Sequence-to-Sequence Model for Joke Generation

Jiri Roznovjak

December 23, 2016

1 Problem Formulation

In this project, the goal was to train a model to generate jokes. In particular, we are interested in the kinds of jokes where the first part is a question (”Why did the chicken cross the road?”) and the second part is an answer, the ”punchline” (”Because it wanted to get to the other side). This can be formulated as a sequence-to-sequence machine learning problem, where the question as a variable-length sequence of words is the input, and the answer is the output, also a variable-length sequence.

The dataset for this project was chosen to be the ”subreddit” r/Jokes (www.reddit.com/r/Jokes). It contains tens of thousands of jokes that users submitted since 2008. The challenge was to be able to download all of them, and decide which ones are the ones of the form we are looking for.

The python tensorflow library was used for training the model.

2 Background

Recurrent neural networks have enjoyed recent success in machine learning problems where the input or the output have a variable length. RNNs store an internal state, and use it as a part of the input. They are therefore able to remember the features of the part of the input they have already seen. RNNs have been used for tasks such as machine translation [1], image captioning [2], or language modeling [3].

A special kind of a recurrent neural network is a long short term memory network (LSTM). LSTMs store an internal state that is capable of remembering long-term dependencies. They are therefore suitable for problems that require learning such dependencies, and are widely used for problems in natural language processing [5].

A fairly new mechanism that is used in training RNNs is called attention. Attention enables the model to focus on different parts of the input, and has been successfully applied on problems in natural language processing as well [4].

3 Getting and Preprocessing the Data

The data were downloaded from the r/Jokes subreddit. Reddit has a public API that should allow accessing its content. One has to create a reddit developer account first, and use a private key.

I tried using the praw library, which is a python wrapper to the reddit public API. However, a problem arose: when getting posts form a specific subreddit, only the 1000 latest posts are fetched. This could be resolved using a time range search, but that didn't work using the API.

I ended up scraping the subreddit using a python script and the requests library. The script is included as scripts/scrape.py. Another problem that arose was that reddit limits the number of requests within a specific time period, and after more than that many requests, it instead sends back a page saying that one should wait X number of seconds (which varies randomly) before making another request. I dealt with this by parsing the number of seconds the page is telling me to wait and making the script wait that number of seconds.

I detected the jokes that were of the desired form by checking whether the last character of the title is a question mark. If it is, I save the title and content. It is possible some jokes were missed, but this turned out to be by far the simplest way.

A total of around 42000 title/content pairs were collected. Jokes with an empty-string content were removed. I also checked the distribution of lengths of the titles and contents. This is important because of a procedure called "bucketing" explained later in methodology. I removed all pairs with a title longer than 100 characters or with content longer than 500 characters. After preprocessing I was left with 38268 examples.

It is (without a surprise) worth noting that the online reddit community does not heed the political correctness or general appropriateness of its content. Many of the examples can be found to be racist or in other ways inappropriate.

4 Methodology

For training I used the script from the tensorflow tutorial for sequence-to-sequence models [1]. That script is originally used for training a machine translation model from English to French. However, our problem is basically concerned with "translating" a question to an answer, and with modifications it was possible to use this model.

The data was split into 32000 for training, and 3134 for validation, and 3134 for testing (roughly 92/8/8 split). I chose to give more data to the training set than usual, because the size of my dataset is fairly small (especially in the context of NLP problems).

The model consists of an RNN encoder and an RNN decoder. One has to define a vocabulary for both, in the original model the encoder had an English vocabulary and the decoder had a French one. Here I combined the set of questions and answers to define a common vocabulary for the encoder and the decoder. For creating the vocabulary I used the script in data_utils.py. There were 37496 tokens in the vocabulary. It is common to reduce the size of the vocabulary by replacing low-frequency words by a dummy token in order to reduce the number of parameters of the model. However, in our case, 20427 tokens appear only once in the dataset. Getting rid of single-occuring tokens would greatly reduce expressiveness of the model, so I chose to keep all of them.

The original script reverses the inputs while training, as explained in [2]. This increases performance of a machine translation model, however likely wouldn't have such effect on my model (why it increases performance of a translation model is hypothesized in the paper), therefore I chose to not reverse the inputs.

A 3-layer LSTM model with 1024 units in each was used for training. This is the default choice from the tutorial, and since it works well on the translation problem, I chose to go with it - since training takes many hours, it would have been very time-consuming to evaluate different models, or different choices of hyperparameters.

The "embedding_attention_seq2seq" function that is provided by tensorflow was used. As the name suggests, the function takes care of word2vec embeddings for us, and implements an attention mechanism as a part of training.

I also used a bucketing mechanism that is in detail explained in [1]. Basically, it defines several "buckets", which are pairs of fixed sizes for our inputs and outputs. We pad the input and output of an example with a special pad token to match one of the buckets. An example falls into a bucket for which it needs the least amount of padding (but where the size of input and ouput doesn't exceed the bucket's input and output size). Bucketing is supposed to increase performance of training, especially in cases where the distribution of sizes of inputs and output varies greatly, as it is true in our case.

The sizes of buckets were chosen to be (100, 20), (100, 50), (100, 100), (100, 500), where the first number is the size of the input and the second is the ouput. This was in accordance with the distribution of sizes of inputs and outputs - the sizes of inputs varied much less than the sizes of outputs.

After a major memory-related problem during training (described below), the sizes of buckets were changed to (100, 20), (100, 60). I didn't have the time to modify my data to remove examples with output length longer than 60 characters, so those examples were simply skipped. This reduces the total size of our dataset by roughly 11 % to 33792. I assumed those examples were uniformly distributed among the training, validation, and test sets.

5 Training

The training was performed with a batch size of 64. The learning rate was initially 0.5. After 4600 steps, I noticed a slow convergence of the model, so I increased the learning rate to 2.0. After 7600 steps I decreased it to 1.0, where it was kept for the rest of the training (23000 steps in total). This means the training went on for roughly 50 epochs. It took about 12 hours in total on 4 NVIDIA Titan X GPUs.

I encountered a major problem when I initially attempted training - the script was crashing because it kept running out of memory. I found out that it was caused by training examples where the size of the ouput is greater than 60 (the size of the input was always padded to be 100). Therefore I chose to redefine the buckets as described above. This resulted in a loss of about 11 % of the data. Had I had more time, I would have tried to come up with a more clever solution, or would have done research on how this problem is usually dealt with, but given my time constraints, I was forced to go with this simple solution.

6 Evaluation

In the original script a metric called perplexity was used for evaluating the model. Perplexity is a measure that comes from information theory and is similar to cross-entropy. However, the value it produces isn't standardized, so it is useful for instance for comparing different models, but doesn't tell us anything by itself.

For a simple evaluation it is the best to feed the trained model actual inputs and see that outputs it produces. Below there are 3 tables of inputs and predicted/actual outputs. First table corresponds to data taken from the training set, second from the test set, and third are different kinds of inputs that I came up with and that help us understand the model.

Training set

Question	**Predicted answer**	**Actual answer**
Why did the chickens cross the road?	Because they were Turkish.	Because they were Turkish.
Why are lemons yellow?	They don't know it that's why people's number that.	They don't know it either - that's why they are so sour
What do you call a nosy pepper?	Out jalapeño!	jalapeno business!
Is venison deer?	No ? Well, your nuts doesn't your face off	No really. Only paid a couple of bucks.
How do pirates know they exist?	They think, therefore they ARRRRRRRR!	They think, therefore they ARRRRRRRR!

Test set

Question	**Predicted answer**	**Actual answer**
What do you call a thieving duck?	Hung off	A robber duck....
How do Russians drive to Alaska?	By 00 on the country.	By bearing straight
What happens when you flip an 8 to it's side?	it it s boy to make a perfect person.	Everything, given enough time
Why is there no Windows 9?	Because 0 ate 0	Because 7 8 9.
What is Apple users favorite movie?	Wall-E	No Escape
What does Gandhi not like on the Internet?	Propanic.	Salt
How do you make Holy Water?	You put the Hell out of it.	You boil the Hell out of it.

Own examples

Question	**Predicted answer**
Why did Trump cross the road?	The employees value Donald Trump
What is the difference between a cat and a dog?	One is claws, the horns is at the reach
Where did Americans come from?	See dinner alone.
Trump	The white of America is a person who is watching on president.
German	Nein

7 Conclusion

We can see that the model has successfully learned the training set and in some cases even reproduces the exact answer. From evaluating the test set and the made up examples, we can see that the model is able to come up with sentences with a meaningful structure. Most of the time it correctly responds to ”why”, ”what” or ”how” questions - the responses aren't necessarily funny, but their structure is correct and sometimes they make sense semantically.

From only about 30000 examples, the model has learned some sort of a language model, and even though it didn't learn humor, it sometimes comes up with sentences that can be considered funny. Humor is of course a very abstract, high-level concept, which relies on understanding the world. Furthermore, in jokes we sometimes produce completely new words, and an RNN where the tokens are words isn't able to do that. A model that will successfully learn humor and will require a much more robust dataset, probably some sort of a way to understand the world, and a much stronger paradigm than a simple LSTM with attention.

All code and data is available on https://github.com/jiriroz/JokeGeneratorSeq2Seq.

8 References

1. https://arxiv.org/pdf/1409.3215.pdf
2. http://cs.stanford.edu/people/karpathy/cvpr2015.pdf
3. https://arxiv.org/pdf/1412.7449v3.pdf
4. https://arxiv.org/pdf/1409.0473.pdf
5. https://www.tensorflow.org/versions/master/tutorials/seq2seq/