Improving Neural Language Models with A Continuous Cache

Edouard Grave, Armand Joulin, Nicolas Usunier

Facebook AI Research

Akash Govindarajula & Jakub Wasylkowski (Group 5)

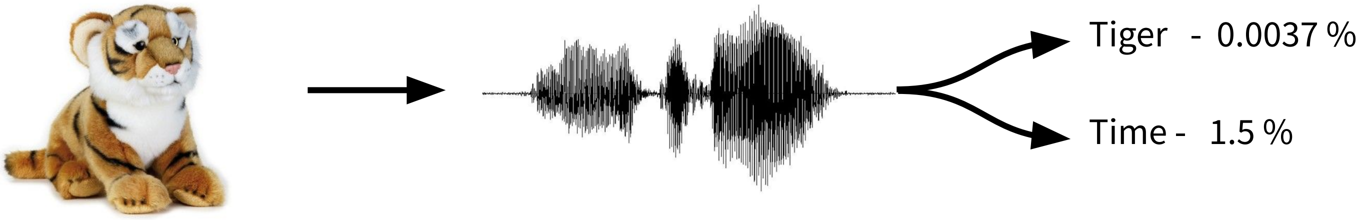
Contents

- Intro
  - Overview of LM, RNNs and cache models
- Neural Cache Model
  - Hidden Representations
  - Architecture, training details
- Related concepts(?)
- Experiments
  - Small & Medium scale
  - Lambada
- Results and Conclusion

The problem with uncommon vocabulary

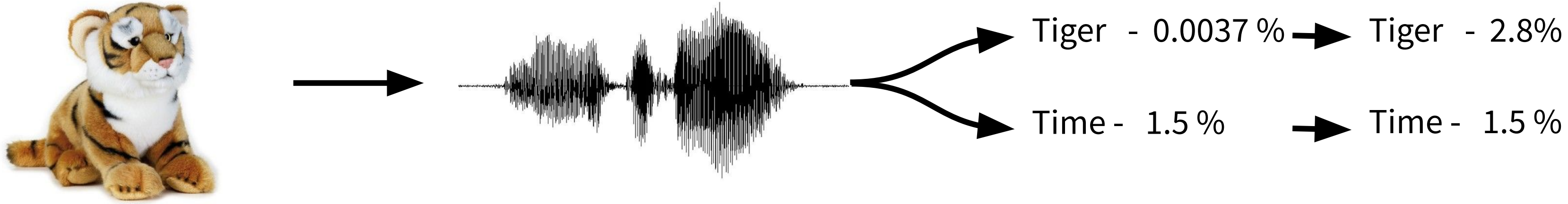
“Traditional Language Models lack the capacity to adapt to their recent history, limiting their application to dynamic environments”

Rare words are discriminated even if repeated in-context.



Adding a static cache to n-grams

- External memory to store new information
- Can predict Out Of Vocabulary words after one appearance
- Memory intensive - read/write operations require training
- Forced limits on cache size and memory usage



Proposition

A continuous Neural Cache Model

- Lightweight alternative to static cache.
- Dynamic adaptation, no need to train.
- Cheap memory overhead, one dot product with the hidden activation.
- Independent of the underlying model.

Reminders

Probability distribution over sequences of words

$$p(x_1, ..., x_T) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, ..., x_1).$$

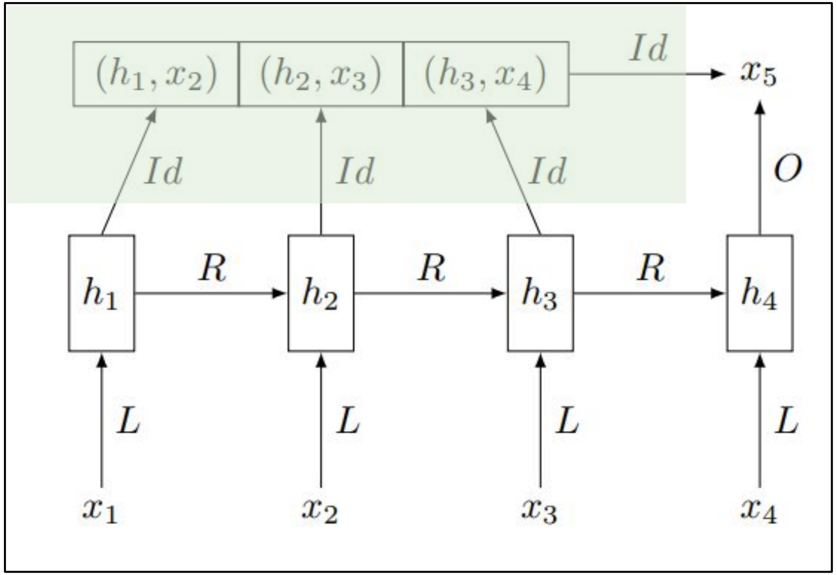
RNN probability given history vector h

$$p_{vocab}(w \mid x_t, ..., x_1) \propto \exp(h_t^\top o_w).$$

$$h_t = \Phi\left(x_t, h_{t-1}\right)$$

Adding a neural cache

Using h to create a probability distribution over the words in the cache.



$$p_{cache}(w \mid h_{1..t},\ x_{1..t}) \propto \sum_{i=1}^{t-1} \mathbb{1}_{\{w=x_{i+1}\}} \exp(\theta h_t^\top h_i)$$

P_{cache} is the probability to retrieve the word w from the memory given the query h_t where the desired answer is x_{t+1}.

Computing word probability

Method 1: Linear interpolation of the standard model with the cache

$$p(w \mid h_{1..t},\, x_{1..t}) = (1-\lambda)p_{vocab}(w \mid h_t) + \lambda p_{cache}(w \mid h_{1..t}, x_{1..t})$$

Method 2: Global normalization over the two distributions

$$p(w \mid h_{1..t},\, x_{1..t}) \propto \left(\exp(h_t^\top o_w) + \sum_{i=1}^{t-1} \mathbb{1}_{\{w=x_{i+1}\}} \exp(\theta h_t^\top h_i + \alpha) \right)$$

Related models

- Cache Model
    - Introduced for speech recognition, and later extended to smoothed trigram LMs (*observed lesser perplexity & word errors*).
    - Della Pietra et al. (1992) adapts cache to a general n-gram to satisfy marginal constraints [1] obtained from current document.
- Adaptive Language Models
    - Various models have been introduced before. I. Weighted interpolated models to adapt dynamically, II. Latent Semantic analysis
    - Topic features used with max. Entropy / Recurrent nets; and other proposal maps to use pairs of distant words for long-range dependency capture.

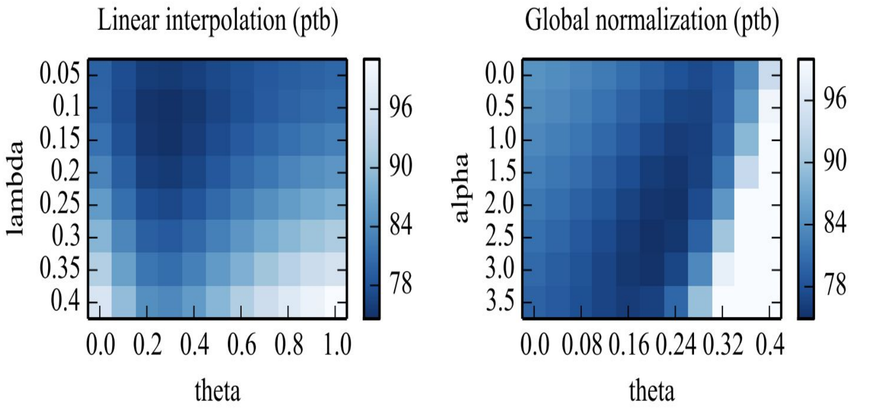
- Memory augmented neural-nets
  - Promising results in sequence prediction (*Mainly storing a representation and using attention mechanism, shows dip in perplexity*).
  - Successfully applied to QA tasks as well (*Attention mechanisms, pointer softmax [machine translation context]*).
  - Pointer sentinel mixture models(2016) cited. In contrast to neural cache model, current hidden activation is used here as representation.

Experiments

- Small scale
  - Pen Tree Bank and wikitext2 datasets
  - Implementation
    - RNN with 1024 LSTM units
    - Regularized dropouts and Adagrad approach
    - Cache sizes on a logarithmic scale
  - Results
    - Perplexity on validation sets, linear performs better than global.
    - Significant improvements at large cache values (*30% over baseline & 12% over smallcahe of 100 words*)

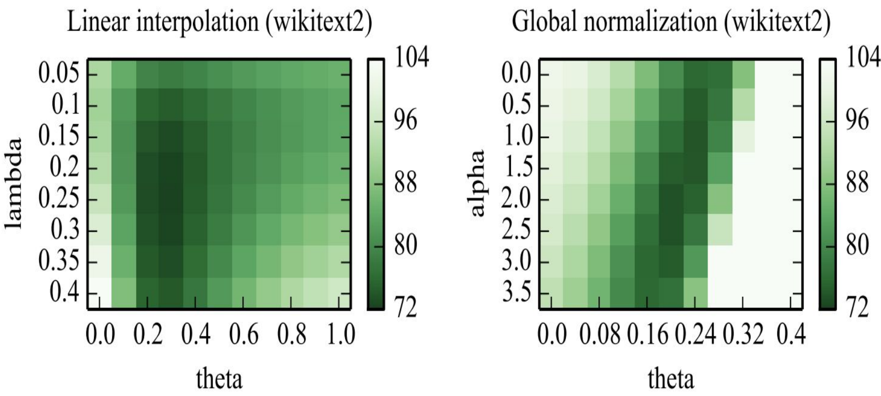
Model	Test PPL
RNN+LSA+KN5+cache (Mikolov & Zweig, 2012)	90.3
LSTM (Zaremba et al., 2014)	78.4
Variational LSTM (Gal & Ghahramani, 2015)	73.4
Recurrent Highway Network (Zilly et al., 2016)	66.0
Pointer Sentinel LSTM (Merity et al., 2016)	70.9
LSTM (our implem.)	82.3
Neural cache model	72.1

Table 1: Test perplexity on the `Penn Tree Bank`.



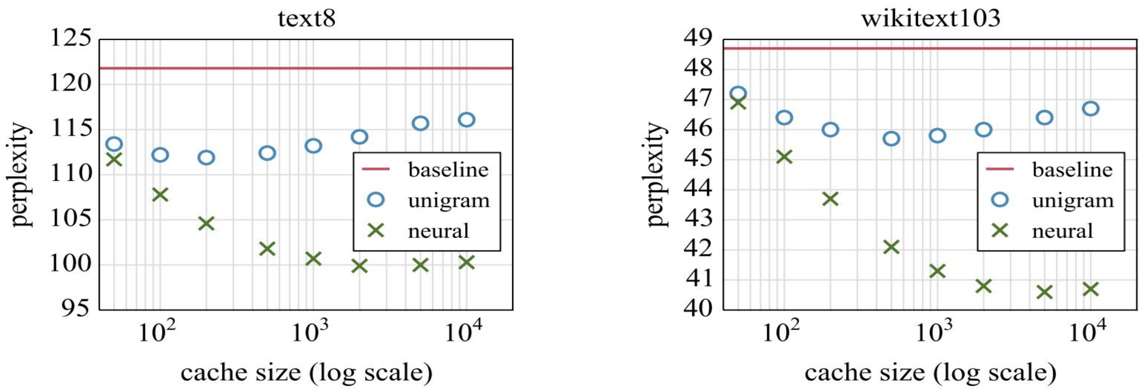
Model	`wikitext2`	`wikitext103`
Zoneout + Variational LSTM (Merity et al., 2016)	100.9	-
Pointer Sentinel LSTM (Merity et al., 2016)	80.8	-
LSTM (our implementation)	99.3	48.7
Neural cache model (size = 100)	81.6	44.8
Neural cache model (size = 2,000)	68.9	40.8

Table 2: Test perplexity on the `wikitext` datasets. The two datasets share the same validation and test sets, making all the results comparable.



- Medium scale

- Implementation
  - Tested on text8 and wikitext103
  - Same settings and adaptive softmax



- Results
  - Our model exploits larger cache, than baseline
  - Perplexity improvement is smaller (16%) over LSTM than that of wikitext2
  - Important to evaluate & compare on large datasets

Lambada

Model	Test
LSTM-500 (Mikolov et al., 2014)	156
SCRNN (Mikolov et al., 2014)	161
MemNN (Sukhbaatar et al., 2015)	147
LSTM-1024 (our implem.)	121.8
Neural cache model	99.9

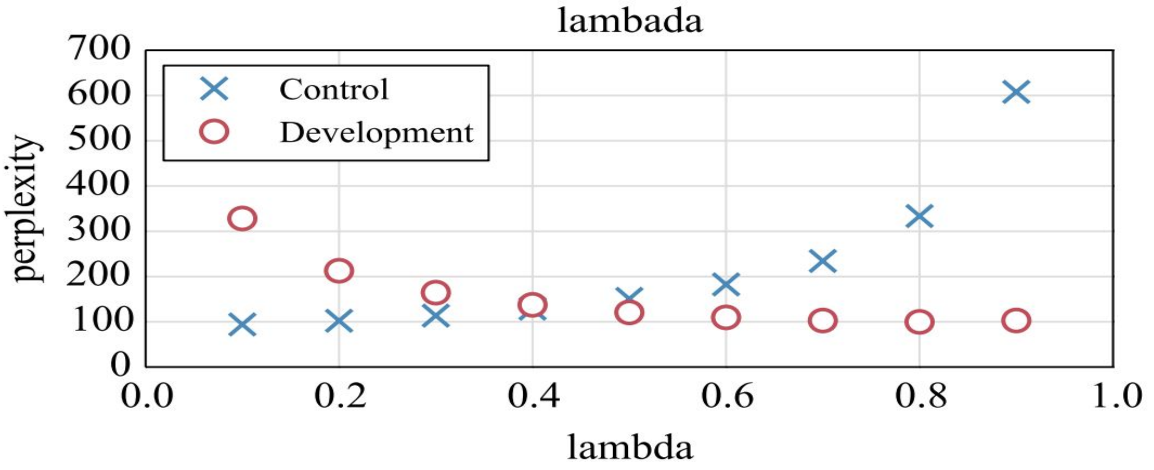
(a) `text8`

Model	Dev	Ctrl
WB5 (Paperno et al., 2016)	3125	285
WB5+cache (Paperno et al., 2016)	768	270
LSTM-512 (Paperno et al., 2016)	5357	149
LSTM-1024 (our implem.)	4088	94
Neural cache model	138	129

(b) `lambada`

Table 3: Perplexity on the `text8` and `lambada` datasets. WB5 stands for 5-gram language model with Witten-Bell smoothing.

- Dataset of short passages extracted from novels
- To predict last word of excerpt
- 200M tokens & vocab. size of ~93k
- Adding neural cache to LSTM baseline amplifies performance on it
    - Variation in best parameter between static model / cache for dev. & control
- Generalization here
    - To adapt parameter based on vec. representation of h_t



Conclusion

- Neural cache model proposed with dynamic updates (*long-term memory*)
- Experiments and Lambada dataset results
  - Show significant gains in performance with external memory component
- Unlike pointer nets, avoids learning the memory lookup component
  - Hence can use larger cache sizes & be applied easily like count-based caches

Questions?