SI 630: Homework 5 – Poetry Generation

Version 1.0

Due: Wednesday, April 17, **4:30pm** (part 1),
Thursday, April 18, **5:30pm** (part 2)

1 Introduction

Does a rose by any other name smell quite as sweet?[1] Ah, poetry, one of the highest forms of expression in a language. But what makes a good poem? Is the poem's meaning in the mind of the author or is meaning dependent entirely upon a reader's interpretation—or is constructed through social exercise? Truly, a pressing philosophical conundrum Wittgenstein [2009]. While others debate such questions, we will forge ahead and generate poetry the old fashion way: using machine learning to generate tons of poems and cherry pick the best.

In Homework 5, you will develop a poetry generation system. The homework in general is open-ended and you can take a variety of approaches to generate poetry to your suiting. Further, recognizing this homework comes at the end of the semester, we have also provided two levels of engagement to reflect specific implementations that are different amounts time (and credit).

Homework 5 has the following learning goals:

1. Build a working language model, preferably a *neural* language model
2. Learn how to add structure and/or outside information to a model
3. Gain a basic familiarity with text generation systems

2 Data and Training

We have provided training data in the forms of existing poetry from both amateur and professional poets. The data is provided in two parts. The first part is `lines.json`, which is a long list of lines of poetry extract from historical sources. Each line of poetry is represented by a JSON object, with one object per line in the archive. The value for the `s` key is the line of poetry itself, and the value for the `gid` is an ID for the original source where the line came from. Lines are reported in their original order in the source. The second source, `poems.json` has one line per poem, where each poem is in a JSON object to preserve its whitespace. This dataset will be uploaded to canvas this week. You are welcome to use as much or as little of these sources as you want to generate your poems.

Your ultimate job is to use this data to train your model. There are two options for your implementation:

[1] `https://en.wikipedia.org/wiki/A_rose_by_any_other_name_would_smell_as_sweet`

1. For full credit: Implement your language model using PyTorch using an LSTM. There are many tutorials for how to do this and you are welcome to follow them, though all code should be your own.
2. For *half* credit: Use a character or word-based language model like we discussed in class and provided code for earlier. This option will still require a bit of work on your part to deal with unknown words and smoothing, but it should be easier.

Poetry can take many forms (free verse, sonnets, rhyming pentameter, haiku, etc.) and you are welcome to try to adapt your model to specialize in one particular kind of poetry or just use what's in the data.

Your model too is up to you. Choices to consider:

1. How many neurons in the network (smaller will train faster)
2. How many layers (you can have one LSTM feed into another!)
3. Do you use off-the-shelf embeddings or train them yourself or just let the model learn the embeddings at the same time?
4. How do you want to train (which learning method (e.g., SGD?) and how big is your batch size?)
5. Can you add some knowledge of pronunciation[2] to the word embedding representation? (Hint: remember that you can concatenate embeddings together)
6. Can you add some knowledge of writing structure? Remember, white space actually can matter in poetry (a Haiku without line breaks doesn't work the same way).
7. Can you bias the model towards generating poems in a particular domain or using certain kinds of words (e.g., only concrete terms, no verbs, etc.)?

Note that in the descriptions and tutorials you'll find for language modeling, you'll see talk of "packing" and "padding" with pytorch methods. These techniques are *optimizations* and you can safely ignore them and get a working model without them.[3] That said, if you finish early and want to try something out, learning how to effectively squish sequences for deep learning can give a decent speed up (3-10x).

3 Poetry Generation

The ultimate goal for this homework is to have you train a neural language model on a body of corpus and have it generate poems Your homework will consist of generating poems under two conditions. Note that Part 1 is due *one hour before class*.

3.1 Part 1: Random Generation

In part 1, you'll generate two poems of your choosing from the model and provide a reproducible program (Jupyter notebook or `.py` file) that shows us how to generate the same poems. Even

[2]Example library: `https://github.com/aparrish/gen-text-workshop/blob/master/cmu_pronouncing_dictionary_notes.md`

[3]Technically speaking, they help cram more data onto your GPUs/CPUs so that the network doesn't make useless computations.

though these models are "random," you can fix the randomness of the model by manually setting PyTorch's and numpy's random seeds to some number and then running your model.

Aside from providing some way of exactly reproducing your setup (we should be able to get the same poem (you can test this too)), how you generate your poem is mostly up to you! We do allow you to do your own prompt-base generation (see Part 2) but you can provide at most three prompt words.

3.2 Part 2: Prompt-based Generation

We have provided you with a few fun prompts to see what kind of poems your model might generate. Provide these prompts to your trained model and submit the response when fixing your model with PyTorch's and numpy's random seeds 1234. This should let us generate the same poem if we re-ran your code on the same trained model.

1. How to
2. Goodbye
3. On my home planet
4. I have known
5. Please fix this
6. Do you like sweaters ?
7. Meow meow meow

4 Submission

Please upload the following to Canvas by the deadlines:

1. by Wednesday March 27 at 4:30pm, upload two files to canvas named "uniqname-poem1.txt" and "uniqname-poem2.txt" along with the python code you used to generate them.
2. by Thursday, March 28, upload a PDF or Word file with your model's generations for each of the 7 prompts, as well as code that generates each.

Code should be submitted for any `.py` file or Jupyter notebook you modified. Please upload your code and response-document separately. We reserve the right to run any code you submit; **code that does not run or produces different poetry outputs will receive a zero**.

5 Academic Honesty

Unless otherwise specified in an assignment all submitted work must be your own, original work. Any excerpts, statements, or phrases from the work of others must be clearly identified as a quotation, and a proper citation provided. Any violation of the University's policies on Academic and Professional Integrity may result in serious penalties, which might range from failing an assignment, to failing a course, to being expelled from the program. Violations of academic and professional integrity will be reported to Student Affairs. Consequences impacting assignment or course grades are determined by the faculty instructor; additional sanctions may be imposed.

References

Ludwig Wittgenstein. *Philosophical investigations*. John Wiley & Sons, 2009.