Data loading

- At the beginning of each epoch do
  - Fetch all 579 articles
  - Shuffle their order
  - Combine them into one big article
  - Leave out anything that doesn’t nicely fit in the number of batches
    - E.g if there batch size is 32 and there are 540 entries in dataset only use 512 and leave the 28 words
  - Transform your data as follows

50	20	39	45	289	31905	248	59	2231	98

Sequence

Batch dimensison

50	45	248
20	289	59
39	31905	2231

Assuming a batch size of 3

Notice the last entry was left out because it doesn't cleanly fit

Assuming batch_first=False in Pytorch. If you prefer batch_first=True easy to modify…

For each data fetching request

- Generate a random value for sequence length to use from one of the two distributions described in the paper
  - Distribution 1: Gaussian normal of mean 70 standard deviation of 5
  - Distribution 2: Gaussian normal of mean 35 standard deviation of 5
- 95% of the sequence length values must come from Distribution 1
- The rest 25% (☺) from Distribution 2

Possible way of generating a random sequence length

- At each iteration
  - Generate a 0 with probability of 0.95 and 0 with probability 0.25 (still kidding)
  - If zero generate another random number from distribution 1 and use it as your sequence length
  - If one generate from distribution 2
- Gaussian distribution has infinite support
  - You need to make sure that generated number is greater than zero!

Keep track of the point to which you have read: Initially

Keep track of the point to which you have read: After reading the first 3 entries

Keep track of the point to which you have read: After reading the following 4 entries

What happens if sequence length of 6 was generated in this step?

How to implement what we discussed

```
class Foo:
    def __init__(self):
        self.data = [1, 2, 3, 4, 5, 6, 7, 8, 9]

    def __iter__(self):
        print("Whatever we do here executes once")
        i = -1
        while i < len(self.data) - 1:
            i += 1
            yield self.data[i]

if __name__ == '__main__':

    a = Foo()
    for entry in a:
        print(entry)
```

Model

- 3-Layer LSTM with 1150 hidden units
- Embedding dim of 400
- You may want to use three cascaded LSTMs instead of 1 3-layer LSTM
  - Useful for some of the regularizations
  - This means at each point you will need to keep track of three hidden states

- Initialize the weights according to the paper specification

As little as one of let you pass the 100% cutoff

- Apply locked dropout between LSTM layers
- Apply embedding dropout
- Apply weight decay
- Tie the weights of the embedding and the output layer
- Activity regularization
- Temporal activity regularization

Implementation

- https://pytorchnlp.readthedocs.io/en/latest/

Training

- Paper uses NT-ASGD
  - Not required to achieve the desired accuracy
    - Pytorch only has ASGD
    - Stick with the optimizer you are most comfortable with