The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)

Spell Once, Summon Anywhere: A Two-Level Open-Vocabulary Language Model

Sebastian J. Mielke, Jason Eisner
Department of Computer Science, Johns Hopkins University, Baltimore, MD, USA
sjmielke@jhu.edu, jason@cs.jhu.edu

Abstract

We show how the spellings of known words can help us deal with unknown words in open-vocabulary NLP tasks. The method we propose can be used to extend any closed-vocabulary generative model, but in this paper we specifically consider the case of neural language modeling. Our Bayesian generative story combines a standard RNN language model (generating the word *tokens* in each sentence) with an RNN-based spelling model (generating the letters in each word *type*). These two RNNs respectively capture sentence structure and word structure, and are kept separate as in linguistics. By invoking the second RNN to generate spellings for novel words in context, we obtain an open-vocabulary language model. For known words, embeddings are naturally inferred by combining evidence from type spelling and token context. Comparing to baselines (including a novel strong baseline), we beat previous work and establish state-of-the-art results on multiple datasets.

1 Introduction

In this paper, we propose a neural language model that incorporates a generative model of word spelling. That is, we aim to explain the training corpus as resulting from a process that first generated a lexicon of word types—the language's vocabulary—and then generated the corpus of tokens by "summoning" those types.

Each entry in the lexicon specifies both a syntactic/semantic embedding vector and an orthographic spelling. Our model captures the correlation between these, so it can extrapolate to predict the spellings of *unknown* words in any syntactic/semantic context. In this sample from our trained English model, the words in `this font` were unobserved in training data, yet have contextually appropriate spellings:

> Following the death of Edward McCartney in `1060` , the new definition was transferred to the `WDIC` of `Fullett` .

While the fully generative approach is shared by previous Bayesian models of language (e.g., Goldwater, Griffiths, and Johnson (2006)), even those that model characters and words at different levels (Mochihashi, Yamada, and Ueda 2009; Goldwater, Griffiths, and Johnson 2011) have no embeddings and hence no way to relate spelling to usage. They also have an impoverished model of sequential structure (essentially, n-gram models with backoff). We instead employ *recurrent neural networks* to model both the sequence of words in a sentence and the sequence of characters in a word type, where the latter sequence is conditioned on the word's embedding. The resulting model achieves state-of-the-art on multiple datasets. It is well-founded in linguistics and Bayesian modeling, but we can easily use it in the traditional non-Bayesian language modeling setting by performing MAP estimation.

We begin by concisely stating a first, closed-vocabulary, version of our model in §2, before explaining our motivations from various perspectives in §3. Then §4 motivates and describes a simple way to extend the model to the open-vocabulary setting. §5 contains quantitative and qualitative experiments on multiple language modeling datasets, with implementation details provided in supplementary material. Finally, we clarify the relation of this model to previous work in §6 and summarize our contribution in §7.

2 A joint model of lexicon and text

2.1 Lexemes have embeddings and spellings

We will model text that has already been tokenized, i.e., it is presented as a sequence of word tokens $w_1, w_2, \ldots$.

We assume that a language's word types, which we henceforth call *lexemes* to avoid confusion, are discrete elements w of the *vocabulary* $\mathcal{V} = \{\textcircled{1}, \textcircled{2}, \ldots \textcircled{v}\}$. In our model, the observable behavior of the lexeme w is determined by two properties: a latent real-valued *embedding* $e(w) \in \mathbb{R}^d$, which governs where w tends to appear, and w's spelling $\sigma(w) \in \Sigma^*$ (for some alphabet of characters Σ), which governs how it looks orthographically when it does appear.

We will use e and σ to refer to the functions that map each lexeme w to its embedding and spelling. Thus the lexicon is specified by (e, σ). Our model[1] (given fixed v and n) specifies a joint distribution over the lexicon and corpus:

$$p(\theta, e, \sigma, w_1, \ldots, w_n) = p(\theta) \cdot \underbrace{\prod_{w \in \mathcal{V}} \Big[\underbrace{p(e(w))}_{\text{prior on embeddings}} \cdot \underbrace{p_{\text{spell}}(\sigma(w) \mid e(w))}_{\text{spelling model for all types}} \Big]}_{\text{lexicon generation}} \cdot \underbrace{\prod_{i=1}^{n} p_{\text{LM}}(w_i \mid \vec{w}_{<i}, e)}_{\text{lexeme-level recurrent language model for all tokens}} \qquad (1)$$

[1]Before extension to the open-vocabulary case, found in Eq. (3).

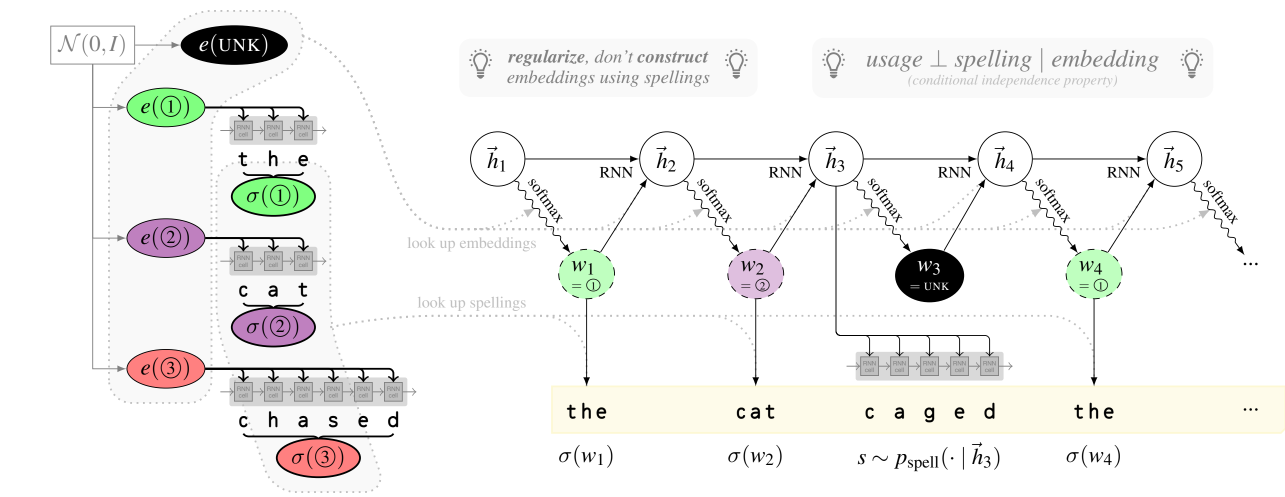

Figure 1: A lexeme's embedding is optimized to be predictive of the lexeme's spelling. That spelling is predicted only once (left); every corpus token of that lexeme type (right) simply "summons," or copies, the type's spelling. For the novel word w_3 (§4), `caged` is preferred over something unpronounceable like `xsmfk`, but also over the ungrammatical `furry`, because the hidden state $\vec{h}_3$ prefers a verb and the spelling model can generalize from the verb `chased` ending in `-ed` to `caged`.

where p_{LM} (§2.2) and p_{spell} (§2.3) are RNN sequence models that are parameterized by θ (the dependence is omitted for space reasons), and $w_1, \ldots, w_n$ is a sequence of word tokens.

Let us unpack this formula. The generative story for the observed training corpus has two steps:

Generate the structure of the language. First draw RNN parameters θ (from a spherical Gaussian). Then draw an embedding $e(w)$ for each lexeme w (from another spherical Gaussian).[2] Finally, sample a spelling $\sigma(w)$ for each lexeme w from the p_{spell} model, conditioned on $e(w)$ (and θ).

Generate the corpus. In the final term of (1), generate a sequence of lexemes $w_1, \ldots, w_n$ from the p_{LM} model (using θ). Applying σ yields the actually observed training corpus $\sigma(w_1), \ldots, \sigma(w_n)$, a sequence of spelled words.

In the present paper we make the common simplifying assumption that the training corpus has no polysemy, so that two word tokens with the same spelling always correspond to the same lexeme. We thus assign distinct lexeme numbers ①, ②, …, ⓥ to the different spelling types in the corpus (the specific assignment does not matter). Thus, we have observed the spellings σ(①), σ(②), …, σ(ⓥ) of these v assumed lexemes. We have also observed the actual token sequence $w_1, \ldots, w_n$ where each w_i is a lexeme number.

Given these observations, we train θ and e jointly by MAP estimation: in other words, we choose them to (locally) maximize (1).[3] This is straightforward using backpropagation and gradient descent (see Appendix A[4] on training and Appendix B for implementation and hyperparameter details).

[2] We will later find the MAP estimates of θ and $e(w)$, so the Gaussian priors correspond to L_2 regularization of these estimates.

[3] The non-Bayesian view on this is that we maximize the *regularized likelihood* of the model given the observations.

[4] All appendices (supp. material) can be found in the full version of this paper at https://arxiv.org/abs/1804.08205.

2.2 Modeling word sequences with p_{LM}

The final term of (1) is simply a neural language model that uses the embeddings e. It should be stressed that *any* such neural language model can be used here. We will use the commonly used AWD-LSTM built by Merity, Keskar, and Socher (2017) with their best reported hyperparameters.

2.3 Modeling letter sequences with p_{spell}

Our model also has to predict the spelling of every lexeme. We model $p_{\text{spell}}(\sigma(w) \mid e(w))$ with a vanilla LSTM language model (Sundermeyer, Schlüter, and Ney 2012), this time over characters, using special characters BOW (beginning of word) and EOW (end of word) to begin and end the sequence.

Our intuition is that in most languages, the spelling of a word tends to weakly reflect categorical properties that are hopefully captured in the embedding. For example, proper names may have a certain form, content words may have to contain at least two syllables (McCarthy and Prince 1999), and past-tense verbs may tend to end with a certain suffix. This is why $p_{\text{spell}}(\sigma(w) \mid e(w))$ is conditioned on the lexeme's embedding $e(w)$. We accomplish this by feeding $e(w)$ into $\text{LSTM}_{\text{spell}}$ as additional input at every step, alongside the ordinary inputs (the previous hidden state $\vec{h}_{t-1}$ and an low-dimensional embedding $\vec{c}_{t-1} \in \mathbb{R}^{d'}$ of the previous character):

$$\vec{h}_t = \text{LSTM}_{\text{spell}}\left(\vec{h}_{t-1}, \, [\vec{c}_t \, ; e(w)] \right) \quad (2)$$

As the spelling model tends to overfit to training lexemes (instead of modeling the language's phonotactics, morphology, and other conventions), we project the embeddings $e(w)$ into a *lower-dimensional space* to combat this overfitting. We do so by regularizing the four input weight matrices[5] of $\text{LSTM}_{\text{spell}}$ with the *nuclear norm* (the sum of each matrix's

[5] W_{ii}, W_{if}, W_{ig}, and W_{io} in PyTorch's `LSTMCell` documentation; regularization only applies to the part that is multiplied with $e(w)$.

singular values; details on it in Appendix C), resulting in a low-rank projection. The nuclear norm (times a positive hyperparameter) is added to the objective, namely the negative log of Eq. (1), as part of the definition of the $-\log p(\theta)$ term. This regularizer indeed helps on development data, and it outperformed L_2 regularization in our pilot experiments.

3 Words, characters, types, and tokens

We now take a step back and discuss our modeling principles. Our model structure above aims to incorporate two perspectives that have been neglected in neural language modeling:

3.1 A linguistic perspective

Hockett (1960) regarded *duality of patterning* as a fundamental property of human language: the *form* of a word is logically separate from its *usage*. For example, while `children` may be an unusual spelling for a plural noun in English, it is listed as one in the lexicon, and that grants it all the same privileges as any other plural noun. The syntactic and semantic processes that combine words are blind to its unusual spelling. In our two-level architecture, this "separation of concerns" holds between p_{spell}, which governs word *form*, and p_{LM}, which governs word *usage*. A word's distribution of contexts is conditionally independent of its spelling, given its embedding, because p_{LM} does not consult spellings but only embeddings. Prior work does *not* do this—character-based language models blur the two levels into one.

Half a century earlier, Saussure (1916) discussed the *arbitrariness of the sign*. In our model, p_{spell} has support on all of Σ^*, so any embedding can in principle be paired in the lexicon with any spelling—even if some pairings may be more likely *a priori* than others, perhaps because they are more pronounceable or the result of a morphological transformation. In contrast with most prior work that compositionally derives a word's embedding from its spelling, our model only *prefers* a word's embedding to correlate with its spelling, in order to raise the factor $p_{\text{spell}}(\sigma(w) \mid e(w))$. This preference is merely a regularizing effect that may be overcome by the need to keep the p_{LM} factors large, particularly for a frequent word that appears in many p_{LM} factors and is thus allowed its own idiosyncratic embedding.

In short, our spelling model is *not* supposed to be able to perfectly predict the spelling. However, it can statistically capture phonotactics, regular and semi-regular inflection, etc.

3.2 A modeling perspective

The distinction between word types (i.e., entries in a vocabulary) and tokens (i.e., individual occurrences of these word types in text) is also motivated by a generative (e.g., Bayesian) treatment of language: a lexeme's spelling is *reused* over all its tokens, not generated from scratch for each token.

This means that the term $p_{\text{spell}}(\sigma(w) \mid e(w))$ appears only once in (1) for each word type w. Thus, the training of the spelling model is not *overly* influenced by frequent (and atypical) words like `the` and `a`,[6] but just as much as by rare words like `deforestation`. As a result, p_{spell} learns how typical word *types*—not typical *tokens*—are spelled. This is useful in predicting how *other types* will be spelled, which helps us regularize the embeddings of rare word types and predict the spellings of novel word types (§4 below).[7]

4 Open vocabulary by "spelling" UNK

Our spelling model not only helps us regularize the embeddings of rare words, but also allows us to handle *unknown words*, a long-standing concern in NLP tasks.[8] How?

As a language usually has a fixed known alphabet (so the held-out data will at least not contain unknown characters), a common approach is to model character sequences instead of word sequences to begin with (Sutskever, Martens, and Hinton 2011). However, such a model does not explicitly represent word units, does not respect duality of patterning (§ 3.1), and thus may have a harder time learning syntactic and semantic patterns at the sentence level. For this reason, several recent approaches have tried to combine character-level modeling with word-level modeling (see §6).

Our approach differs from this previous work because we have an explicit spelling model to use. Just as p_{spell} has an opinion about how to spell rare words, it also has an opinion about how to spell novel words. This allows the following trick. We introduce a special lexeme UNK, so that the vocabulary is now $\mathcal{V} = \{\text{UNK}, ①, ②, \ldots, ⓥ\}$ with finite size $v+1$. We refine our story of how the corpus is generated. First, the model again predicts a complete sequence of lexemes $w_1, \ldots, w_n$. In most cases, w_i is spelled out deterministically as $\sigma(w_i)$. However, if $w_i = \text{UNK}$, then we spell it out by sampling from $p_{\text{spell}}(\cdot \mid \vec{e}_i)$, where $\vec{e}_i$ is an appropriate embedding, explained below. The downside of this approach is that each UNK token samples a fresh spelling, so multiple tokens of an out-of-vocabulary word type are treated as if they were separate lexemes.

Recall that the spelling model generates a spelling *given an embedding*. So what embedding $\vec{e}_i$ should we use to generate this unknown word? Imagine the word had been in the vocabulary. Then, if the model had wanted to predict that word, $\vec{e}_i$ would have had to have a high dot product with the hidden state of the lexeme-level RNN at this time step, $\vec{h}$. So, clearly, the embedding that maximizes the dot product with the hidden state is just the hidden state itself.[9] It follows that we should sample the generated spelling $s \sim p(\cdot \mid \vec{h})$, using

[6]The striking difference between types and tokens is perhaps most visible with `th`. It is the most common character bigram in words of the Penn Treebank as preprocessed by Mikolov et al. (2010) when counting word *tokens*, but only appears in 156th place when counting word *types*. Looking at trigrams (with spaces) produces an even starker picture: `␣th`, `the`, `he␣` are respectively the 1st, 2nd, and 3rd most common trigrams when looking at tokens, but only the 292nd, 550th, and 812th (out of 5261) when considering types.

[7]Baayen and Sproat (1996) argue for using only the *hapax legomena* (words that only appear once) to predict the behavior of rare and unknown words. The Bayesian approach (MacKay and Peto 1995; Teh 2006) is a compromise: frequent word types are also used, but they have no more influence than infrequent ones.

[8]Often 5–10% of held-out word tokens in language modeling datasets were never seen in training data. Rates of 20–30% or more can be encountered if the model was trained on out-of-domain data.

[9]At least, this is the best $\vec{e}_i$ for which $||\vec{e}_i|| \leq ||\vec{h}||$ holds.

the current hidden state of the lexeme-level RNN.[10]

Continuing the generative story, the lexeme-level RNN moves on, but to simplify the inference we feed $e(\text{UNK})$ into the lexeme-level RNN to generate the next hidden state, rather than feeding in $\vec{h}$ (our guess of $e(\sigma^{-1}(s))$).[11]

Now we can expand the model described in §2 to deal with sequences containing unknown words. Our building blocks are two old factors from Eq. (1) and a new one:

the lexicon generation $\prod_{w\in\mathcal{V}}\left[p(e(w))\cdot p_{\text{spell}}(\sigma(w)\mid e(w))\right]$
predicts the spellings of in-vocabulary lexemes from their embeddings

the lexeme-level RNN $\prod_{i=1}^{n} p_{\text{LM}}(w_i \mid \vec{w}_{<i}, e)$
predicts lexeme w_i from the history $\vec{w}_{<i}$ summarized as $\vec{h}_i$

the spelling of an UNK *(new!)* $p_{\text{spell}}(s \mid \vec{h}')$
predicts the spelling s for an UNK lexeme that appears in a context that led the lexeme-level RNN to hidden state $\vec{h}'$

Using these we can again find the MAP estimate of our parameters, i.e., the (regularized) maximum likelihood (ML) solution, using the posterior that is proportional to the new joint (with the change from Eq. (1) in black):

$$p(\theta, e, \sigma, s_1 \cdots s_n) \;=\; p(\theta)\;\cdot \tag{3}$$
$$\prod_{w\in\mathcal{V}}\left[p(e(w))\cdot p_{\text{spell}}(\sigma(w)\mid e(w))\right]\cdot\prod_{i=1}^{n} p_{\text{LM}}(w_i \mid \vec{w}_{<i}, e)\;\cdot$$
$$\prod_{i:\; w_i=\text{UNK}} p_{\text{spell}}(s_i \mid \vec{h}_i)$$

where $s_1, \ldots, s_n$ are the observed spellings that make up the corpus and $w_i = \sigma^{-1}(s_i)$ if defined, i.e., if there is a $w \in \mathcal{V}$ with $\sigma(w) = s_i$, and $w_i = \text{UNK}$ otherwise.

The entire model is depicted in Fig. 1. We train it using SGD, computing the different factors of Eq. (3) in an efficient order (implementation details are presented in Appendix A).

5 Experiments

We will now describe the experiments we perform to show that our approach works well in practice.[12] A detailed discussion of all hyperparameters can be found in Appendix B.

5.1 Datasets

We evaluate on two open-vocabulary datasets, *WikiText-2* (Merity et al. 2017) and the *Multilingual Wikipedia Corpus* (Kawakami, Dyer, and Blunsom 2017).[13] For each corpus, we follow Kawakami, Dyer, and Blunsom and replace characters that appear fewer than 25 times by a special symbol ◇.[14]

WikiText-2 The WikiText-2 dataset (Merity et al. 2017) contains more than 2 million tokens from the English Wikipedia. We specifically use the "raw" version, which is tokenized but has no UNK symbols (since we need the spellings of *all* words).

The results for WikiText-2 are shown in Table 1 in the form of bits per character (bpc). Our full model is denoted **FULL**. The other rows report on baselines (§5.2) and ablations (§5.3), which are explained below.

Multilingual Wikipedia Corpus The Multilingual Wikipedia Corpus (Kawakami, Dyer, and Blunsom 2017) contains 360 Wikipedia articles in English, French, Spanish, German, Russian, Czech, and Finnish. However, we re-tokenize the dataset, not only splitting on spaces (as Kawakami, Dyer, and Blunsom do) but also by splitting off each punctuation symbol as its own token. This allows us to use the same embedding for a word regardless of whether it has adjacent punctuation. For fairness in comparison, we ensure that our tokenizer preserves all information from the original character sequence (i.e., it is reversible). The exact procedure—which is simple and language-agnostic—is described in Appendix E, with accompanying code.

The results for the MWC are shown in Table 2 in the form of bits per character (bpc).

5.2 Comparison to baseline models

The first baseline model is a purely character-level RNN language model (**PURE-CHAR**). It is naturally open-vocabulary (with respect to words; like all models we evaluate, it does assume a closed character set). This baseline reaches by far the worst bpc rate on the held-out sets, perhaps because it works at too short a time scale to capture long-range dependencies.

A much stronger baseline—as it turns out—is a subword-level RNN language model (**PURE-BPE**). It models a sequence of *subword units*, where each token in the corpus is split into one or more subword units by *Byte Pair Encoding* (BPE), an old compression technique first used for neural machine translation by Sennrich, Haddow, and Birch (2016). This gives a kind of interpolation between a word-level model and a character-level model. The set of subword units is finite and determined from training data, but includes all characters in Σ, making it posssible to explain any novel word in held-out data. The size of this set can be tuned by specifiying the number of BPE "merges."[15] To our surprise, it is the strongest competitor to our proposed model, even outperforming it on the MWC. One has to wonder why BPE has not (to our knowledge) been tried previously as an open-vocabulary language model, given its ease of use and general applicability.

Notice, however, that even when PURE-BPE performs well as a language model, it does not provide *word embeddings* to use in other tasks like machine translation, parsing, or entailment. We cannot extract the usual static type embeddings

[10] Note that an in-vocabulary token can now be generated in two ways, as the spelling of a known lexeme or of UNK. Appendix D discusses this (largely inconsequential) issue.

[11] This makes the implementation simpler and faster. One could also imagine feeding back, e.g., the final hidden state of the speller.

[12] Code at github.com/sjmielke/spell-once.

[13] Unlike much previous LM work, we do not evaluate on the Penn Treebank (PTB) dataset as preprocessed by Mikolov et al. (2010) as its removal of out-of-vocabulary words makes it fundamentally unfit for open-vocabulary language model evaluation.

[14] This affects fewer than 0.03% of character tokens of WikiText-2 and thus does not affect results in any meaningful way.

[15] A segmented example sentence from WikiText-2 is "The |ex|os|kel|eton |is |gener|ally |blue". Technically we do not model the string, but the specific segmented string chosen by BPE. Modeling the string would require marginalizing over all possible segmentations (which is intractable to do exactly with a neural language model); more discussion on that in Appendix D.2.

WikiText-2	*dev*				*test*
types w/ count	0	[1, 100)	[100; ∞)	all	all
# of such tokens	7116	47437	163077	Σ	Σ
PURE-CHAR	**3.89**	2.08	1.38	1.741	1.775
PURE-BPE	4.01	1.70	**1.08**	1.430	1.468
ONLY-REG	4.37	1.68	1.10	1.452	1.494
SEP-REG	4.17	1.65	1.10	1.428	1.469
NO-REG	4.14	1.65	1.10	1.426	1.462
1GRAM	5.09	1.73	1.10	1.503	1.548
UNCOND	4.13	1.65	1.10	1.429	1.468
FULL	4.00	**1.64**	1.10	**1.416**	**1.455**
HCLM	–	–	–	1.625	1.670
HCLMcache	–	–	–	1.480	1.500

Table 1: Bits per character (lower is better) on the dev and test set of **WikiText-2** for our model and baselines, where FULL refers to our main proposed model and HCLM and HCLMcache refer to Kawakami, Dyer, and Blunsom (2017)'s proposed models. All our hybrid models use a vocabulary size of 50000, PURE-BPE uses 40000 merges (both tuned from Fig. 2). All pairwise differences except for those between PURE-BPE, UNCOND, and SEP-REG are statistically significant (paired permutation test over all 64 articles in the corpus, $p < 0.011$).

from it, nor is it obvious how to create dynamic per-token embeddings like the *contextualized embeddings* of Peters et al. (2018). Our model allows for both, namely $e(w)$ and $\vec{h}_i$.

Finally, we also compare against the character-aware model of Kawakami, Dyer, and Blunsom (2017), both without (**HCLM**) and with their additional cache (**HCLMcache**). To our knowledge, that model has the best previously known performance on the *raw* (i.e., open-vocab) version of the WikiText-2 dataset, but we see in both Table 1 and Table 2 that our model and the PURE-BPE baseline beat it.

5.3 Analysis of our model on WikiText-2

Ablating the training objective How important are the various influences on p_{spell}? Recall that p_{spell} is used to relate embeddings of in-vocabulary types to their spellings at training time. We can omit this *regularization* of in-vocabulary embeddings by dropping the second factor of the training objective, Eq. (3), which gives the **NO-REG** ablation. p_{spell} is also trained explicitly to spell out UNK tokens, which is how it will be used at test time. Omitting this part of the training by dropping the fourth factor gives the **ONLY-REG** ablation.

We can see in Table 1 that neither NO-REG nor ONLY-REG performs too well (no matter the vocabulary size, as we will see in Figure 2). That is, the spelling model benefits from being trained on both in-vocabulary types and UNK tokens.

To tease apart the effect of the two terms, we evaluate what happens if we use two separate spelling models for the second and fourth factors of Eq. (3), giving us the **SEP-REG** ablation. Now the in-vocabulary words are spelled from a different model and do not influence the spelling of UNK.[16]

[16] Though this prevents sharing statistical strength, it might actually be a wise design if UNKs are in fact spelled differently (e.g., they tend to be long, morphologically complex, or borrowed).

Interestingly, SEP-REG does not perform better than NO-REG (in Fig. 2 we see no big difference), suggesting that it is not the "smoothing" of embeddings using a speller model that is responsible for the improvement of FULL over NO-REG, but the benefit of training the speller on more data.[17]

Speller architecture power We also compare our full model (FULL) against two ablated versions that simplify the spelling model: **1GRAM**, where $p(\sigma(w)) \propto \prod_{i=1}^{|\sigma(w)|} q(\sigma(w)_i)$ (a learned unigram distribution q over characters instead of an RNN) and **UNCOND**, where $p(\sigma(w)) \propto p_{\text{spell}}(\sigma(w) \mid \vec{0})$, (the RNN character language model, but without conditioning on a word embedding).

In Table 1, we clearly see that as we go from 1GRAM to UNCOND to FULL, the speller's added expressiveness improves the model.

Rare versus frequent words It is interesting to look at bpc broken down by word frequency,[18,19] shown in Table 1. The first bin contains (held-out tokens of) words that were never seen during training, the second contains words that were only rarely seen (about half of them in $\mathcal{V}$), and the third contains frequent words. Unsurprisingly, rarer words generally incur the highest loss in bpc, although of course their lower frequency does limit the effect on the overall bpc.

On the frequent words, there is hardly any difference among the several models—they can all memorize frequent words—except that the PURE-CHAR baseline performs particularly badly. Recall that PURE-CHAR has to re-predict the spelling of these often irregular types each time they occur. Fixing this was the original motivation for our model.

On the infrequent words, PURE-CHAR continues to perform the worst. Some differences now emerge among the other models, with our FULL model winning. Even the ablated versions of FULL do well, with 5 out of our 6 beating both baselines. The advantage of our systems is that they create lexical entries that memorize the spellings of all in-vocabulary training words, even infrequent ones that are rather neglected by the baselines.

On the novel words, our 6 systems have the same relative ordering as they do on the infrequent words. The surprise in this bin is that the baseline systems do extremely well, with PURE-BPE nearly matching FULL and PURE-CHAR beating it, even though we had expected the baseline models to be too biased toward predicting the spelling of frequent words. Note, however, that p_{spell} uses a weaker LSTM than p_{LM} (fewer nodes and different regularization), which may explain the difference.

[17] All this, of course, is only evaluated with the hyperparameters chosen for FULL. Retuning hyperparameters for every condition might change these results, but is infeasible.

[18] We obtain the number for each frequency bin by summing the contextual log-probabilities of the tokens whose types belong in that bin, and dividing by the number of characters of all these tokens. (For the PURE-CHAR and PURE-BPE models, the log-probability of a token is a sum over its characters or subword units.)

[19] Low bpc means that the model can predict the tokens in this bin from their *left* contexts. It does not also assess whether the model makes good use of these tokens to help predict their right contexts.

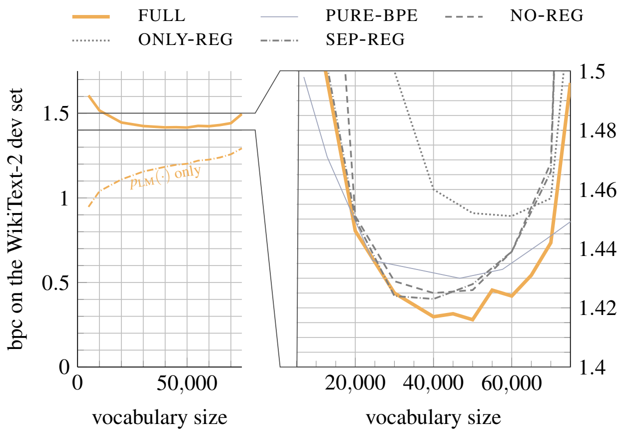

Figure 2: Bits-per-character (lower is better) on WikiText-2 dev data as a function of vocabulary size. Left: The total cross-entropy is dominated by the third factor of Eq. (3), p_{LM}, the rest being its fourth factor. Right (zoomed in): baselines.

Vocabulary size as a hyperparameter In Fig. 2 we see that the size of the vocabulary—a hyperparameter of both the PURE-BPE model (indirectly by the number of merges used) and our FULL model and its ablations—does influence results noticeably. There seems to be a fairly safe plateau when selecting the 50000 most frequent words (from the raw WikiText-2 vocabulary of about 76000 unique types), which is what we did for Table 1. Note that at *any* vocabulary size, both models perform far better than PURE-CHAR, whose bpc of 1.775 is far above the top of the graph.

Figure 2 also shows that as expected, the loss of the FULL model (reported as bpc on the entire dev set) is made up mostly of the cross-entropy of p_{LM}. This is especially so for larger vocabularies, where very few UNKs occur that would have to be spelled out using p_{spell}.

5.4 Results on the multilingual corpus

We evaluated on each MWC language using the system and hyperparameters that we had tuned on WikiText-2 development data. Even the vocabulary size stayed fixed at 60000.[20]

Frustratingly, lacking tuning to MWC, we do not outperform our own (novel) BPE baseline on MWC. We perform at most equally well, even when leveling the playing field through proper tokenization (§5.1). Nevertheless we outperform the best model of Kawakami, Dyer, and Blunsom (2017) on most datasets, even when using the space-split version of the data (which, as explained in §5.1, hurts our models).

Interestingly, the datasets on which we lose to PURE-BPE are Czech, Finnish, and Russian—languages known for their morphological complexity. Note that PURE-BPE greatly benefits from the fact that these languages have a concatenative morphological system unlike Hebrew or Arabic. Explicitly incorporating morpheme-level information into our FULL model might be useful (cf. Matthews, Neubig, and Dyer (2018)). Our present model or its current hyperparameter settings (especially the vocabulary size) might not be as language-agnostic as we would like.

5.5 What does the speller learn?

Finally, Table 3 presents non-cherry-picked samples from p_{spell}, after training our FULL model on WikiText-2. p_{spell} seems to know how to generate appropriate random forms that appear to have the correct part-of-speech, inflectional ending, capitalization, and even length.

We can also see how the speller chooses to create forms *in context*, when trying to spell out UNK given the hidden state of the lexeme-level RNN. The model knows *when* and *how* to generate sensible years, abbreviations, and proper names, as seen in the example in the introduction (§1).[21] Longer, non-cherry-picked samples for several of our models can be found in Appendix F.

6 Related work

Unlike most previous work, we try to *combine* information about words and characters to achieve open-vocabulary modeling. The extent to which previous work achieves this is as shown in Table 4 and explained in this section.

Mikolov et al. (2010) first introduced a purely word-level (closed-vocab) RNN language model (later adapted to LSTMs by Sundermeyer, Schlüter, and Ney (2012)). Sutskever, Martens, and Hinton (2011) use an RNN to generate pure character-level sequences, yielding an open-vocabulary language model, but one that does not make use of the existing word structure.

Kim et al. (2016) and Ling et al. (2015) first combined the two layers by deterministically constructing word embeddings from characters (training the embedding function on tokens, not types, to "get frequent words right"—ignoring the issues discussed in §3). Both only perform language modeling with a closed vocabulary and thus use the subword information only to improve the estimation of the word embeddings (as has been done before by dos Santos and Zadrozny (2014)).

Another line of work instead augments a character-level RNN with word-level "impulses." Especially noteworthy is the work of Hwang and Sung (2017), who describe an architecture in which character-level and word-level models run in parallel from left to right and send vector-valued messages to each other. The word model sends its hidden state to the character model, which generates the next word, one character at a time, and then sends its hidden state back to update the state of the word model. However, as this is another example of *constructing* word embeddings from characters, it again overemphasizes learning frequent spellings (§3.2).

Finally, the most relevant previous work is the (independently developed) model of Kawakami, Dyer, and Blunsom (2017), where each word has to be "spelled out" using a character-level RNN if it cannot be directly copied from the

[20] Bigger vocabularies require smaller batches to fit our GPUs, so changing the vocabulary size would have complicated fair comparisons across methods and languages, as the batch size has a large influence on results. However, the optimal vocabulary size is presumably language- and dataset-dependent.

[21] Generated at temperature $T = 0.75$ from a FULL model with $|\mathcal{V}| = 20000$.

MWC		*en* dev	*en* test	*fr* dev	*fr* test	*de* dev	*de* test	*es* dev	*es* test	*cs* dev	*cs* test	*fi* dev	*fi* test	*ru* dev	*ru* test
space-split	*#types→ merges/vocab*	*195k → 60k*		*166k → 60k*		*242k → 60k*		*162k → 60k*		*174k → 60k*		*191k → 60k*		*244k → 60k*	
	PURE-BPE	1.50	1.439	1.40	1.365	1.49	1.455	1.46	1.403	1.92	1.897	1.73	1.685	1.68	1.643
	FULL	1.57	1.506	1.48	1.434	1.66	1.618	1.53	1.469	2.27	2.240	1.93	1.896	2.00	1.969
	HCLM	1.68	1.622	1.55	1.508	1.66	1.641	1.61	1.555	2.07	2.035	1.83	1.796	1.83	1.810
	HCLMcache	1.59	1.538	1.49	1.467	1.60	1.588	1.54	1.498	2.01	1.984	1.75	1.711	1.77	1.761
tokenize	*#types→ merges/vocab*	*94k → 60k*		*88k → 60k*		*157k → 60k*		*93k → 60k*		*126k → 60k*		*147k → 60k*		*166k → 60k*	
	PURE-BPE	**1.45**	**1.386**	**1.36**	**1.317**	**1.45**	**1.414**	**1.42**	**1.362**	**1.88**	**1.856**	**1.70**	**1.652**	**1.63**	**1.598**
	FULL	**1.45**	**1.387**	**1.36**	**1.319**	1.51	1.465	**1.42**	**1.363**	1.95	1.928	1.79	1.751	1.74	1.709

Table 2: Bits per character (lower is better) on the dev and test sets of the **MWC** for our model (FULL) and Kawakami, Dyer, and Blunsom (2017)'s HCLM and HCLMcache, both on the space-split version used by Kawakami, Dyer, and Blunsom (2017) and the more sensibly tokenized version. Values across all rows are comparable, since the tokenization is reversible and bpc is still calculated w.r.t. the number of characters in the original version. All our models did not tune the vocabulary size, but use 60000.

$\sigma(w)$	$s \sim p_{\text{spell}}(\cdot \mid e(w))$
`grounded`	`stipped`
`differ`	`coronate`
`Clive`	`Dickey`
`Southport`	`Strigger`
`Carl`	`Wuly`
`Chants`	`Tranquels`
`valuables`	`migrations`

Table 3: Take an in-vocabulary word w (non-cherry-picked), and compare $\sigma(w)$ to a random spelling $s \sim p_{\text{spell}}(\cdot \mid e(w))$.

	closed-vocab	open-vocab
(pure) words	Mikolov et al. (2010), Sundermeyer, Schlüter, and Ney (2012)	*-impossible-*
words + chars	Kim et al. (2016), Ling et al. (2015)	Kawakami, Dyer, and Blunsom (2017), Hwang and Sung (2017), ★
(pure) chars	*-impossible-*	Sutskever, Martens, and Hinton (2011)

Table 4: Contextualizing this work (★) on two axes

recent past. As in Hwang and Sung (2017), there is no fixed vocabulary, so words that have fallen out of the cache have to be re-spelled. Our hierarchical generative story—specifically, the process that generates the lexicon—handles the re-use of words more gracefully. Our speller can then focus on representative phonotactics and morphology of the language instead of generating frequent function words like `the` over and over again. Note that the use case that Kawakami et al. originally intended for their cache, the copying of highly infrequent words like `Noriega` that repeat on a very local scale (Church 2000), is not addressed in our model, so adding their cache module to our model might still be beneficial.

Less directly related to our approach of improving language models is the work of Bhatia, Guthrie, and Eisenstein (2016), who similarly realize that placing priors on word embeddings is better than compositional construction, and Pinter, Guthrie, and Eisenstein (2017), who prove that the spelling of a word shares information with its embedding.

Finally, in the highly related field of machine translation, Luong and Manning (2016) before the re-discovery of BPE proposed an open-vocabulary neural machine translation model in which the prediction of an UNK triggers a character-level model as a kind of "backoff." We provide a proper Bayesian explanation for this trick and carefully ablate it (calling it NO-REG), finding that it is insufficient, and that training on types (as suggested by far older research) is more effective for the task of language modeling.

7 Conclusion

We have presented a generative two-level open-vocabulary language model that can memorize spellings and embeddings of common words, but can also generate new word types in context, following the spelling style of in- and out-of-vocabulary words. This architecture is motivated by linguists' "duality of patterning." It resembles prior Bayesian treatments of type reuse, but with richer (LSTM) sequence models.

We introduced a novel, surprisingly strong baseline, beat it by tuning our model, and carefully analyzed the performance of our model, baselines, and a variety of ablations on multiple datasets. The conclusion is simple: pure character-level modeling is not appropriate for language, nor required for an open vocabulary. Our ablations show that the generative story our model is based on is superior to distorted or simplified models resembling previous ad-hoc approaches.

In future work, our approach could be used in other generative NLP models that use word embeddings. Our spelling model relates these embeddings to their spellings, which could be used to regularize embeddings of rare words (using the speller loss as another term in the generation process), or to infer embeddings for unknown words to help make closed-vocabulary models open-vocabulary. Both are likely to be extremely helpful in tasks like text classification (e.g., sentiment), especially in low-resource languages and domains.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. 1718846. Part of the research made use of computational resources at the Maryland Advanced Research Computing Center (MARCC). The authors would like to thank Benjamin Van Durme and Yonatan Belinkov for helpful suggestions on experiments, and Annabelle Carrell as well as the anonymous reviewers for their suggestions on improving the exposition.

References

Baayen, H., and Sproat, R. 1996. Estimating lexical priors for low-frequency syncretic forms. *Computational Linguistics* 22(2):155–166.

Bhatia, P.; Guthrie, R.; and Eisenstein, J. 2016. Morphological priors for probabilistic neural word embeddings. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 490–500. Austin, Texas: Association for Computational Linguistics.

Church, K. W. 2000. Empirical estimates of adaptation: The chance of two Noriegas is closer to $p/2$ than p^2. In *Proceedings of the 18th Conference on Computational Linguistics - Volume 1*, 180–186. Association for Computational Linguistics.

dos Santos, C., and Zadrozny, B. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 1818–1826.

Goldwater, S.; Griffiths, T. L.; and Johnson, M. 2006. Contextual dependencies in unsupervised word segmentation. In *Proc. of COLING-ACL*.

Goldwater, S.; Griffiths, T. L.; and Johnson, M. 2011. Producing power-law distributions and damping word frequencies with two-stage language models. *Journal of Machine Learning Research* 12(Jul):2335–2382.

Hockett, C. F. 1960. The origin of speech. *Scientific American* 203(3):88–97.

Hwang, K., and Sung, W. 2017. Character-level language modeling with hierarchical recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, 5720–5724. IEEE.

Kawakami, K.; Dyer, C.; and Blunsom, P. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. In *Proceedings of the 55th Annual Meeting of the ACL (Volume 1: Long Papers)*, 1492–1502. Vancouver, Canada: Association for Computational Linguistics.

Kim, Y.; Jernite, Y.; Sontag, D.; and Rush, A. M. 2016. Character-aware neural language models. In *AAAI*, 2741–2749.

Kudo, T. 2018. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 66–75. Melbourne, Australia: Association for Computational Linguistics.

Ling, W.; Dyer, C.; Black, A. W.; Trancoso, I.; Fermandez, R.; Amir, S.; Marujo, L.; and Luis, T. 2015. Finding function in form: Compositional character models for open vocabulary word representation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1520–1530. Lisbon, Portugal: Association for Computational Linguistics.

Luong, M.-T., and Manning, C. D. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers)*, 1054–1063. Berlin, Germany: Association for Computational Linguistics.

MacKay, D. J. C., and Peto, L. C. B. 1995. A hierarchical Dirichlet language model. *Journal of Natural Language Engineering* 1(3):289–308.

Matthews, A.; Neubig, G.; and Dyer, C. 2018. Using morphological knowledge in open-vocabulary neural language models. In *HLT-NAACL*.

McCarthy, J., and Prince, A. 1999. Faithfulness and identity in prosodic morphology. In Kager, R.; van der Hulst, H.; and Zonneveld, W., eds., *The Prosodic Morphology Interface*. Cambridge University Press. 269–309.

Merity, S.; Xiong, C.; Bradbury, J.; and Socher, R. 2017. Pointer sentinel mixture models. In *Proc. International Conference on Learning Representations*.

Merity, S.; Keskar, N. S.; and Socher, R. 2017. Regularizing and optimizing LSTM language models. *arXiv preprints* abs/1708.02182.

Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*.

Mochihashi, D.; Yamada, T.; and Ueda, N. 2009. Bayesian unsupervised word segmentation with nested pitman-yor language modeling. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 100–108. Suntec, Singapore: Association for Computational Linguistics.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *Proc. International Conference on Learning Representations*.

Pinter, Y.; Guthrie, R.; and Eisenstein, J. 2017. Mimicking word embeddings using subword rnns. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 102–112. Copenhagen, Denmark: Association for Computational Linguistics.

Saussure, F. d. 1916. *Course in General Linguistics*. Columbia University Press. English edition of June 2011, based on the 1959 translation by Wade Baskin.

Sennrich, R.; Haddow, B.; and Birch, A. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the ACL (Volume 1: Long Papers)*, 1715–1725. Berlin, Germany: Association for Computational Linguistics.

Sundermeyer, M.; Schlüter, R.; and Ney, H. 2012. LSTM neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

Sutskever, I.; Martens, J.; and Hinton, G. 2011. Generating text with recurrent neural networks. In Getoor, L., and Scheffer, T., eds., *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, 1017–1024. New York, NY, USA: ACM.

Teh, Y. W. 2006. A hierarchical bayesian language model based on Pitman-Yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the ACL*, 985–992. Sydney, Australia: Association for Computational Linguistics.