CIDEr	2.1172	2.1775

When comparing the results from the test set to the reported scores for the E2E challenge, the BLEU score falls just shy of the reported baseline scores for the TGen Model, with the rest of the scores being similar to TGen's base performance. This slight difference could come down to a different number of passes, and I would wager that increasing this pass count could yield better performance as the computer has more time to analyze and learn from the input data. It is not surprising to me that the Dev and Test set scores are similar, because this system has not encountered either during its training process, as I discussed earlier. I think the difference between the scores can also be accounted for due to the fact that the two sets have a different number of meaning representations and reference samples. When looking at the NIST score, both NIST and BLEU beat the system I trained here. The METEOR and ROUGE scores compare to the base TGen system, with CIDEr being slightly lower. Again all of these scores could most likely be improved by increasing the training time.

6 Challenge Results

Overall, I think the system I trained here is comparable, with some slight differences in performance compared to the base TGen model. I think this system could be improved by increasing the number of passes/epochs, adding alpha decay or momentum to affect the learning rate, or adjusting the embedding size. I can't say for certain that using the delexicalized data is better than the lexicalized data, since I could not get it to run. I would guess that the delexicalized data would perform better, since removing the name and location from the data would improve the ability for the system to generalize this data. I think this because most of the time restaurant names, and locations are going to be unique so it doesn't make much sense to me to include this in the training data, because the likelihood of encountering these terms would be low. I think removing them is a good call, so the system can focus on the other aspects that are going to be similar between the data samples.

I am also unsure of how data pre-processing would effect the performance of this system, because I used the default tools provided by the TGen system. I have thought about other ways to prepare the data, and I can't think of any methods that would make sense. I would be interested to see how performance would compare if I was able to get the lexicalized data to run in the system. I think it would not perform as well, for the reasons I discussed above. I would also be interested to see how this system would perform using different data much as movie reviews or something similar to this. You would obviously have to train a new model, as this system is specifically trained to handle restaurant reviews, but in theory this model should work with other data.

7 Conclusion

Overall, this project has been a great learning experience. Working with bridges is the first time I have ever working with a high-performance cluster over a ssh connection. I have worked with a HPC before, but it was locally running directly on the machine, so it was great to gain experience with a contrasting approach. The other great thing about this project, was gaining experience trying to improve a pre-made system rather than building something from scratch. While being able to build new solutions independently is a great skill, I believe building on others work can offer different learning opportunities. For example, when building my own projects I often wonder if an issue that arises is due to an issue in my code, or just misconfigured parameters. I'll admit I had some difficulty getting this project setup properly, but once I got things working I learned a lot.

During the training process, I have thought about ways that this system may be improved, which I will reiterate here. I think adding more passes or epochs could improve performance, as well as introducing alpha decay or increasing the embedding size. The main takeaway I have gotten from this, is that GPU access plays a vital role. I understood the importance of GPU utilization, but it becomes so much more clear when implementing projects. A lot of my previous work in Machine Learning has dealt with image data, so it's been nice to analyze other types of data.

References

[1] J. Novikova, O. Dusek, and V. Riser, *End-to-End Natural Language Generation: The E2E NLG Challenge*, Computer Speech and Language, vol. 59, pp. 123-156, Jan. 2020

[2] O. Dusek and F. Jurcicek, Sequence-to-Sequence Generation for Spoken Language Dialogue via Deep Syntax Trees and Strings in *Proceedings of the 54th Annual Meeting of the Association of Computational Linguistics*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp-45-51.