trying to make sense of

‘LSTMs Exploit Linguistic Attributes of Data’ by Liu et al. (2018)

Johannes Bernhard & Sasha Mayn

Outline

1. Introduction & Question under Investigation
2. The Memorization Task
    a. Setup
    b. Datasets Used
    c. Results
3. Analysis & Conjectures
4. Conclusions and Outstanding Questions

Introduction

- RNNs are general models of sequential data
    - Not designed specifically to capture language
- Do LSTMs perform better when the following (linguistic) properties are present?
    - Hierarchical structure
    - Dependencies
    - Zipfian frequency distribution
- LSTMs trained on linguistic data are able to memorize sequences of greater length!
    - Why? Stay tuned!

The Task

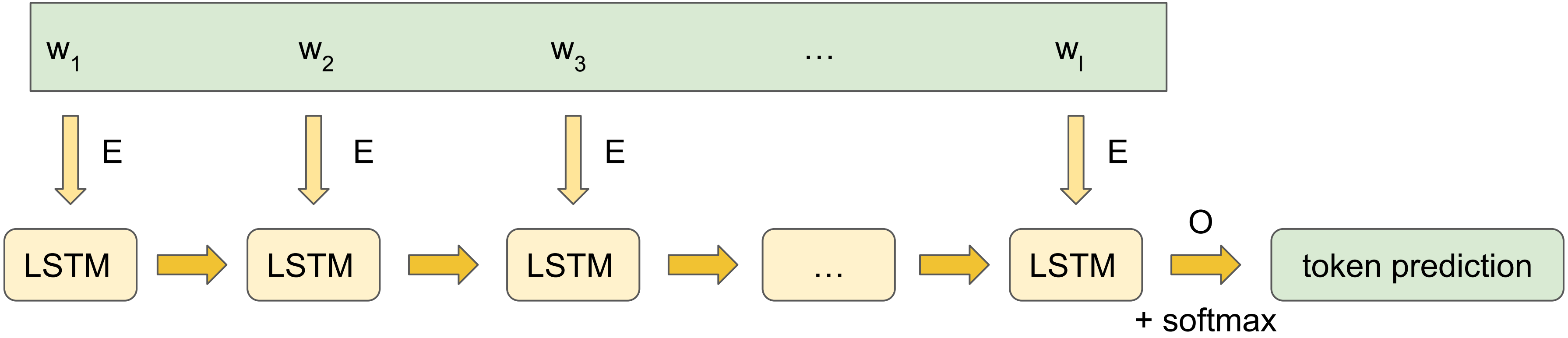
- Simple non-linguistic task: predict the middle token
  - Middle so that the LSTM can't cheat
- All sequences of equal length (train & test)
- Evaluated on 100 uniformly sampled rarest words
  - Can't just output 'the' every time!

["dog", "if", "joy", "walking", "the", "a", "the", "verdant", "place"]

["the"]

Setup: LSTM sketch

- Tokens embedded with a randomly initialized matrix
- Encoded by a single-layer LSTM
- Output is a probability distribution over the vocabulary



Setup: One does not simply...

- The weights of the embedding matrix & the output projection are **frozen** so that the LSTM doesn't cheat by shifting weights in one of the vector spaces (embedding or output)
- LSTM parameters are the only trainable weights

Setup: freezing and tying weights

Embedding matrix: $E \in \mathbb{R}^{h \times v}$
Output matrix: $O \in \mathbb{R}^{v \times h}$

$$O = E^T$$

$$\mathrm{w} \rightarrow \text{one-hot}(\mathrm{w}) \in \mathbb{R}^v \rightarrow E \times \text{one-hot}(\mathrm{w}) \in \mathbb{R}^h$$

$$(\rightarrow \text{LSTM}) \rightarrow E^T E \times \text{one-hot}(\mathrm{w}) \in \mathbb{R}^v$$

$$E^T E w_{\mathrm{i}} \approx \lambda w_{\mathrm{i}}$$

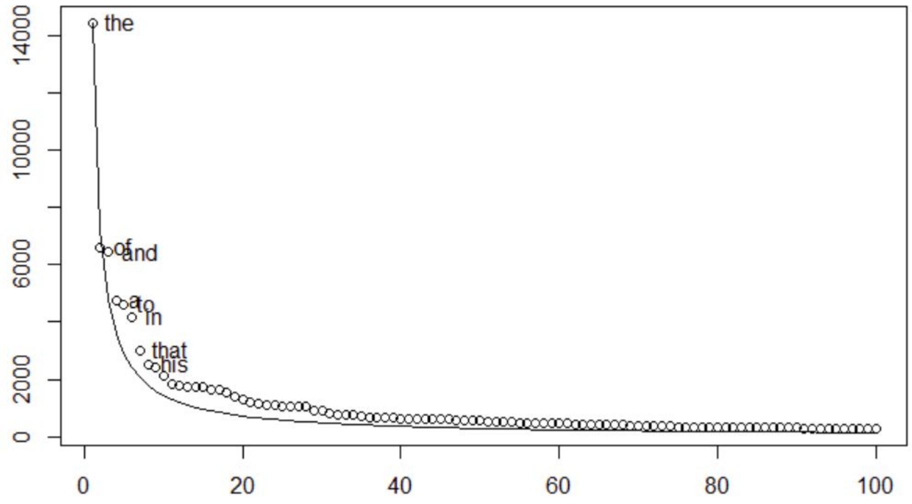
$$(E^T E)_{\mathrm{i,j}} = \begin{cases} \mathrm{e_i} * \mathrm{e_j} = \| e_{\mathrm{i}} \|^2 > 0 \text{ for i=j} \\ \mathrm{e_i} * \mathrm{e_j} \approx 0 \text{ for } \mathrm{i} \neq j \end{cases}$$

Setup: freezing and tying weights

```
>>> E0 = np.random.randn(3,5) #h=3,v=5
>>> E = E0/(E0.max()+0.1) #Random embedding matrix
>>> print(E)
[[ 0.42530099  0.66029249 -0.8340761   0.64709947 -1.10771449]
 [-0.69160407 -0.90618131 -0.64054174  0.3377413  -0.74151355]
 [-0.19313935 -0.72759331  0.91551918  0.58517726 -0.43793254]]
>>> O = E.T #Output matrix
>>> encode_decode = np.matmul(O,E)
>>> print(encode_decode)
[[ 0.69649993  1.04806862 -0.08855489 -0.07139197  0.12630373]
 [ 1.04806862  1.78654274 -0.63641286 -0.30455099  0.25916695]
 [-0.08855489 -0.63641286  1.94415202 -0.2203266   0.99795293]
 [-0.07139197 -0.30455099 -0.2203266   0.87523933 -1.22350937]
 [ 0.12630373  0.25916695  0.99795293 -1.22350937  1.96865864]]
>>> v = [0,1,0,0,0]
>>> out = np.matmul(encode_decode,v)
>>> print(out)
[ 1.04806862  1.78654274 -0.63641286 -0.30455099  0.25916695]
>>> print(softmax(out))
[ 0.25055209  0.52434034  0.04648759  0.06478326  0.11383672]
```

Datasets

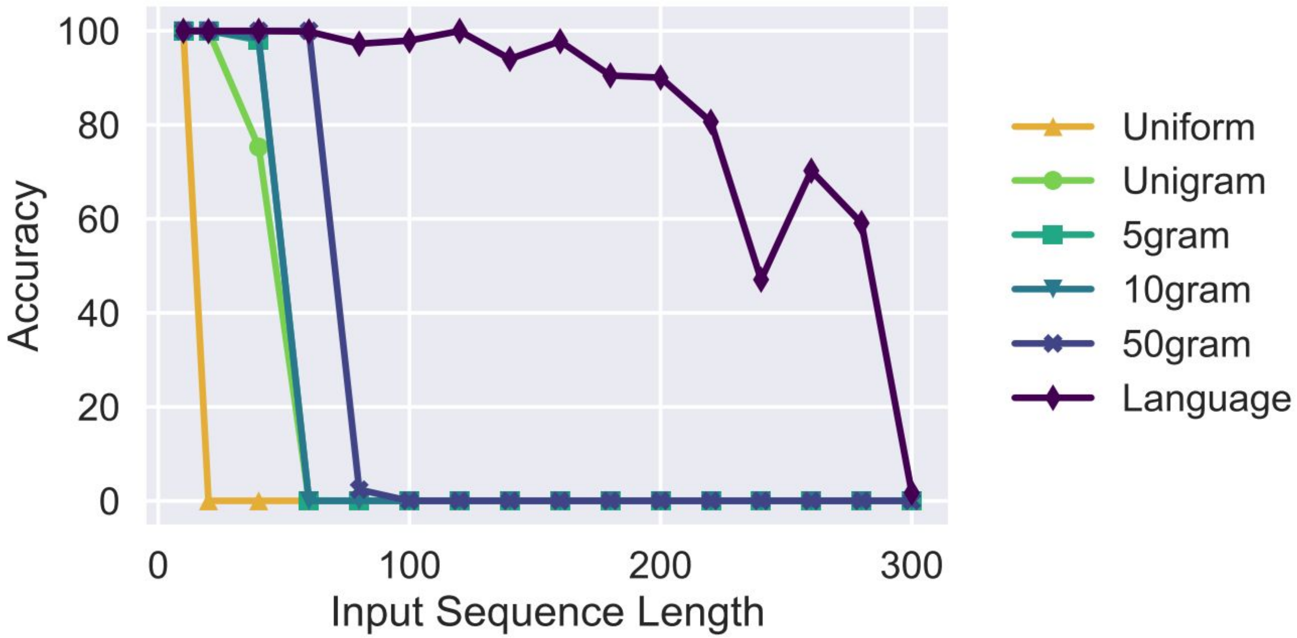
- Uniform
  - Words randomly sampled from a uniform distribution over the vocabulary
  - ‘the’ should appear as frequently as ‘sesquipedalian’
- Unigram
  - Integrate Zipfian token frequencies
- N-gram (N=5,10,50)
  - Permuted chunks of text of length N
- Language
  - Real language
- Our experiments:
  - Uniform with |V|=9
  - Unigram with a less dramatic distribution



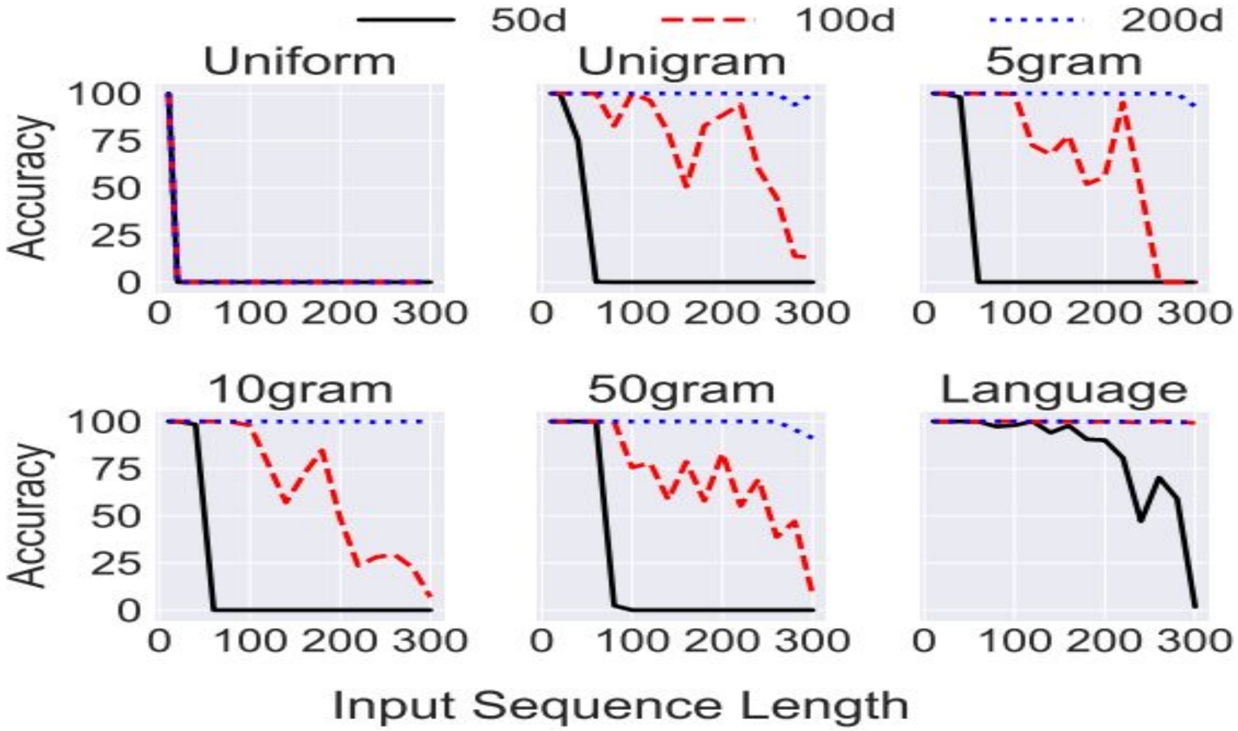
Zipfian distribution (from phys.org)

Results: Comparing the Settings

- 50 hidden units
- Up to a certain threshold, LSTM performs perfectly in all settings



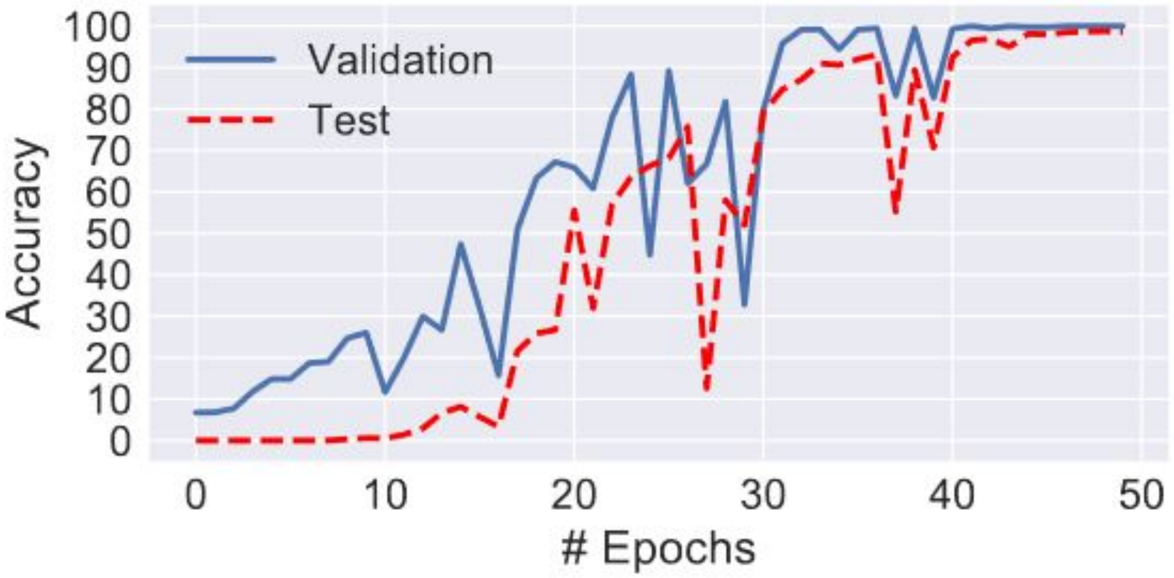
Results: Adding Hidden Units



Accuracy to seq length for different numbers of h.u.

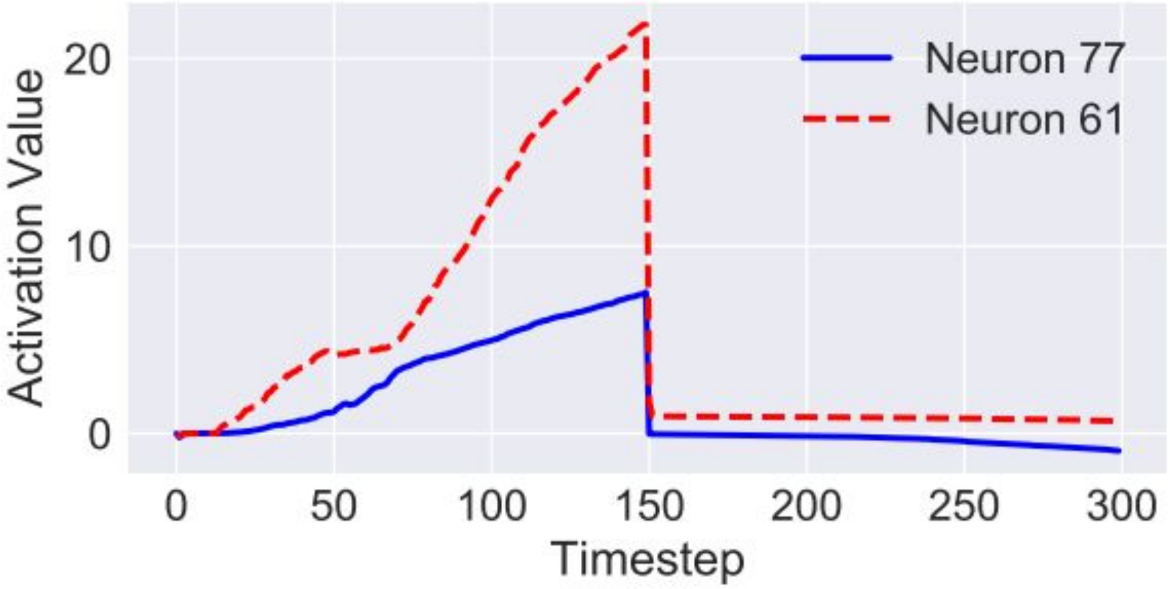
Results: Validation and Test sets

- Exploitation of linguistic features in training data
- Curve convergence indicates true memorization



A Peek Inside the LSTM

- How? Counting!
    a. Count up to the token
    b. Then, consume label
    c. Output said label
- linear regression model to identify “counting” neurons



Activation levels for two neurons, 61 and 77, which appear to have counting behavior

Our Experiments: Uniform & Unigram

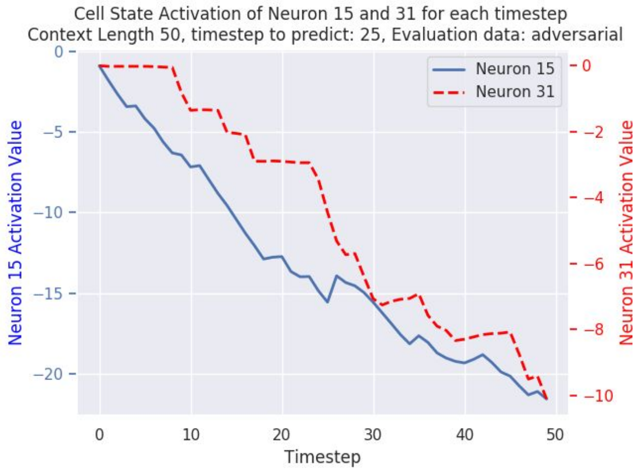
- What is it about unigram that helps the model learn?
- Would a much less dramatic distribution work?

A much simpler task: $|V| = 9$ (digits 0 through 8)

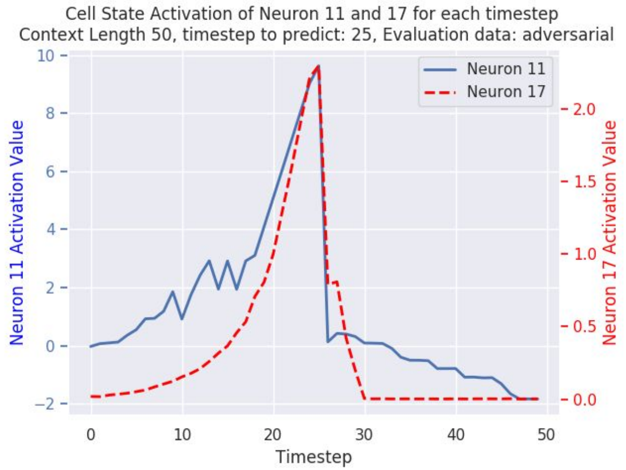
Uniform: all digits equiprobable Validation accuracy: 100% Test accuracy: 82%	Unigram: one digit has the p=0.12, the rest: p=0.11 Validation accuracy: 100% Test accuracy: 100%

Both models learn fast (5 epochs to achieve perfect validation accuracy), but "uniform" overfits

Our Experiments: Uniform & Unigram



Uniform



Unigram

The model doesn’t seem to be employing the same strategy for learning (i.e., no counting behavior exhibited in uniform)

Conclusions

- Uniformly sampled data is only trainable up to seq. length 10
- Language data allows trainability up to seq. length 300

- Specific neurons are used to track timestep information

- Open question as to why the task can be solved on linguistic data
  - additional patterns and structure in language-based data?

Wait...but why?

- The big question: if the task is nonlinguistic/does not rely on structure in the data, why is structure needed to learn it?

- The authors' conjecture:

  > "Linguistic data offers more reasonable, if approximate, pathways to loss minimization, such as counting frequent words and phrases"

- What does that mean exactly? Alternative ideas?

Thank you for your time!

Any questions?