Are Deep Neural Networks the Best Choice for Modeling Source Code?

Prestation by
Disha A Patel

Today we study code as an act of communication.

To understand why, we must look back.

For most of the 20th century, the field of linguistics studied natural languages in a formal framework, using grammars and formal rules.

In the 1970s, the rise of computational power began to change this: some linguists turned to corpus-based studies instead.

These studies added a powerful complementary understanding of how people actually used language.

Among the many resulting insights, perhaps the most interesting one was that people use language in astonishingly predictable and repetitive ways!

This allows the construction of statistical language models that power many of todays language-related innovations, like Alexa, auto-correct.

What are your next steps?

Now, in the last decade, we have seen a boom in data-driven SE research: vast amounts of code and communication about that code have entered the open-source domain, which means that we can now study statistical and behavioral characteristics of what coding actually looks like in the wild; a very similar trend to linguistics.

one of the main observations that followed was that code, too, is highly predictable and can be effectively modeled in ways similar to natural language.

Language Models

In computational linguistics, the discovery of these statistical patterns in natural language led to the development of powerful models to capture their repetition and variation.

Traditionally, these language models where count-based, but more recently deep learning has proven successful at capturing more latent patterns in text, especially recurrent neural networks.

Challenges

– Vocabulary

– Evolution

+ Many Scopes

Structure

- A bit more background
- Our contributions
- Where do we go?

Language Models

Count-based (n-gram):

$$p(\text{"Fransisco"}|\text{"San"}) \approx \frac{count(\text{"San Fransisco"})}{count(\text{"San"})}$$

$$p(\text{"Fransisco"}|\text{"Go to San"})?$$

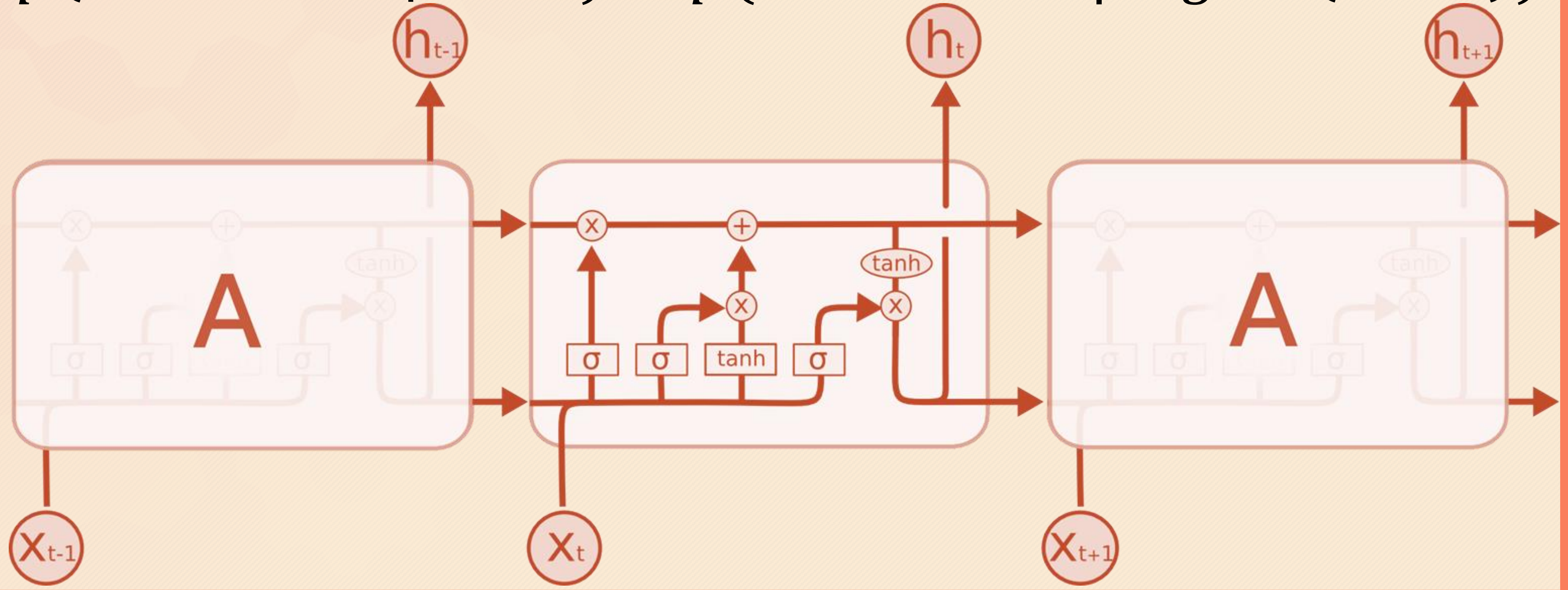
$$= \lambda * \frac{count(\text{"Go to San Fransisco"})}{count(\text{"Go to San"})}$$

$$+ (1\ - \lambda) * p(\text{"Fransisco"}|\text{"to San"})$$

Language Models

Deep Learning (RNN):

$$p(\text{"Fransisco"}|\text{"}San\text{"}) \approx p(\text{"Fransisco"}\ |\ digest(\text{"San"}))$$



Evaluating Models

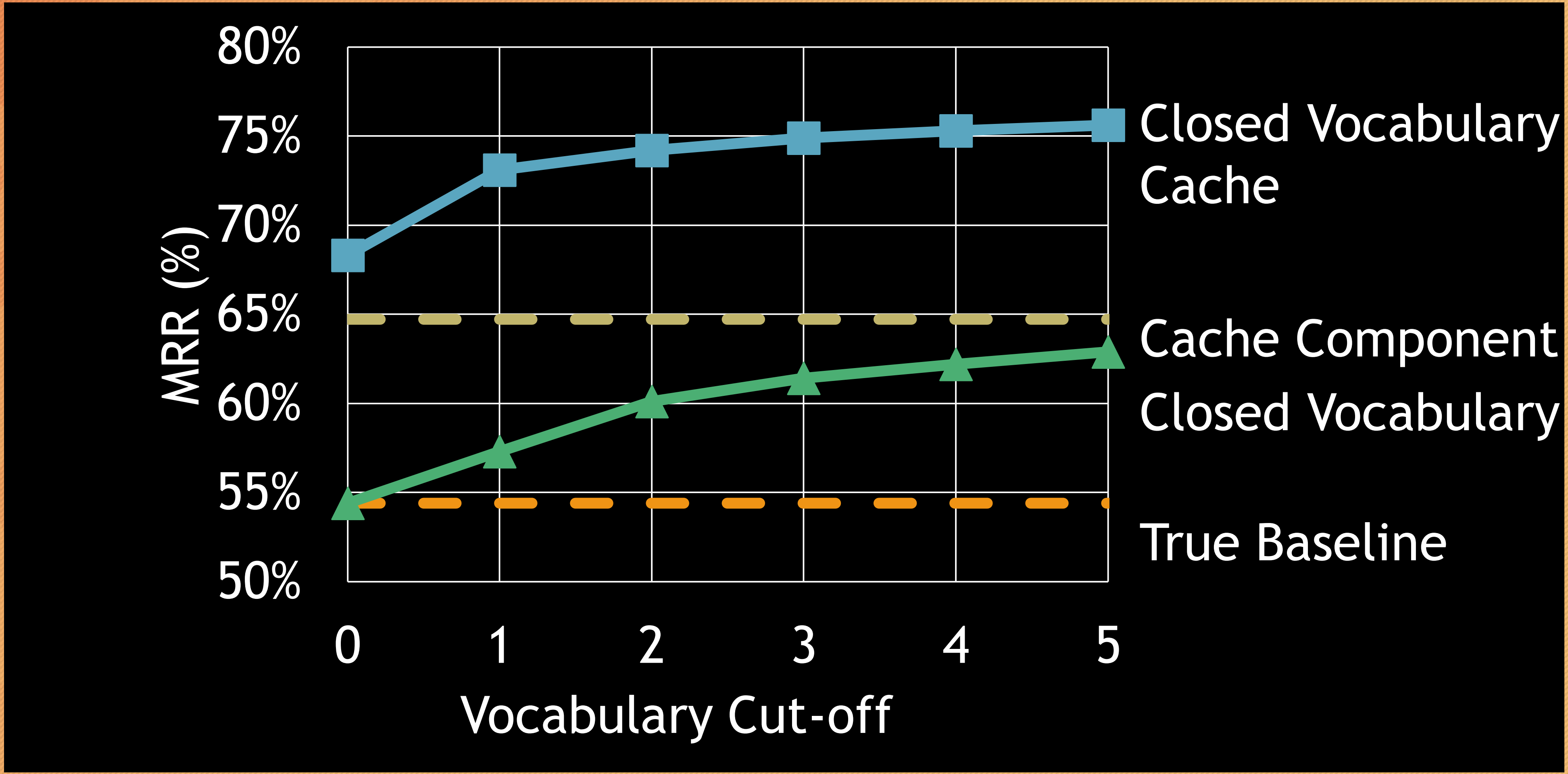
Entropy (intrinsic):

$$H_p(\text{"Fransisco"}|\text{"San"}) = \log_2 p(\text{"Fransisco"} \mid \text{"San "})$$

Mean Reciprocal Ranking (extrinsic):

$$MRR(\text{"Fransisco"}|\text{"San"}) = 1/rank(\text{"Fransisco"}|\text{"San"})$$

Vocabulary


80%
75%
70%
65%
60%
55%
50%
MRR (%)
0
1
2
3
4
5
Vocabulary Cut-off
Closed Vocabulary Cache
Cache Component
Closed Vocabulary
True Baseline

Where next?

Exploit what we Know

Tailor DNN for Code

Explore Linguistics of Code

Implicit Language Models

- Recurrent neural networks (RNN)
- Long short-term Memory

Recurrent neural networks (RNN)

Maintain a hidden state vector to capture a digested representation of the current context, as they scan forward, token by token. Learned parameters both read out this vector (e.g., to predict, score a token) and update this vector upon seeing the next token. These models are quite effective when trained with sufficient data.

Long short-term Memory networks (LSTM)

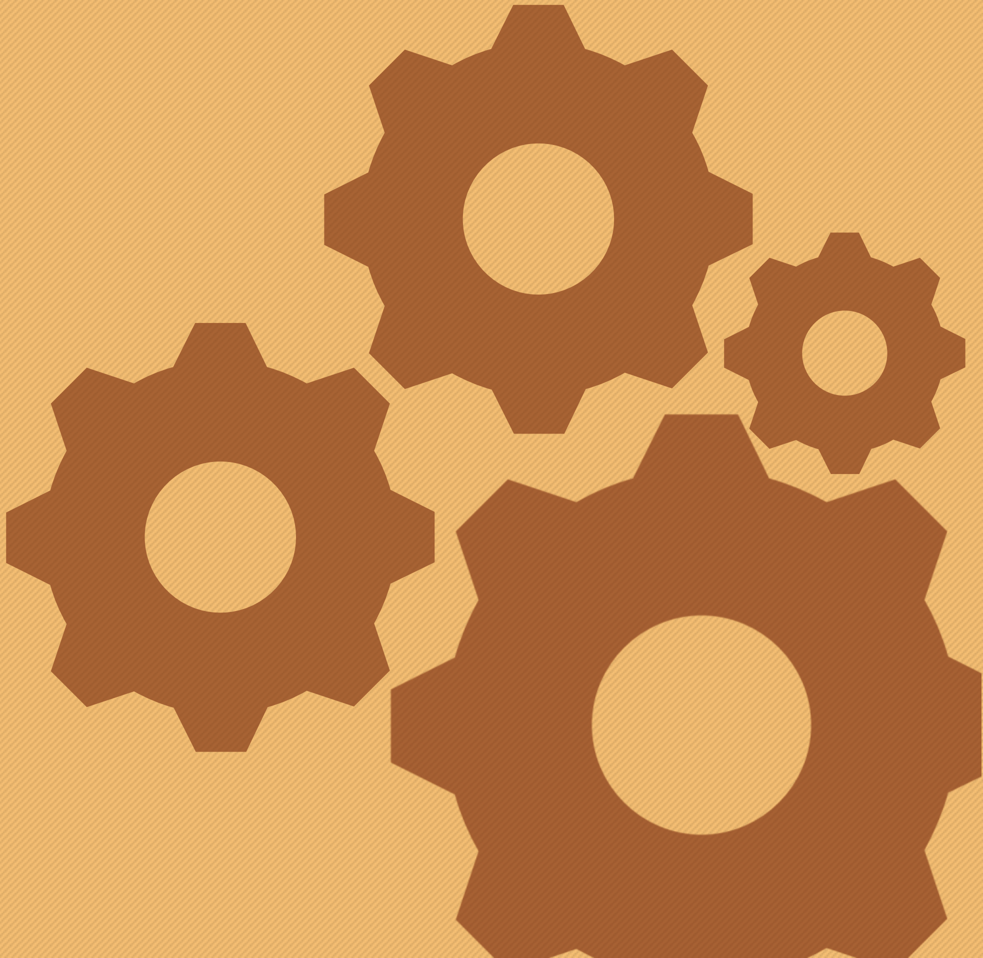
Long short-term Memory networks (LSTM): are extensions of RNNs which can be trained to selectively “forget" information from the hidden state, thus allowing room to take in more important information.

Cache Models

Nested models only apply when the models have a view of the context of the file to be modeled, either incremental or full. Thus, we demonstrate results in two settings: dynamically updated models, which add each file to their training corpus after modeling it, and software maintenance models, which are allowed to see each file in the project excluding the one to be modeled, as well as the nested, scoped directory hierarchy in which the test file occurs

Results

They present results on varying corpora in comparison with traditional N-gram, as well as RNN, and LSTM deep-learning language models, and release all our source code for public use. Their evaluations suggest that carefully adapting N-gram models for source code can yield performance that surpasses even RNN and LSTM based deep-learning models.



Mixing results

- Current statistical language modeling techniques, including deep- learning based models, have proven to be quite effective for source code.

- Evaluations suggest that carefully adapting N-gram models for source code can yield performance that surpasses even RNN and LSTM based deep-learning models.

Questions

Thank you