Text Generation with Transfer Learning and LSTM

Matthias Eyassu

Goal:

The objective of this project is to build a neural language model with the capacity to generate reams of science fiction in the creative and humorous style of authors like Adams, Vonnegut, Colfer, and others; a style that is markedly different from conventional science fiction. Traditionally, those books feature complex, regular plot structures with overarching themes, several mainstay characters, intricately worked resolutions, and carefully measured pacing which varies in proportion to the intensity of the events at a given point. While these things make for a more realistic and immersive experience, they are signs of a thinking author writing with intent and dictating from a preconceived plan whereas the works of Douglas Adams, such as *Hitchiker's Guide,* are more erratic and unpredictable in nature. For this reason, using works written in his style as training examples for the models will likely result in more exciting output. At the same time, they sometimes verge on being incoherent so the dataset will be augmented with some classic novels by Wells. Hopefully, this will teach the model to constantly infuse rigidity into the output as it is being generated. The challenge will be to avoid high-level inconsistencies or contradictions and maintain stable low-level language structures responsible for grammar, syntax, and semantics while producing surprising and amusing text. Since I am wr

Sample I/O:

Input: Writing by Adams and other authors with a similar style along with Wells

Model: the generative model

Output: a few pages of text

If this project is successful, it will likely result in very funny output and the models could be trained on any style of literature so maybe authors could use it to spark their imagination although I am sure there are professionally developed programs like this already available. It admittedly does not solve any pressing problems.

Methods:

Neural language models are extremely popular so I decided to train a single-layer, unidirectional LSTM from scratch and fine-tune an instance of GPT-2 on the dataset that I collected earlier in the semester with Huggingface's `transformers` library. The data is made up of the following books: *The Time Machine, Life, the Universe, and Everything, So Long, and Thanks for All the Fish, War of the Worlds, The Hitchhiker's Guide to the Galaxy, and The Restaurant at the End of the Universe. And Another Thing* is the held-out test sample. Each book is around 200 pages, so there are approximately 1400 pages of text for the networks to train on. The data was preprocessed by removing extraneous symbols (punctuation, html tags, numbers, newlines, random dashes), expanding contractions, separating hyphenated words, and stripping excess spaces.

In the case of the LSTM, encoding maps were then built relating the words in the vocabulary to unique numbers, and trigrams were constructed. The training loop is not controlled by early stopping, in hindsight, this would have been simple to include since the function could just compute the perplexity over the test sample after every epoch. This would have prevented the model from overfitting to the training set (i.e. memorizing large portions of it). Initially, I tried using pretrained embeddings from FastText when I trained the network locally on my CPU but I eventually switched to Google Colab for GPU access which has very limited RAM (~12 GiB) so I decided to start the training with randomly initialized word vectors. Earlier in the semester, I was training the LSTM for only around 20 epochs and the output quality varied imperceptibly after the introduction of FastText embeddings so their omission may not be significant. On the other hand, this may have been because of the short model training time. I also tried including start and end of sentence symbols but the early, hardly trained LSTMs tended to spew out long strings of them all at once instead of using them to indicate the extremities of a proper sentence. As a result, I removed the lines that inserted them from the preprocessing function. In an effort to improve legitimacy of the output, I implemented a bi-directional LSTM hoping that its ability to process sequences from both sides would somehow help with it to retain more relevant information from previous forward passes. At this point, I reintroduced the start and end of sentence symbols in order to inform the network of its orientation in the sequence but the same problem mentioned above arose. The bi-LSTM was very slow to train so I decided to use PyTorch's `DataLoader` class and write a custom `collate_fn` to batch process the trigrams. Instead of imposing this structure on the data during preprocessing:
`[trigram_0, ... , trigram_n ]` where a trigram is defined as: (`[context....], target),`
I instead formed the following list batch-wise in the following way:
`[ [ (trigram_0_context, ... , trigram_n_context), (target_0, ... , trigram_n) ] *more batches here*].`
Each forward pass would accept a series of contexts and produce a list of corresponding predictions to quicken training. While this improved training time, it completely destroyed the network's ability to form language sequentially and with concern for preceding blocks of text. The output was impossible to understand and far worse than the uni-LSTM but it would still probably outperform a traditional, count-based n-gram model. The quality of the output barely budged even after relatively long training times, so I turned my attention to fine-tuning a GPT-2 model at this point. I could not ignore a nagging suspicion that there was a bug in the forward propagation function but extensive debugging failed to uncover anything significant. At any rate, the bi-LSTM is not part of the submission mostly because the text it generated is so terrible. It is also incomplete since I decided to focus on GPT-2 instead of taking the time to equip it with a perplexity function. In short, it was a failure.

As is the case in computer vision, transfer learning is an extremely widespread practice in NLP. There are many publicly available networks pretrained on massive datasets such as BERT, GPT, ELMo, ALBERT, and XLNET. There are two kinds of transfer learning: fine-tuning and fixed-feature extraction. In fine-tuning, the idea is to modify the weights of the entire model during training while in fixed-feature extraction, the earlier components which are responsible for recognizing low-level features such as basic syntax and grammar are held constant during training while the parameters of the later layers are modified. The reasoning behind this second technique is that later layers correspond to domain-specific features and need to be adjusted in order to enable the model to perform well on some niche task. I chose to fine-tune GPT-2 since it is a complex architecture and I wanted to minimize the potential for serious errors during experimentation. This task was made relatively straightforward by a well-written Jupyter notebook. The function responsible for generating text is very similar to the one in the LSTM file. It starts with some primer text and repeatedly queries the network for output while aggregating all previous outputs together at every iteration. The network samples from a multinomial distribution. In terms of preprocessing, a function in `utility.py` takes in as input a set of file paths to some books in the training set and outputs the files such that each line contains exactly one sentence. This makes it easier to train GPT-2 on a sentence-wise basis. The network trains quickly on GPU, finishing several epochs in under an hour. The output after a few epochs (bounded at 140 characters) failed to impress. The sentences were well-formed and semantically superior to anything the LSTMs produced but they were also mundane resembling text encountered in slow-moving novels or dreary newspapers.

The **Experiments** section will discuss the peak output of the networks further; the examples cited here were preliminary results chosen to demonstrate the methods employed in their development.

Currently, NLG is mostly performed with encoder-decoder sequence-to-sequence models equipped with attention capabilities to learn the relative significance of the elements of a given sentence. RNNs are commonly used architectures, especially LSTMs which are designed to sidestep the vanishing gradient problem. The transformer architecture has exceeded other topologies in recent years. As mentioned earlier, it is rare for an NLP practitioner to train an entire model from scratch given the accessibility and superiority of other pretrained networks.

Experiments

The experiments mostly consisted of generating text with the models. The training data consisted of the books mentioned in the beginning of the **Methods** section. The most interesting outputs are shown below along with a short description of the source model.

Model A: Uni-LSTM trained on fraction of *Hitchhiker's Guide* for ~20 epochs

Output:

- “though he burst through it was a small packet of ebony”
- “we left one of the encyclopedia galactica with a black hole mounted in the galaxy”
- “whhhrrrr said marvin in a hushed voice”
- “i think we will be having tremendous difficulty with the improbability factors”
- “they awoke screaming through it in a ⁻blaster in the platinum suit into space”

Model B: Uni-LSTM trained on entire dataset for ~20 epochs

Output:

- "he looked at his face and i think " he said that he could say to take us"
- "arthur watched the mind and he looked up"
- "oh " he said " your planet i would hurt by stitching on the sort of cloud whilst"...
- "at nasa he tugged"
- "going to be a little grim and a little " said trillian"
- "he was therefore powerless to take on his scattered back of a sign from which we used to act by the universe ' s junk"
- "trillian nodded again again in a strange thing in the land at they went down"
- "he pressed it chanced him as he left him and he looked round to his chest"
- "he stared wildly and looking round its component parts"
- "a little thing had taken man into the sky"

Model C: GPT-2 fine-tuned on entire dataset for ~10 epochs

- "not the bloody disease, but the very natural numbers of whole animal survivals."
- "They were going to lay down their weapons, we were to think, and continue this restlessly insane story of guns, pistols and small artillery pieces."
- "Not like anything he had ever been stung with as well."
- "You've got this under control," said Ford, "I'm not going to let you hit me."
- He had to find a kitchen and towel he had to buy some polished concrete"
- "Arthur gritted his limbs and shouted"
- "hitchhiker of the long long road with a handkerchief."
- "a raft of flying spectres, all screaming and leaping like ghosts, peering up into the clouds and heaving."

Model D: Bi-LSTM trained on a fraction of *Hitchhiker's Guide* for ~20 epochs with FT embeddings

- "it man the cylinder me they is to a his have to twirled"
- "man the ! " i he the and and ' and heather"
- "that the we is it not i ' ' " " a " it " the and !"

Qualitatively, the models performed well in general. I planned on computing the perplexity of the networks over a held-out test set but I ran into difficulties so I will rely on subjective criteria to assess the models. Models A, B, and C produced respectable segments of coherent text fairly regularly. Training set size seems to correlate well with the number of attractive phrases. Model A compensated for this by being the most amusing and inspiring network. Its output resembles Adams the most. Model B generated many quality sentences with strong ties to the books in the training set but they did not break any new ground in terms of structure or content. I was happy to see the fourth output listed above, "at nasa he tugged" since the model correctly inferred that NASA can be a place despite its infrequent appearance in the training set. Model B also referred to characters more in a coherent way. Model C showed mastery of grammatical structure with amazing regularity. Almost every sentence of the hundred

it outputted was well-formed. The issue is, they tended to be dull and copied from random sources sometimes which is a common complaint with many pretrained Transformer models. The second sentence (“They were going to lay down their weapons, we were to think, and continue this restlessly insane story of guns, pistols and small artillery pieces.”
“) conveys the scene of two parties dueling which is in line with the general plot of Adams' work. If it had explicitly set the scene in space, it would easily have been the best sentence produced during this project. The fourth sentence (“You’ve got this under control,” said Ford, “I’m not going to let you hit me.”) reflects Ford’s paternal and reassuring personality, Arthur’s shaky and nervous traits, and the perilous situations they usually find themselves in. The fifth and seventh sentences also reflect unique properties of the books in that they identify towels, handkerchiefs, and concrete as being important objects in the plots therein. The fifth sentence also sounds very similar to Adams, he often starts lists with normal objects and then ends them with out of place objects as a joke. The last sentence is an ugly duckling with no relation to anything relevant; it is well-formed, though. I think its formation may have been influenced by the Wells novels. The bi-LSTM was the worst model by far, which was initially very surprising to me. I thought it would outperform the normal LSTMs since it could process data more thoroughly and from multiple directions. In hindsight, it probably generated poor text because writing is a unidirectional activity.

The baseline model was Model A. Except for model D, the subsequent models soundly outperformed it in terms of sentence quality and the volume of legitimate output. I also think they would have had lower perplexities if I had the time to implement it. I think the other models outperformed Model A because they had more training data at their disposal and trained for longer. GPT-2 was pretrained on a large corpus so it was at an advantage grammatically and semantically.

Conclusions: I learned that bidirectional LSTMs are not always improvements to normal LSTMs. I also learned that pretrained models excel in terms of grammar and semantics but may not always produce the most novel or relevant sentences. This is likely because the additional training set has minimal influence over the existing parameters of the network especially considering the fact that transfer learning is usually done with very low learning rates. In the future, I might try freezing the early layers of GPT-2 and then increasing the learning rate to a normal value (e.g. 0.1) to see if the model starts to write sentences with stronger relationships to the novels. More generally, I learned that training on GPUs dramatically improves efficiency. Initially, I was training on my own CPU for several nights in a row. In the future, I may add more training data, use different pretrained networks, combine multiple evaluation metrics (including perplexity) to assess performance, and write a simple function to check if phrases generated by the models are in the training set (to check originality). Overall, I enjoyed working on this project.