# homework8

April 9, 2019

## 1 Homework 8 - Berkeley STAT 157

**Your name: XX, SID YY, teammates A,B,C** (Please add your name, SID and teammates to ease Ryan and Rachel to grade.)

**Please submit your homework through gradescope**

Handout 4/9/2019, due 4/16/2019 by 4pm.

This homework deals with sequence models for text and numbers. Due to the computational cost, we strongly encourage you to implement this on a GPU enabled machine. To make things a bit more interesting we will use a larger text collection here - Shakespeare's collected works which are freely downloadable at Project Gutenberg.

**This is teamwork.**

### 1.1 Prerequisites - Load Data

```
In [1]: import urllib3
        import collections
        import re
        shakespeare = 'http://www.gutenberg.org/files/100/100-0.txt'

        http = urllib3.PoolManager()
        text = http.request('GET', shakespeare).data.decode('utf-8')
        raw_dataset = ' '.join(re.sub('[^A-Za-z]+', ' ', text).lower().split())

        print('number of characters: ', len(raw_dataset))
        print(raw_dataset[0:70])
```

```
number of characters:  5032359
project gutenberg s the complete works of william shakespeare by willi
```

This dataset is quite a bit bigger than the time machine (5 million vs. 160k). For convenience we also include the remaining preprocessing steps. A bigger dataset will allow us to generate more meaningful models.

```
In [5]: idx_to_char = list(set(raw_dataset))
        char_to_idx = dict([(char, i) for i, char in enumerate(idx_to_char)])
        vocab_size = len(char_to_idx)
        corpus_indices = [char_to_idx[char] for char in raw_dataset]
        sample = corpus_indices[:20]
        print('chars:', ''.join([idx_to_char[idx] for idx in sample]))
        print('indices:', sample)

        train_indices = corpus_indices[:-100000]
        test_indices = corpus_indices[-100000:]
```

```
chars: project gutenberg s
indices: [21, 19, 22, 10, 26, 24, 13, 3, 12, 9, 13, 26, 20, 4, 26, 19, 12, 3, 5, 3]
```

Lastly we import other useful libraries to help you getting started.

```
In [6]: import d2l
        import math
        from mxnet import autograd, gluon, init, nd
        from mxnet.gluon import loss as gloss, nn, rnn
        import time
```

```
4932359 100000
```

### 1.2 1. Train Recurrent Latent Variable Models

Train a number of different latent variable models using `train_indices` to assess their performance. By default pick 256 dimensions for the hidden units. You can use the codes provided in the class. Also, we strongly encourage you to use the Gluon implementation since it's a lot faster than building it from scratch.

1. Train a single-layer RNN (with latent variables).
2. Train a single-layer GRU.
3. Train a single-layer LSTM.
4. Train a two-layer LSTM.

How low can you drive the perplexity? Can you reproduce some of Shakespeare's finest writing (generate 200 characters). Start the sequence generator with `But Brutus is an honorable man`. Experiment with a number of settings:

- Number of hidden units.
- Embedding length.
- Gradient clipping.
- Number of iterations.
- Learning rate.

**Save** the models (at least in memory since you'll need them in the next exercise.

### 1.3 2. Test Error

So far we measured perplexity only on the training set.

1. Implement a perplexity calculator that does not involve training.
2. Compute the perplexity of the best models in each of the 4 categories on the test set. By how much does it differ?

## 1.4 3. $N$-Gram Model

So far we only considered latent variable models. Let's see what happens if we use a regular $N$-gram model and an autoregressive setting. That is, we aim to predict the next character given the current characters one character at a time. For this implement the following:

1. Split data into $(x, y)$ pairs as before, just that we now use very short subsequences, e.g. only 5 characters. That is, `But Brutus` turns into the tuples `((But B, r), (ut Br, u), (t Bru, t), ( Brut, u), (Brutu, s))`.
2. Use one-hot encoding for each character separately and combine them all.
    - In one case use a sequential encoding to obtain an embedding proportional to the length of the sequence.
    - Use a bag of characters encoding that sums over all occurrences.
3. Implement an MLP with one hidden layer and 256 hidden units.
4. Train it to output the next character.

How accurate is the model? How does the number of operations and weights compare to an RNN, a GRU and an LSTM discussed above?