Oliver Moody
5/20/2021
CMPU 366
Gordon
Link to repo: https://github.com/olmoody/boxscore-nlg

NBA Box Score NLG Final Paper

This project is designed around the use of the Harvardnlp boxscore dataset. This dataset pairs NBA box score data with human written summaries of the game. The summaries are taken from two different sources: Rotowire and SBnation. I focused on the rotowire summaries because they are more focused on the stats themselves. The SBNation summaries come from local writers who often focus on storylines that don't appear in the box score making it very difficult to perform any sort of natural language generation. There are "4853 distinct rotowire summaries, covering NBA games played between 1/1/2014 and 3/29/2017" (Wiseman et. al). The data is JSON-formatted which makes it easy to read into a python script.

The majority of existing work using this dataset attempts to generate full summaries given the box score data. While my task ended up being slightly different, it is still important to consider these models and what techniques they used. Wiseman et al. introduced this dataset and built a standard encoder-decoder model using a LSTM recurrent neural network with a copy mechanism. The copy mechanism simply refers to the idea of pulling words straight from the box score when appropriate. This could include specific numbers or names of teams or players. I attempted to use a similar technique. Another element of Wiseman et al.'s work I tried to implement was the information extractor. They trained a simple model that, when given a sentence, could identify what stats were being referred to. I used a simplified version where if the name and stat matched in both the sentence and the box score, I assumed that stat was being discussed. Each model that I built had to label or extract data in a slightly different way which

will be discussed further later on. Another group who worked on this NBA dataset is Rebuffel et al. As opposed to a standard LSTM model, they use a hierarchical model that generates summaries by encoding the data-structure at multiple levels. A low-level transformer encodes the data and that result is passed into a high-level transformer which encodes it further. They then use a similar two-layer LSTM network with a copy mechanism to decode. This technique is currently the best at generating full summaries. They report a BLEU score of 17.5 while Wiseman et al. report a BLEU score of 14.5.

As I mentioned earlier, my goal for this project was not to generate full summaries like the papers above. Instead, I attempted to generate one sentence at a time. My hope was this makes the task simpler because I can separate the process of selecting which stats to represent and I can eliminate the challenge of ordering the stats in a natural way. I ended up building three different models that go about this task in different ways. Each of these models at its core is a LSTM recurrent neural network built with Tensorflow's Keras library. Keras has built-in functions for creating and training LSTMs as well as a Tokenizer class for turning sentences into sequences of integers. I followed Jason Brownlee's tutorial to get started. Working with these models required fiddling with different settings to obtain the best output. This included changing the batch size and number of epochs to conduct a thorough training session in a reasonable amount of time. In order to speed up the training process, I also used Vassar's lambda-quad machine's gpu. Another setting I had to determine was how many of the previous words to consider when selecting the next word in the sentence. After a couple of trials I found that smaller input sequences led to more accurate results. After starting with an input length of 50, my final models all use a length of 15. One final thing I altered was how to select the next word when actually generating a sentence. There are two ways to do this, pick the most likely word

according to the model (np.argmax(pred, axis=-1)) or pick the word based on the probability distribution generated by the model (np.random.choice(len(pred[0]), p=pred[0])). I ended up using the second option because both seemed to give similar results in terms of readability and accuracy but the more random model outputs more interesting sentences.

The first model I built (model_1/) was given three dictionaries to represent points, rebounds, and assists. Each dictionary would have up to three player names (although usually just one) along with their total for the given stat. The hope was something like [{“Lowry”:24}, {“Lowry”:4}, {“Lowry”: 8}] could lead to an output like “Kyle Lowry carried the load for the Raptors, accumulating 24 points, four rebounds and eight assists.” The information extractor (model_1/infoextract.py) for this model went through each sentence in the training data and tried to find pairs of names and stats present in both the sentence and the box score. If found, this pair would be added to the correct dictionary. The dictionaries were then added to the beginning of the sentence. For all three models, since I was only focusing on generating sentences referring to a stat or stats, if the information extractor script did not find any matches, the sentence was not included in the training data. The summary was then split into sequences of 15 words and paired with a target word (the next word in the sentence) (model_1/main.py). The final version of the model took 2 hours on the gpu to complete 50 epochs with a batch size of 256. The results were for the most part grammatically correct, but there was a serious problem. Here are some examples of input-output pairs (from model_1/results-stats.txt):

[harden 29 *empty* *empty* *empty* *empty* harden 13 *empty* *empty* *empty* *empty* harden 11 *empty* *empty* *empty* *empty*] ->

“james robinson followed with a double - double off the bench , amassing 24 points , 7 rebounds , 7 assists and 3 blocks on the loss .”

[lopez 20 *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty* *empty*] ->
“gordon parker stauskas added 17 points , 2 rebounds) , superstar will mudiay 's shot against the grizzlies and came by the bulk of 25 points on 9 - 22 shooting mark from the field .”

As you can see, the names and stats generated seem to be completely random. A quick python script (model_1/full-data-test.py) told me that after 2,000 generations about 3% of names from the input matched with the output and about 10% of stats overall. To me this meant the model wasn’t really relying on the input much, if at all, and instead generating a random sentence.

My next model (model_2/) tried to eliminate these random names by replacing all relevant names and stats with generic tags. The information extractor (model_2/infoextract2.py) for this model again identifies name-stat pairs but this time just replaces them with tags such as *lastname* or *rebval*. I figured since all the names were genericized, there was no point in attaching the data to the training data. This means this model is a text-to-text generation model. In other words, it captures the essence of the rotowire summaries and outputs text that could fit right in with the real summaries. The model was trained in a very similar way (model_2/main2.py), relying on just the sequence of previous words this time. This model was also trained using the gpu and took 2 hours to complete 50 epochs with a batch size of 256. Once each sentence was generated, I needed to replace the tags with values from a given game. To do this, I simply selected the top three scorers in the box score and placed their stats in different lists (one for first names, one for points, ...) (model_2/full-test.py). I then looped through each tag and

replaced it with a value from the correct list, incrementing the index to take next from that list each time. The results aren't all perfect, but they seem pretty good (model_2/results.txt):
"Anthony Davis led the team with 33 points and made a game - high 7 from beyond the arc ."
"Nikola Vucevic matched Millsap 's scoring total and added 15 rebounds and 3 assists ."
"impending implemented Thomas 's Isaiah Rivers score a chance of performance of his own , and still accumulated 24 points , 0 rebounds and 2 assists on 7 - of - 9 shooting ."

In an attempt to get a more quantitative analysis of how accurate these generations are, I created a quick test to see if I could differentiate between generated and real sentences. My test pairs a generated sentence with a sentence taken directly from a summary and asks you to guess which one was written by a human (model_2/testmodel/runtest.py). Through 200 trials of myself and one other person, about 83% were correctly guessed. While this may seem high, most of the time the thing that would give it away would be one number or one word. A lot of the mistakes were as simple as "8-5 3pt" meaning the player made 8 of 5 shots. These errors could be removed by replacing more than just points, rebounds, and assists with generic tags. This would get me closer to the goal of 50% accuracy.

While happy with these results, I felt I had gotten away from the true data-to-text essence of this task. I wanted a model that could pass in the data and see the output with accurate names present. My attempt at this was a model (model_3/) that was essentially a combination of the previous two. The information extractor (model_3/infoextract3.py) for this model again replaces each name or stat, but this time the tag includes the index of the value in the box score. For example if the list of last names in the box score was ["Brown", "Tatum", "Walker"], "Tatum" in the sentence would be replaced with "*lastname1*". The list of each stat is then also appended to the sentence. The hope was the model would somehow take advantage of the data presented

while also learning to pair the same indexes with each other. For example, “*firstname12*” would usually be followed by “*lastname12*”. Due to the amount of data presented with each sequence, this model took 8 hours for the gpu to make it through 50 epochs with a batch size of 256 (model_3/main3.py). The results, unfortunately, did not turn out as I had hoped (from model_3/test3.py):

“*firstname0* *lastname6* led the sixers , dropping *ptsval4* points , *rebval7* rebounds and *rebval15* rebounds .”

“*firstname17* *lastname17* tied a game - high to score was steph *lastname2* forward - *firstname1* *lastname14* matched the game , the eastern period through the career - late while shooting *astval0* - point line to score 37 minutes .”

It clearly has not learned to match indexes with each other and I have no real way of knowing how much having the data helped or hurt the model (it certainly made it slower to train). At this point, this model provides nothing that the previous two models don’t already have. Perhaps with tweaking of the settings or maybe even just longer training, we would begin to see better results. However, I believe the more likely solution is to find a new way entirely to encode the data. I have looked into different encoders from keras and other libraries, but I couldn’t find anything simple enough to implement in the given time frame.

Overall, I’m happy with the output from my text-to-text model and even my original specific stats model. As I mentioned, the text-to-text model could be further improved by tagging more items such as team names and team stats. The function for replacing the tags could also be tweaked to be more accurate. As I just said, if I were to continue this project, I would need to find a different way to encode the data and separate that from the previous word sequence input.

One application of this project I would like to see one day is a twitter bot that could automatically get a boxscore and tweet out sentences describing the game.

Sources

Brownlee, Jason. “How to Develop a Word-Level Neural Language Model and Use It to Generate Text.” Machine Learning Mastery, 7 Oct. 2020, machinelearningmastery.com/how-to-develop-a-word-level-neural-language-model-in-keras/.

Rebuffel, C., Soulier, L., Scoutheeten, G., Gallinari, P.: “A Hierarchical Model for Data-to-Text Generation.” *ArXiv.org*, 20 Dec. 2019, arxiv.org/abs/1912.10011.

Wiseman, S., Shieber, S., Rush, A.: Challenges in data-to-document generation. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing. pp. 2253–2263. Association for Computational Linguistics, Copenhagen, Denmark (Sep 2017). https://doi.org/10.18653/v1/D17-1239,