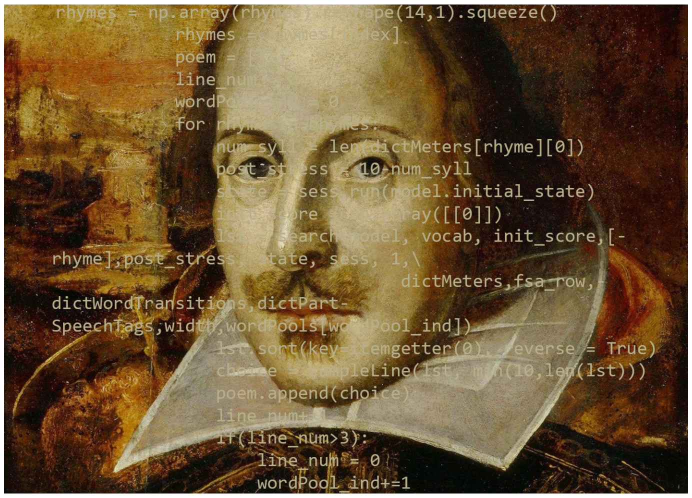
Machine Learning **S2023**

Problem Set 3

Handed Out: March 7th, 2023 *Due: March* 30th, 2023



```
rhymes = np.array(rhymes).reshape(14,1).squeeze()
        rhymes = rhymes[index]
        poem = [illegible]
        line_num [illegible]
        wordPo[illegible] = 0
        for rhyme in rhymes:
            num_syll = len(dictMeters[rhyme][0])
            post_stress = 10-num_syll
            state = sess.run(model.initial_state)
            init_score = np.array([[0]])
            lst = search(model, vocab, init_score,[-
rhyme],post_stress, state, sess, 1,\
                                dictMeters,fsa_row,-
dictWordTransitions,dictPart-
SpeechTags,width,wordPools[wordPool_ind])
            lst.sort(key=itemgetter(0), reverse = True)
            choice = sampleLine(lst, min(10,len(lst)))
            poem.append(choice)
            line_num+=1
            if(line_num>3):
                line_num = 0
                wordPool_ind+=1
```

In this assignment you will use recurrent neural networks to do the impossible: create new plays by William Shakespeare. The fact that he's famously dead is insignificant against the power of machine learning as long as we have some examples of his writing. Lucky for us, there are plenty of examples primed and ready for the taking!

For this assignment, I have provided you with a small subset of William Shakespeare plays. This dataset, named tiny-shakespeare.txt, contains 40,000 lines of Shakespearean text. In this assignment, you will use this text to train a recurrent neural network to generate new lines of Shakespearean text.

You will submit a writeup in Word or PDF that summarizes your results and all code as a zip file. Submit the writeup (with attached source code) to the Canvas submission locker before 11:59pm on the due date.

Create a char-RNN to Generate Shakespearean Text (80 points)

The first part of this assignment, create a character-based recurrent neural network to learn based on the provided dataset. This RNN should take, as input, a character and attempt to predict the next character. Since implementing a RNN by scratch is a bit tricky, I'm allowing you to use any additional libraries you would like for this task. I highly encourage you to make use of some of the more popular ML libraries out there: Tensorflow+Keras, and PyTorch.

You are also allowed to use any data preprocessing, architecture, loss, or optimizer that you'd like. Selecting these is all relatively straight forward when using these libraries, so I encourage you to experiment around with different settings.

In your writeup you need to report important implementation details (such as your values for the above settings), your training regimen (how many epochs, batch size, etc.) and your final training loss. In addition, I want you to include a writing sample from your network. Just generate somewhere between 50-100 characters based on some starting sequence. Really I just want to get a sense for what your network can write.

For example, here's the output of my network after 10 epochs of training when starting with the characters "Q U E E N :"

QUEEN:
I mean, I pollove.

EDWARD:
Who must I set me. I my daughter careless
To gaintly conjurion, none, beat to make
her, an is it
not to support his double advestol hasfe,
And black go villains, having cride, 'tis flownds.
Drawing but my soul, my royal brotherly,
As I go, and now 'tis not sir; and my self-is land
Hath left in a man will be my still,
Think those that sights ere dishome to my affair.
Now bare such lows,
And knows much bound abobe; dear Paris, good nay,
I shall about thee what nerefore now be said; his gracious wor show that
well we him to-morrow thou canst.

ISABELLA:
Wash, speak.

Lord Marshal
My son to you, or I'll peris summer ten bole: they that loves me,
Of morning love-pelD:
Now, poor tormoners, with an eavourable.
Madam, I say, by me, and he should kiss the mark;
This is a bitter gut-the asple:
Thing by their welcome suffer and his best nettle.

DERBY:
No, but not a trant that upon the sun.
Adiem, nothing but we know they seek of this

That heaven that embrace'd hasten

Overall, my loss was 1.2422 after 10 epochs of training using sparse categorical crossentropy provided by keras.

Details:

- You are free to use any supporting library you'd like for this!
- All code should be written in Python3.
- Remember that this is a character-based RNN. This means you'll need to, first, determine all possible characters that are used in the dataset. Use this to create your RNN input and output.
- In your writeup, remember to list your architecture, training loss, and a sample of your network's output.
- You are free to design your RNN however you'd like. This includes using more complex RNN types such as LSTMs or GRUs.
- Explain any implementation choices you had to make in the final report.

And now, a new section: TIPS FOR GETTING STARTED!

- The first thing you want to do, probably before you even set up your RNN is to process your input data. Create your vocabulary, which should contain every single character you want your RNN to be able to generate.
- I find it useful to set up 2 structures to handle this, one structure that can convert a character to an integer (will be useful when setting up one-hot vectors), and one to convert back from an integer to a character (will be useful in generating output).
- Next, you'll need some code that can generate one-hot tensors. I've found the easiest way to do this involves converting your input text into a sequence of integer values (that method you wrote earlier will come in handy here) and then using those integer values to determine what element of your one-hot needs to be set to 1. Remember, your one-hots will always be the same length as your vocabulary.l
- This is probably the most critical advice, at least if you're using PyTorch: your input and desired output will be offset by 1. Your input sequence needs to only contain values up to the last character, so if you have a sequence of length n, your input should only contain n-1 values (the last one is cut off). Your desired output values should not contain a first value. This offset makes your input and desired output line up with the RNN structure. Your first input will be paired with the first desired output (which would technically be the second character in the sequence), and so on and so forth.

Presentation (20 points)

Your report must be complete and clear. A few key points to remember:

- Complete: the report does not need to be long, but should include everything that was requested.
- Clear: your grammar should be correct and it should be easy to follow what you're saying.
- Concise: I sometimes print out reports to ease grading, don't make figures larger than they need to be. Graphics and text should be large enough to get the point across, but not much larger.
- Credit (partial): if you are not able to get something working, or unable to generate a particular figure, explain why in your report. If you don't explain, I can't give partial credit.

Bonus: Word-Based RNN for Shakespeare (10 Points, required for graduate students)

In this part of the assignment, you will extend your RNN to be based on full words instead of characters. This should only require you to alter how you determine the size and composition of your vocabulary. If you'd like, however, you can choose to do additional natural language processing on the input (tokenization of strings, stemming, etc.). As with your character-based RNN, report training loss, a writing sample, and the architecture of your network.

Details:

- Same rules apply as in Part 1.
- Explain any implementation choices you had to make in the final report.
- Remember to include training loss values along with a discussion of the architecture that you used as well as a writing sample!