Learning to Create and Reuse Words in Open-Vocabulary Neural Language Modeling

Kazuya Kawakami[♠] **Chris Dyer**[♣] **Phil Blunsom**[♠♣]
[♠]Department of Computer Science, University of Oxford, Oxford, UK
[♣]DeepMind, London, UK
{kazuya.kawakami,phil.blunsom}@cs.ox.ac.uk,cdyer@google.com

Abstract

Fixed-vocabulary language models fail to account for one of the most characteristic statistical facts of natural language: the frequent creation and reuse of new word types. Although character-level language models offer a partial solution in that they can create word types not attested in the training corpus, they do not capture the "bursty" distribution of such words. In this paper, we augment a hierarchical LSTM language model that generates sequences of word tokens character by character with a caching mechanism that learns to reuse previously generated words. To validate our model we construct a new open-vocabulary language modeling corpus (the Multilingual Wikipedia Corpus; MWC) from comparable Wikipedia articles in 7 typologically diverse languages and demonstrate the effectiveness of our model across this range of languages.

1 Introduction

Language modeling is an important problem in natural language processing with many practical applications (translation, speech recognition, spelling autocorrection, etc.). Recent advances in neural networks provide strong representational power to language models with distributed representations and unbounded dependencies based on recurrent networks (RNNs). However, most language models operate by generating words by sampling from a closed vocabulary which is composed of the most frequent words in a corpus. Rare tokens are typically replaced by a special token, called the unknown word token, ⟨UNK⟩. Although fixed-vocabulary language models have some important practical applications and are appealing models for study, they fail to capture two empirical facts about the distribution of words in natural languages. First, vocabularies keep growing as the number of documents in a corpus grows: new words are constantly being created (Heaps, 1978). Second, rare and newly created words often occur in "bursts", i.e., once a new or rare word has been used once in a document, it is often repeated (Church and Gale, 1995; Church, 2000).

The open-vocabulary problem can be solved by dispensing with word-level models in favor of models that predict sentences as sequences of characters (Sutskever et al., 2011; Chung et al., 2017). Character-based models are quite successful at learning what (new) word forms look like (e.g., they learn a language's orthographic conventions that tell us that *sustinated* is a plausible English word and *bzoxqir* is not) and, when based on models that learn long-range dependencies such as RNNs, they can also be good models of how words fit together to form sentences.

However, existing character-sequence models have no explicit mechanism for modeling the fact that once a rare word is used, it is likely to be used again. In this paper, we propose an extension to character-level language models that enables them to reuse previously generated tokens (§2). Our starting point is a hierarchical LSTM that has been previously used for modeling sentences (word by word) in a conversation (Sordoni et al., 2015), except here we model words (character by character) in a sentence. To this model, we add a caching mechanism similar to recent proposals for caching that have been advocated for closed-vocabulary models (Merity et al., 2017; Grave et al., 2017). As word tokens are generated, they are placed in an LRU cache, and, at each time step the model decides whether to copy a previously generated word from the cache or to generate it from scratch, character by character. The decision of whether

to use the cache or not is a latent variable that is marginalised during learning and inference. In summary, our model has three properties: it creates new words, it accounts for their burstiness using a cache, and, being based on LSTM s over word representations, it can model long range dependencies.

To evaluate our model, we perform ablation experiments with variants of our model without the cache or hierarchical structure. In addition to standard English data sets (PTB and WikiText-2), we introduce a new multilingual data set: the Multilingual Wikipedia Corpus (MWC), which is constructed from comparable articles from Wikipedia in 7 typologically diverse languages (§3) and show the effectiveness of our model in all languages (§4). By looking at the posterior probabilities of the generation mechanism (language model vs. cache) on held-out data, we find that the cache is used to generate "bursty" word types such as proper names, while numbers and generic content words are generated preferentially from the language model (§5).

2 Model

In this section, we describe our hierarchical character language model with a word cache. As is typical for RNN language models, our model uses the chain rule to decompose the problem into incremental predictions of the next word conditioned on the history:

$$p(\boldsymbol{w}) = \prod_{t=1}^{|\boldsymbol{w}|} p(w_t \mid \boldsymbol{w}_{<t}).$$

We make two modifications to the traditional RNN language model, which we describe in turn. First, we begin with a cache-less model we call the hierarchical character language model (HCLM; §2.1) which generates words as a sequence of characters and constructs a "word embedding" by encoding a character sequence with an LSTM (Ling et al., 2015). However, like conventional closed-vocabulary, word-based models, it is based on an LSTM that conditions on words represented by fixed-length vectors.[1]

The HCLM has no mechanism to reuse words that it has previously generated, so new forms will only be repeated with very low probability. However, since the HCLM is not merely generating sentences as a sequence of characters, but also segmenting them into words, we may add a word-based cache to which we add words keyed by the hidden state being used to generate them (§2.2). This cache mechanism is similar to the model proposed by Merity et al. (2017).

Notation. Our model assigns probabilities to sequences of words $\boldsymbol{w} = w_1, \ldots, w_{|\boldsymbol{w}|}$, where $|\boldsymbol{w}|$ is the length, and where each word w_i is represented by a sequence of characters $\boldsymbol{c}_i = c_{i,1}, \ldots, c_{i,|\boldsymbol{c}_i|}$ of length $|\boldsymbol{c}_i|$.

2.1 Hierarchical Character-level Language Model (HCLM)

This hierarchical model satisfies our linguistic intuition that written language has (at least) two different units, characters and words.

The HCLM consists of four components, three LSTMs (Hochreiter and Schmidhuber, 1997): a character encoder, a word-level context encoder, and a character decoder (denoted LSTM_{enc}, LSTM_{ctx}, and LSTM_{dec}, respectively), and a softmax output layer over the character vocabulary. Fig. 1 illustrates an unrolled HCLM.

Suppose the model reads word w_{t-1} and predicts the next word w_t. First, the model reads the character sequence representing the word $w_{t-1} = c_{t-1,1}, \ldots, c_{t-1,|\boldsymbol{c}_{t-1}|}$ where $|\boldsymbol{c}_{t-1}|$ is the length of the word generated at time $t-1$ in characters. Each character is represented as a vector $\mathbf{v}_{c_{t-1,1}}, \ldots, \mathbf{v}_{c_{t-1,|\boldsymbol{c}_{t-1}|}}$ and fed into the encoder LSTM_{enc} . The final hidden state of the encoder LSTM_{enc} is used as the vector representation of the previously generated word w_{t-1},

$$\mathbf{h}_t^{enc} = \text{LSTM}_{enc}(\mathbf{v}_{c_{t-1,1}}, \ldots, \mathbf{v}_{c_{t-1,|\boldsymbol{c}_t|}}).$$

Then all the vector representations of words $(\mathbf{v}_{w_1}, \ldots, \mathbf{v}_{w_{|\boldsymbol{w}|}})$ are processed with a context LSTM_{ctx} . Each of the hidden states of the context LSTM_{ctx} are considered representations of the history of the word sequence.

$$\mathbf{h}_t^{ctx} = \text{LSTM}_{ctx}(\mathbf{h}_1^{enc}, \ldots, \mathbf{h}_t^{enc})$$

Finally, the initial state of the decoder LSTM is set to be $\mathbf{h}_t^{ctx}$ and the decoder LSTM reads a vector representation of the start symbol $\mathbf{v}_{\langle \text{S} \rangle}$ and generates the next word w_{t+1} character by character. To predict the j-th character in w_t, the decoder

[1] The HCLM is an adaptation of the hierarchical recurrent encoder-decoder of Sordoni et al. (2015) which was used to model dialog as a sequence of actions sentences which are themselves sequences of words. The original model was proposed to compose words into query sequences but we use it to compose characters into word sequences.

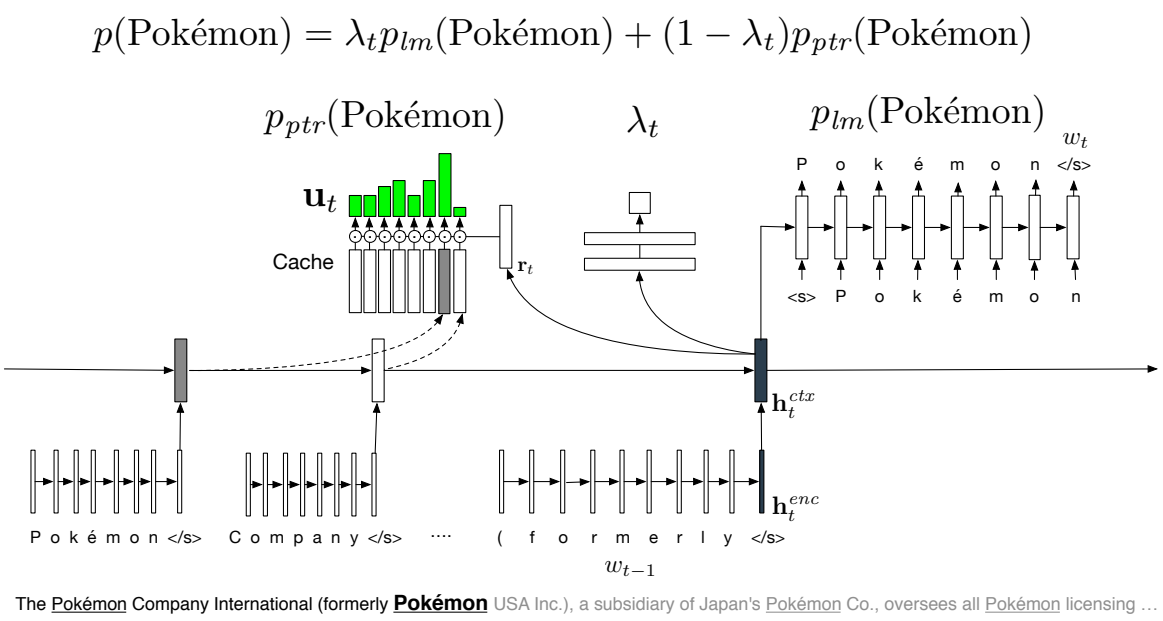

Figure 1: Description of Hierarchical Character Language Model with Cache.

LSTM reads vector representations of the previous characters in the word, conditioned on the context vector $\mathbf{h}_t^{ctx}$ and a start symbol.

$$\mathbf{h}_{t,j}^{dec} = \text{LSTM}_{dec}(\mathbf{v}_{c_{t,1}}, \ldots, \mathbf{v}_{c_{t,j-1}}, \mathbf{h}_t^{ctx}, \mathbf{v}_{\langle s \rangle}).$$

The character generation probability is defined by a softmax layer for the corresponding hidden representation of the decoder LSTM .

$$p(c_{t,j} \mid \boldsymbol{w}_{<t}, \boldsymbol{c}_{t,<j}) = \text{softmax}(\mathbf{W}_{dec}\mathbf{h}_{t,j}^{dec} + \mathbf{b}_{dec})$$

Thus, a word generation probability from HCLM is defined as follows.

$$p_{lm}(w_t \mid \boldsymbol{w}_{<t}) = \prod_{j=1}^{|\boldsymbol{c}_t|} p(c_{t,j} \mid \boldsymbol{w}_{<t}, \boldsymbol{c}_{t,<j})$$

2.2 Continuous cache component

The cache component is an external memory structure which store K elements of recent history. Similarly to the memory structure used in Grave et al. (2017), a word is added to a key-value memory after each generation of w_t. The key at position $i \in [1, K]$ is $\mathbf{k}_i$ and its value m_i. The memory slot is chosen as follows: if the w_t exists already in the memory, its key is updated (discussed below). Otherwise, if the memory is not full, an empty slot is chosen or the least recently used slot is overwritten. When writing a new word to memory, the key is the RNN representation that was used to generate the word ($\mathbf{h}_t$) and the value is the word itself (w_t). In the case when the word already exists in the cache at some position i, the $\mathbf{k}_i$ is updated to be the arithmetic average of $\mathbf{h}_t$ and the existing $\mathbf{k}_i$.

To define the copy probability from the cache at time t, a distribution over copy sites is defined using the attention mechanism of Bahdanau et al. (2015). To do so, we construct a query vector ($\mathbf{r}_t$) from the RNN's current hidden state $\mathbf{h}_t$,

$$\mathbf{r}_t = \tanh(\mathbf{W}_q\mathbf{h}_t + \mathbf{b}_q),$$

then, for each element i of the cache, a 'copy score,' $u_{i,t}$ is computed,

$$u_{i,t} = \mathbf{v}^T \tanh(\mathbf{W}_u\mathbf{k}_i + \mathbf{r}_t).$$

Finally, the probability of generating a word via the copying mechanism is:

$$p_{mem}(i \mid \mathbf{h}_t) = \text{softmax}_i(\mathbf{u}_t)$$
$$p_{ptr}(w_t \mid \mathbf{h}_t) = p_{mem}(i \mid \mathbf{h}_t)[m_i = w_t],$$

where $[m_i = w_t]$ is 1 if the ith value in memory is w_t and 0 otherwise. Since p_{mem} defines a distribution of slots in the cache, p_{ptr} translates it into word space.

2.3 Character-level Neural Cache Language Model

The word probability $p(w_t \mid \boldsymbol{w}_{<t})$ is defined as a mixture of the following two probabilities. The first

one is a language model probability, $p_{lm}(w_t \mid \boldsymbol{w}_{<t})$ and the other is pointer probability , $p_{ptr}(w_t \mid \boldsymbol{w}_{<t})$. The final probability $p(w_t \mid \boldsymbol{w}_{<t})$ is

$$\lambda_t p_{lm}(w_t \mid \boldsymbol{w}_{<t}) + (1 - \lambda_t) p_{ptr}(w_t \mid \boldsymbol{w}_{<t}),$$

where λ_t is computed by a multi-layer perceptron with two non-linear transformations using $\mathbf{h}_t$ as its input, followed by a transformation by the logistic sigmoid function:

$$\gamma_t = \text{MLP}(\mathbf{h}_t), \qquad \lambda_t = \frac{1}{1 - e^{-\gamma_t}}.$$

We remark that Grave et al. (2017) use a clever trick to estimate the probability, λ_t of drawing from the LM by augmenting their (closed) vocabulary with a special symbol indicating that a copy should be used. This enables word types that are highly predictive in context to compete with the probability of a copy event. However, since we are working with an open vocabulary, this strategy is unavailable in our model, so we use the MLP formulation.

2.4 Training objective

The model parameters as well as the character projection parameters are jointly trained by maximizing the following log likelihood of the observed characters in the training corpus,

$$\mathcal{L} = -\sum \log p(w_t \mid \boldsymbol{w}_{<t}).$$

3 Datasets

We evaluate our model on a range of datasets, employing preexisting benchmarks for comparison to previous published results, and a new multilingual corpus which specifically tests our model's performance across a range of typological settings.

3.1 Penn Tree Bank (PTB)

We evaluate our model on the Penn Tree Bank. For fair comparison with previous works, we followed the standard preprocessing method used by Mikolov et al. (2010). In the standard preprocessing, tokenization is applied, words are lowercased, and punctuation is removed. Also, less frequent words are replaced by unknown an token (UNK),[2] constraining the word vocabulary size to be 10k. Because of this preprocessing, we do not expect this dataset to benefit from the modeling innovations we have introduced in the paper. Fig.1 summarizes the corpus statistics.

[2]When the unknown token is used in character-level model, it is treated as if it were a normal word (i.e. UNK is the sequence U, N, and K). This is somewhat surprising modeling choice, but it has become conventional (Chung et al., 2017).

	Train	Dev	Test
Character types	50	50	48
Word types	10000	6022	6049
OOV rate	-	**0.00%**	**0.00%**
Word tokens	0.9M	0.1M	0.1M
Characters	5.1M	0.4M	0.4M

Table 1: PTB Corpus Statistics.

3.2 WikiText-2

Merity et al. (2017) proposed the WikiText-2 Corpus as a new benchmark dataset.[3] They pointed out that the preprocessed PTB is unrealistic for real language use in terms of word distribution. Since the vocabulary size is fixed to 10k, the word frequency does not exhibit a long tail. The wikiText-2 corpus is constructed from 720 articles. They provided two versions. The version for word level language modeling was preprocessed by discarding infrequent words. But, for character-level models, they provided raw documents without any removal of word or character types or lowercasing, but with tokenization. We make one change to this corpus: since Wikipedia articles make extensive use of characters from other languages; we replaced character types that occur fewer than 25 times were replaced with a dummy character (this plays the role of the ⟨UNK⟩ token in the character vocabulary). Tab. 2 summarizes the corpus statistics.

	Train	Dev	Test
Character types	255	128	138
Word types	76137	19813	21109
OOV rate	-	4.79%	5.87%
Word tokens	2.1M	0.2M	0.2M
Characters	10.9M	1.1M	1.3M

Table 2: WikiText-2 Corpus Statistics.

3.3 Multilingual Wikipedia Corpus (MWC)

Languages differ in what word formation processes they have. For character-level modeling it is therefore interesting to compare a model's performance

[3]http://metamind.io/research/the-wikitext-long-term-dependency-language-modeling-dataset/

across languages. Since there is at present no standard multilingual language modeling dataset, we created a new dataset, the Multilingual Wikipedia Corpus (MWC), a corpus of the same Wikipedia articles in 7 languages which manifest a range of morphological typologies. The MWC contains English (EN), French (FR), Spanish (ES), German (DE), Russian (RU), Czech (CS), and Finnish (FI).

To attempt to control for topic divergences across languages, every language's data consists of the same articles. Although these are only comparable (rather than true translations), this ensures that the corpus has a stable topic profile across languages.[4]

Construction & Preprocessing We constructed the MWC similarly to the WikiText-2 corpus. Articles were selected from Wikipedia in the 7 target languages. To keep the topic distribution to be approximately the same across the corpora, we extracted articles about entities which explained in all the languages. We extracted articles which exist in all languages and each consist of more than 1,000 words, for a total of 797 articles. These cross-lingual articles are, of course, not usually translations, but they tend to be comparable. This filtering ensures that the topic profile in each language is similar. Each language corpus is approximately the same size as the WikiText-2 corpus.

Wikipedia markup was removed with WikiExtractor,[5] to obtain plain text. We used the same thresholds to remove rare characters in the WikiText-2 corpus. No tokenization or other normalization (e.g., lowercasing) was done.

Statistics After the preprocessing described above, we randomly sampled 360 articles. The articles are split into 300, 30, 30 sets and the first 300 articles are used for training and the rest are used for dev and test respectively. Table 3 summarizes the corpus statistics.

Additionally, we show in Fig. 2 the distribution of frequencies of OOV word types (relative to the training set) in the dev+test portions of the corpus, which shows a power-law distribution, which is expected for the burstiness of rare words found in prior work. Curves look similar for all languages (see Appendix A).

[4]The Multilingual Wikipedia Corpus (MWC) is available for download from http://k-kawakami.com/research/mwc

[5]https://github.com/attardi/wikiextractor

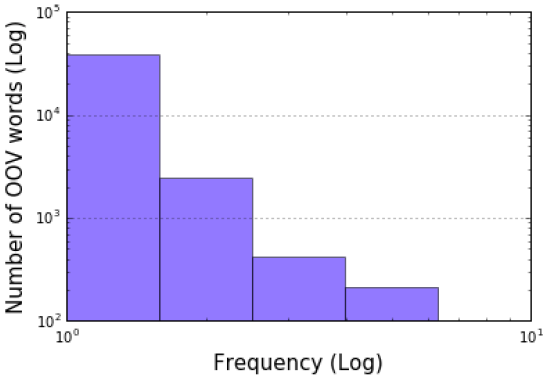

Figure 2: Histogram of OOV word frequencies in the dev+test part of the MWC Corpus (EN).

4 Experiments

We now turn to a series of experiments to show the value of our hierarchical character-level cache language model. For each dataset we trained the model with LSTM units. To compare our results with a strong baseline, we also train a model without the cache.

Model Configuration For HCLM and HCLM with cache models, We used 600 dimensions for the character embeddings and the LSTMs have 600 hidden units for all the experiments. This keeps the model complexity to be approximately the same as previous works which used an LSTM with 1000 dimension. Our baseline LSTM have 1000 dimensions for embeddings and reccurence weights.

For the cache model, we used cache size 100 in every experiment. All the parameters including character projection parameters are randomly sampled from uniform distribution from -0.08 to 0.08. The initial hidden and memory state of LSTM_{enc} and LSTM_{ctx} are initialized with zero. Mini-batches of size 25 are used for PTB experiments and 10 for WikiText-2, due to memory limitations. The sequences were truncated with 35 words. Then the words are decomposed to characters and fed into the model. A Dropout rate of 0.5 was used for all but the recurrent connections.

Learning The models were trained with the Adam update rule (Kingma and Ba, 2015) with a learning rate of 0.002. The maximum norm of the gradients was clipped at 10.

Evaluation We evaluated our models with bits-per-character (bpc) a standard evaluation metric

	Char. Types			Word Types			OOV rate		Tokens			Characters		
	Train	Valid	Test	Train	Valid	Test	Valid	Test	Train	Valid	Test	Train	Valid	Test
EN	307	160	157	193808	38826	35093	6.60%	5.46%	2.5M	0.2M	0.2M	15.6M	1.5M	1.3M
FR	272	141	155	166354	34991	38323	6.70%	6.96%	2.0M	0.2M	0.2M	12.4M	1.3M	1.6M
DE	298	162	183	238703	40848	41962	7.07%	7.01%	1.9M	0.2M	0.2M	13.6M	1.2M	1.3M
ES	307	164	176	160574	31358	34999	6.61%	7.35%	1.8M	0.2M	0.2M	11.0M	1.0M	1.3M
CS	238	128	144	167886	23959	29638	5.06%	6.44%	0.9M	0.1M	0.1M	6.1M	0.4M	0.5M
FI	246	123	135	190595	32899	31109	8.33%	7.39%	0.7M	0.1M	0.1M	6.4M	0.7M	0.6M
RU	273	184	196	236834	46663	44772	7.76%	7.20%	1.3M	0.1M	0.1M	9.3M	1.0M	0.9M

Table 3: Summary of MWC Corpus.

for character-level language models. Following the definition in Graves (2013), bits-per-character is the average value of $-\log_2 p(w_t \mid \boldsymbol{w}_{<t})$ over the whole test set,

$$bpc = -\frac{1}{|\boldsymbol{c}|} \log_2 p(\boldsymbol{w}),$$

where $|\boldsymbol{c}|$ is the length of the corpus in characters.

4.1 Results

PTB Tab. 4 summarizes results on the PTB dataset.[6] Our baseline HCLM model achieved 1.276 bpc which is better performance than the LSTM with Zoneout regularization (Krueger et al., 2017). And HCLM with cache outperformed the baseline model with 1.247 bpc and achieved competitive results with state-of-the-art models with regularization on recurrence weights, which was not used in our experiments.

Expressed in terms of per-word perplexity (i.e., rather than normalizing by the length of the corpus in characters, we normalize by words and exponentiate), the test perplexity on HCLM with cache is 94.79. The performance of the unregularized 2-layer LSTM with 1000 hidden units on word-level PTB dataset is 114.5 and the same model with dropout achieved 87.0. Considering the fact that our character-level models are dealing with an open vocabulary without unknown tokens, the results are promising.

WikiText-2 Tab. 5 summarizes results on the WikiText-2 dataset. Our baseline, LSTM achieved 1.803 bpc and HCLM model achieved 1.670 bpc. The HCLM with cache outperformed the baseline models and achieved 1.500 bpc. The word level perplexity is 227.30, which is quite high compared to the reported word level baseline result 100.9 with LSTM with ZoneOut and Variational Dropout regularization (Merity et al., 2017). However, the character-level model is dealing with 76,136 types in training set and 5.87% OOV rate where the word level models only use 33,278 types without OOV in test set. The improvement rate over the HCLM baseline is 10.2% which is much higher than the improvement rate obtained in the PTB experiment.

[6]Models designated with a * have more layers and more parameters.

Method	Dev	Test
CW-RNN (Koutnik et al., 2014)	-	1.46
HF-MRNN (Mikolov et al., 2012)	-	1.41
MI-RNN (Wu et al., 2016)	-	1.39
ME n-gram (Mikolov et al., 2012)	-	1.37
RBN (Cooijmans et al., 2017)	1.281	1.32
Recurrent Dropout (Semeniuta et al., 2016)	1.338	1.301
Zoneout (Krueger et al., 2017)	1.362	1.297
HM-LSTM (Chung et al., 2017)	-	1.27
HyperNetwork (Ha et al., 2017)	1.296	1.265
LayerNorm HyperNetwork (Ha et al., 2017)	1.281	1.250
2-LayerNorm HyperLSTM (Ha et al., 2017)*	-	1.219
2-Layer with New Cell (Zoph and Le, 2016)*	-	1.214
LSTM (Our Implementation)	1.369	1.331
HCLM	1.308	1.276
HCLM with Cache	**1.266**	**1.247**

Table 4: Results on PTB Corpus (bits-per-character). HCLM augmented with a cache obtains the best results among models which have approximately the same numbers of parameter as single layer LSTM with 1,000 hidden units.

Method	Dev	Test
LSTM	1.758	1.803
HCLM	1.625	1.670
HCLM with Cache	**1.480**	**1.500**

Table 5: Results on WikiText-2 Corpus .

Multilingual Wikipedia Corpus (MWC) Tab. 6 summarizes results on the MWC dataset. Similarly to WikiText-2 experiments, LSTM

is strong baseline. We observe that the cache mechanism improve performance in every languages. In English, HCLM with cache achieved 1.538 bpc where the baseline is 1.622 bpc. It is 5.2% improvement. For other languages, the improvement rates were 2.7%, 3.2%, 3.7%, 2.5%, 4.7%, 2.7% in FR, DE, ES, CS, FI, RU respectively. The best improvement rate was obtained in Finnish.

5 Analysis

In this section, we analyse the behavior of proposed model qualitatively. To analyse the model, we compute the following posterior probability which tell whether the model used the cache given a word and its preceding context. Let z_t be a random variable that says whether to use the cache or the LM to generate the word at time t. We would like to know, given the text $\boldsymbol{w}$, whether the cache was used at time t. This can be computed as follows:

$$p(z_t \mid \boldsymbol{w}) = \frac{p(z_t, w_t \mid \mathbf{h}_t, \text{cache}_t)}{p(w_t \mid \mathbf{h}_t, \text{cache}_t)} = \frac{(1-\lambda_t) p_{ptr}(w_t \mid \mathbf{h}_t, \text{cache}_t)}{p(w_t \mid \mathbf{h}_t, \text{cache}_t)},$$

where cache_t is the state of the cache at time t. We report the average posterior probability of cache generation excluding the first occurrence of w, $\overline{p(z \mid w)}$.

Tab. 7 shows the words in the WikiText-2 test set that occur more than 1 time that are most/least likely to be generated from cache and character language model (words that occur only one time cannot be cache-generated). We see that the model uses the cache for proper nouns: *Lesnar*, *Gore*, etc., as well as very frequent words which always stored somewhere in the cache such as single-token punctuation, *the*, and *of*. In contrast, the model uses the language model to generate numbers (which tend not to be repeated): *300*, *770* and basic content words: *sounds*, *however*, *unable*, etc. This pattern is similar to the pattern found in empirical distribution of frequencies of rare words observed in prior wors (Church and Gale, 1995; Church, 2000), which suggests our model is learning to use the cache to account for bursts of rare words.

To look more closely at rare words, we also investigate how the model handles words that occurred between 2 and 100 times in the test set, but fewer than 5 times in the training set. Fig. 3 is a scatter plot of $\overline{p(z \mid w)}$ vs the empirical frequency in the test set. As expected, more frequently repeated words types are increasingly likely to be drawn from the cache, but less frequent words show a range of cache generation probabilities.

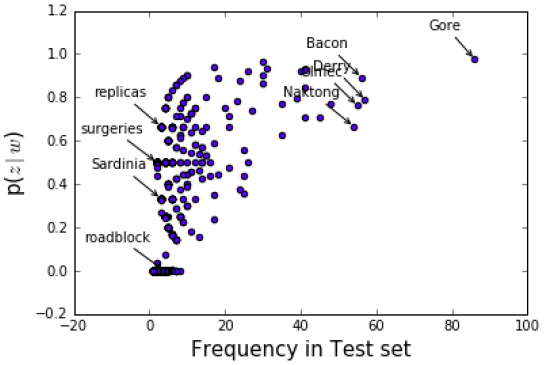

Figure 3: Average $p(z \mid w)$ of OOV words in test set vs. term frequency in the test set for words not obsered in the training set. The model prefers to copy frequently reused words from cache component, which tend to names (upper right) while character level generation is used for infrequent open class words (bottom left).

Tab. 8 shows word types with the highest and lowest average $p(z \mid w)$ that occur fewer than 5 times in the training corpus. The pattern here is similar to the unfiltered list: proper nouns are extremely likely to have been cache-generated, whereas numbers and generic (albeit infrequent) content words are less likely to have been.

6 Discussion

Our results show that the HCLM outperforms a basic LSTM. With the addition of the caching mechanism, the HCLM becomes consistently more powerful than both the baseline HCLM and the LSTM. This is true even on the PTB, which has no rare or OOV words in its test set (because of preprocessing), by caching repetitive common words such as *the*. In true open-vocabulary settings (i.e., WikiText-2 and MWC), the improvements are much more pronounced, as expected.

Computational complexity. In comparison with word-level models, our model has to read and generate each word character by character, and it also requires a softmax over the entire memory at every time step. However, the computation is still linear in terms of the length of the sequence, and the softmax over the memory cells and character

	EN		FR		DE		ES		CS		FI		RU	
	dev	test	dev	test	dev	test	dev	test	dev	test	dev	test	dev	test
LSTM	1.793	1.736	1.669	1.621	1.780	1.754	1.733	1.667	2.191	2.155	1.943	1.913	1.942	1.932
HCLM	1.683	1.622	1.553	1.508	1.666	1.641	1.617	1.555	2.070	2.035	1.832	1.796	1.832	1.810
HCLM with Cache	**1.591**	**1.538**	**1.499**	**1.467**	**1.605**	**1.588**	**1.548**	**1.498**	**2.010**	**1.984**	**1.754**	**1.711**	**1.777**	**1.761**

Table 6: Results on MWC Corpus (bits-per-character).

Word	$\overline{p(z \mid \boldsymbol{w})} \downarrow$	Word	$\overline{p(z \mid w)} \uparrow$
.	0.997	300	0.000
Lesnar	0.991	act	0.001
the	0.988	however	0.002
NY	0.985	770	0.003
Gore	0.977	put	0.003
Bintulu	0.976	sounds	0.004
Nerva	0.976	instead	0.005
,	0.974	440	0.005
UB	0.972	similar	0.006
Nero	0.967	27	0.009
Osbert	0.967	help	0.009
Kershaw	0.962	few	0.010
Manila	0.962	110	0.010
Boulter	0.958	Jersey	0.011
Stevens	0.956	even	0.011
Rifenburg	0.952	y	0.012
Arjona	0.952	though	0.012
of	0.945	becoming	0.013
31B	0.941	An	0.013
Olympics	0.941	unable	0.014

Table 7: Word types with the highest/lowest average posterior probability of having been copied from the cache while generating the test set. The probability tells whether the model used the cache given a word and its context. **Left:** Cache is used for frequent words (*the*, *of*) and proper nouns (*Lesnar*, *Gore*). **Right:** Character level generation is used for basic words and numbers.

Word	$\overline{p(z \mid \boldsymbol{w})} \downarrow$	Word	$\overline{p(z \mid w)} \uparrow$
Gore	0.977	770	0.003
Nero	0.967	246	0.037
Osbert	0.967	Lo	0.074
Kershaw	0.962	Pitcher	0.142
31B	0.941	Poets	0.143
Kirby	0.935	popes	0.143
CR	0.926	Yap	0.143
SM	0.924	Piso	0.143
impedance	0.923	consul	0.143
Blockbuster	0.900	heavyweight	0.143
Superfamily	0.900	cheeks	0.154
Amos	0.900	loser	0.164
Steiner	0.897	amphibian	0.167
Bacon	0.893	squads	0.167
filters	0.889	los	0.167
Lim	0.889	Keenan	0.167
Selfridge	0.875	sculptors	0.167
filter	0.875	Gen.	0.167
Lockport	0.867	Kipling	0.167
Germaniawerft	0.857	Tabasco	0.167

Table 8: Same as Table 7, except filtering for word types that occur fewer than 5 times in the training set. The cache component is used as expected even on rare words: proper nouns are extremely likely to have been cache-generated, whereas numbers and generic content words are less likely to have been; this indicates both the effectiveness of the prior at determining whether to use the cache and the burstiness of proper nouns.

vocabulary are much smaller than word-level vocabulary. On the other hand, since the recurrent states are updated once per character (rather than per word) in our model, the distribution of operations is quite different. Depending on the hardware support for these operations (repeated updates of recurrent states vs. softmaxes), our model may be faster or slower. However, our model will have fewer parameters than a word-based model since most of the parameters in such models live in the word projection layers, and we use LSTMs in place of these.

Non-English languages. For non-English languages, the pattern is largely similar for non-English languages. This is not surprising since morphological processes may generate forms that are related to existing forms, but these still have slight variations. Thus, they must be generated by the language model component (rather than from the cache). Still, the cache demonstrates consistent value in these languages.

Finally, our analysis of the cache on English does show that it is being used to model word reuse, particularly of proper names, but also of frequent words. While empirical analysis of rare word distributions predicts that names would be reused, the fact that cache is used to model frequent words suggests that effective models of language should have a means to generate common words as units. Finally, our model disfavors copying numbers from the cache, even when they are available. This suggests that it has learnt that numbers are not generally repeated (in contrast to names).

7 Related Work

Caching language models were proposed to account for burstiness by Kuhn and De Mori (1990), and recently, this idea has been incorporated to augment neural language models with a caching mechanism (Merity et al., 2017; Grave et al., 2017).

Open vocabulary neural language models have been widely explored (Sutskever et al., 2011; Mikolov et al., 2012; Graves, 2013, *inter alia*). Attempts to make them more aware of word-level dynamics, using models similar to our hierarchical formulation, have also been proposed (Chung et al., 2017).

The only models that are open vocabulary language modeling together with a caching mechanism are the nonparametric Bayesian language models based on hierarchical Pitman–Yor processes which generate a lexicon of word types using a character model, and then generate a text using these (Teh, 2006; Goldwater et al., 2009; Chahuneau et al., 2013). These, however, do not use distributed representations on RNNs to capture long-range dependencies.

8 Conclusion

In this paper, we proposed a character-level language model with an adaptive cache which selectively assign word probability from past history or character-level decoding. And we empirically show that our model efficiently model the word sequences and achieved better perplexity in every standard dataset. To further validate the performance of our model on different languages, we collected multilingual wikipedia corpus for 7 typologically diverse languages. We also show that our model performs better than character-level models by modeling burstiness of words in local context.

The model proposed in this paper assumes the observation of word segmentation. Thus, the model is not directly applicable to languages, such as Chinese and Japanese, where word segments are not explicitly observable. We will investigate a model which can marginalise word segmentation as latent variables in the future work.

Acknowledgements

We thank the three anonymous reviewers for their valuable feedback. The third author acknowledges the support of the EPSRC and nvidia Corporation.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. ICLR*.

Victor Chahuneau, Noah A. Smith, and Chris Dyer. 2013. Knowledge-rich morphological priors for bayesian language models. In *Proc. NAACL*.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2017. Hierarchical multiscale recurrent neural networks. In *Proc. ICLR*.

Kenneth W Church. 2000. Empirical estimates of adaptation: the chance of two Noriegas is closer to $p/2$ than p^2. In *Proc. COLING*.

Kenneth W Church and William A Gale. 1995. Poisson mixtures. *Natural Language Engineering* 1(2):163–190.

Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. 2017. Recurrent batch normalization. In *Proc. ICLR*.

Sharon Goldwater, Thomas L Griffiths, and Mark Johnson. 2009. A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition* 112(1):21–54.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. In *Proc. ICLR*.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850* .

David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. In *Proc. ICLR*.

Harold Stanley Heaps. 1978. *Information retrieval: Computational and theoretical aspects*. Academic Press, Inc.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. ICLR*.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. 2014. A clockwork RNN. In *Proc. ICML*.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. 2017. Zoneout: Regularizing rnns by randomly preserving hidden activations. In *Proc. ICLR*.

Roland Kuhn and Renato De Mori. 1990. A cache-based natural language model for speech recognition. *IEEE transactions on pattern analysis and machine intelligence* 12(6):570–583.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. EMNLP*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. ICLR*.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. Interspeech*.

Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and Jan Cernocky. 2012. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)* .

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent dropout without memory loss. In *Proc. COLING*.

Alessandro Sordoni, Yoshua Bengio, Hossein Vahabi, Christina Lioma, Jakob Grue Simonsen, and Jian-Yun Nie. 2015. A hierarchical recurrent encoder-decoder for generative context-aware query suggestion. In *Proc. CIKM*.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proc. ICML*.

Yee Whye Teh. 2006. A hierarchical Bayesian language model based on Pitman-Yor processes. In *Proc. ACL*.

Yuhuai Wu, Saizheng Zhang, Ying Zhang, Yoshua Bengio, and Ruslan R Salakhutdinov. 2016. On multiplicative integration with recurrent neural networks. In *Proc. NIPS*.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* .

A Corpus Statistics

Fig. 4 show distribution of frequencies of OOV word types in 6 languages.

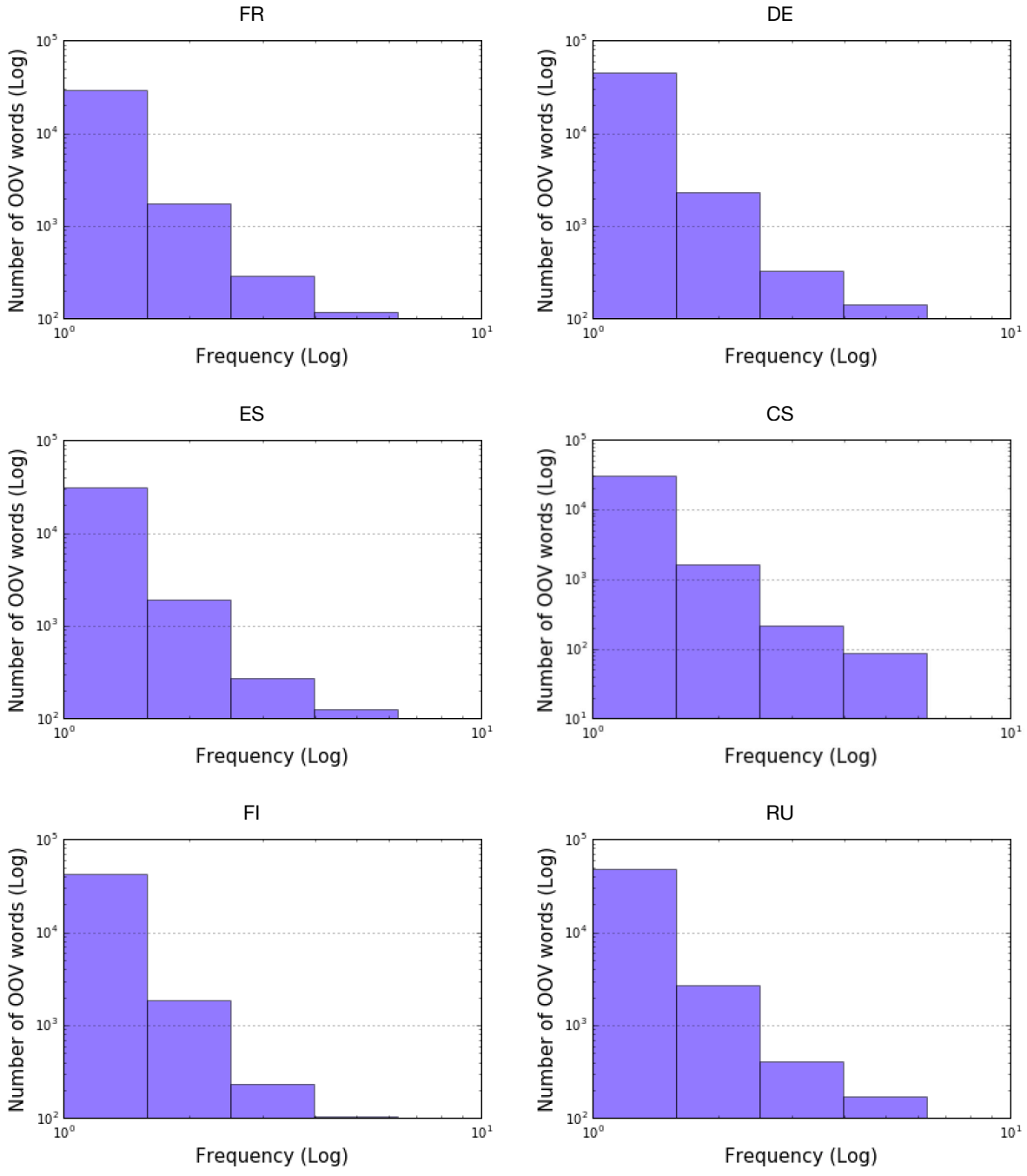

Figure 4: Histogram of OOV word frequencies in MWC Corpus in different languages.