Training Hybrid Language Models by Marginalizing over Segmentations

Edouard Grave **Sainbayar Sukhbaatar** **Piotr Bojanowski** **Armand Joulin**
Facebook AI Research
{egrave,sainbar,bojanowski,ajoulin}@fb.com

Abstract

In this paper, we study the problem of hybrid language modeling, that is using models which can predict both characters and larger units such as character ngrams or words. Using such models, multiple potential segmentations usually exist for a given string, for example one using words and one using characters only. Thus, the probability of a string is the sum of the probabilities of all the possible segmentations. Here, we show how it is possible to marginalize over the segmentations efficiently, in order to compute the true probability of a sequence. We apply our technique on three datasets, comprising seven languages, showing improvements over a strong character level language model.

1 Introduction

Statistical language modeling is the problem of estimating a probability distribution over text data (Bahl et al., 1983). Most approaches formulate this problem at the word level, by first segmenting the text using a fixed vocabulary. A limitation of these methods is that they cannot generate new words, or process out of vocabulary words. A popular alternative is to directly model sequences at the character level. These models can potentially generate any sequence, and are thus sometimes referred to as *open vocabulary*. However, they tend to underperform compared to word level models when trained on the same data.

For these reasons, a few works have proposed hybrid models, that work both at the character and word level (or sometimes groups of characters). A first class of hybrid models switch between word and character level representations, depending on whether they predict that the upcoming word is in the vocabulary or not (Kawakami et al., 2017; Mielke and Eisner, 2019). For example, a first model can be trained on tokenized data, where out-of-vocabulary words are replaced by the <unk> token. A second model is then used to generate the character sequences corresponding to out-of-vocabulary words. Another approach, which does not require tokenization, is to process groups of characters, which are obtained based on linguistic knowledge or low level statistics. These include merging characters using mutual information (Mikolov et al., 2012) or the byte pair encoding algorithm (Sennrich et al., 2016). This approach first produces a segmentation for the text, and then learns a language model on it. However, some sequences have multiple possible segmentations, and a model considering a single one might underestimate the true probability of the sequence. Thus, it is important to marginalize over the set of segmentations to obtain the true probability of a sequence (van Merriënboer et al., 2017; Buckman and Neubig, 2018).

In this paper, we propose an alternative approach to address this limitation, and in particular, to train models by marginalizing over the set of segmentations. As the number of possible segmentations grows exponentially with the sequence size, using an efficient algorithm such as dynamic programming is important. Computing the representation of the context at the character level allows to apply dynamic programming to this problem, without using approximations. This technique was previously considered in the context of automatic speech recognition (Wang et al., 2017) or to copy tokens from the input for code generation (Ling et al., 2016). We evaluate our method on three datasets for character level language modeling, showing that adding n-grams to the predictions improve the perplexity of the model.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 1477–1482
Florence, Italy, July 28 - August 2, 2019.

2 Approach

The goal of character level language modeling is to learn a probability distribution over sequences of characters $c_1, ..., c_T$. Using the chain rule, such a distribution can be factorized as the product of the probability distribution of a character conditioned on its history:

$$p(c_1, ..., c_T) = \prod_{t=1}^{T} p(c_t \mid c_0, ..., c_{t-1}),$$

where c_0 is a special symbol indicating the beginning of the sequence. In this paper, we learn these conditional probability distributions using neural networks. For each time step t, a neural network builds a representation $\mathbf{h}_t$ from the history that is used to predict the upcoming character. This representation can be obtained from any architecture, such as feedforward (Bengio et al., 2003) or recurrent networks (Mikolov et al., 2010). We focus on the transformer network, recently introduced by Vaswani et al. (2017), because of its high performance on character level language modeling (Dai et al., 2018). We refer to Vaswani et al. (2017) for the details of this architecture.

2.1 Hybrid language models

Hybrid language models predict multiple tokens, instead of one, at each time step. One way to perform this is to add n-grams to the output vocabulary of the model. Under such models, a character sequence has multiple segmentations, and the model estimates its probability by summing the probability of all its segmentations. For example, if the model predicts bigrams in addition to characters, the word *dog* can be decomposed as

`[d], [o], [g]` or `[do], [g]` or `[d], [og]`.

Thus, the probability of the sequence of characters *dog* is given by

$$\begin{aligned} p(\texttt{dog}) &= p(\texttt{d}) \times p(\texttt{o} \mid \texttt{d}) \times p(\texttt{g} \mid \texttt{do}) \\ &\quad + p(\texttt{do}) \times p(\texttt{g} \mid \texttt{do}) + p(\texttt{d}) \times p(\texttt{og} \mid \texttt{d}). \end{aligned}$$

More formally, let us denote by $\mathcal{S}(\mathbf{c}_{1:T})$ the set of all possible segmentations of a given sequence $\mathbf{c}_{1:T} = c_1, ..., c_T$. Then, the probability of the character sequence is

$$p(\mathbf{c}_{1:T}) = p(c_1, ..., c_T) = \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c})} p(\mathbf{s}). \quad (1)$$

The set of all possible segmentations grows exponentially with the sequence size, making it impractical to evaluate this probability by directly summing over all segmentations.

2.2 Factorization of the segmentation probabilities

A segmentation $\mathbf{s}$ can be decomposed into a sequence $s_1, ..., s_K$ of consecutive atoms in the vocabulary on which we apply the chain rule to get:

$$p(\mathbf{s}) = \prod_{k=1}^{K} p(s_k \mid s_0, ..., s_{k-1}).$$

Using this factorization of the probability distribution, it is not possible to directly apply dynamic programming to compute the probability of a sequence. The reason is that the conditional distribution of symbols depends on the segmentation, preventing to reuse computation across different segmentations. For example, previous work proposed to use the segmentation both in the input and output of the model. The hidden representations $\mathbf{h}_t$ of the neural network were thus intrinsically linked to the segmentation, preventing to share computations. A potential workaround is to merge the different representations corresponding to all the segmentations ending at the same character, for example by avergaging them (van Merriënboer et al., 2017; Buckman and Neubig, 2018). In our case, we use n-grams only in the output, making the representations $\mathbf{h}_t$ independent of the segmentations, and the application of dynamic programming straightforward.

To do so, we define the conditional distribution using characters, instead of the segmentation. Given a sequence $s_1, ..., s_K$ of n-grams, we introduce the concatenation operator concat, such that

$$\text{concat}(s_1, ..., s_K) = c_1, ..., c_J$$

corresponds to the sequence of J characters that compose the segmentation sequence. For example, the two segmentations `[do], [g], [s]` and `[d], [og], [s]` of the word *dogs* share the same output from the concat operator:

concat(`[do], [g], [s]`) = `d, o, g, s`,
concat(`[d], [og], [s]`) = `d, o, g, s`.

We now define the conditional distribution as

$$p(s_k \mid \mathbf{s}_{1:k-1}) = p(s_k \mid \text{concat}(\mathbf{s}_{1:k-1})). \quad (2)$$

This reformulation is exact under the conditional independence assumption, i.e., that the symbol at position t in the character sequence is independent of the segmentation, given the characters up to time $t-1$. In the next section, we show how, under this assumption, the probability of a sequence can be computed with dynamic programming.

2.3 Dynamic programming

For this section, we restrict ourselves to predicting characters and bigrams for simplicity. However, our approach is straightforward to apply to n-grams or words. Given a sequence of character $\mathbf{c} = c_1, ..., c_T$, all segmentations end with either the character c_T or the bigram $c_{T-1}c_T$. More precisely, we can decompose the probability of $\mathbf{c}$ as:

$$p(\mathbf{c}_{1:T}) = \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c}_{1:T-1})} p(c_T \mid \mathbf{s}) p(\mathbf{s}) + \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c}_{1:T-2})} p(c_{T-1}c_T \mid \mathbf{s}) p(\mathbf{s}).$$

Using the reformulation of the conditional probability of Eq. (2) under the conditional independence assumption on segmentations, we get

$$p(\mathbf{c}_{1:T}) = \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c}_{1:T-1})} p(c_T \mid \mathbf{c}_{1:T-1}) p(\mathbf{s}) + \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c}_{1:T-2})} p(c_{T-1}c_T \mid \mathbf{c}_{1:T-2}) p(\mathbf{s}).$$

We now move the conditional probabilities out of the sums:

$$p(\mathbf{c}_{1:T}) = p(c_T \mid \mathbf{c}_{1:T-1}) \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c}_{1:T-1})} p(\mathbf{s}) + p(c_{T-1}c_T \mid \mathbf{c}_{1:T-2}) \sum_{\mathbf{s} \in \mathcal{S}(\mathbf{c}_{1:T-2})} p(\mathbf{s}).$$

Finally, using Eq. (1), we obtain a recurrence relation over the sequence probabilities:

$$p(\mathbf{c}_{1:T}) = p(c_T \mid \mathbf{c}_{1:T-1}) p(\mathbf{c}_{1:T-1}) + p(c_{T-1}c_T \mid \mathbf{c}_{1:T-2}) p(\mathbf{c}_{1:T-2}).$$

We can thus optimize over all the possible segmentations using dynamic programing.

2.4 Conditional distribution of symbols

In this section, we briefly describe how to model the conditional probability distribution of symbols, either characters or ngrams, given the character history. We learn a character level neural network to encode the context with hidden representation $\mathbf{h}_t$ for each character t. The probability distribution of the next symbol, either a character or a n-gram, is obtained by taking the softmax over the full vocabulary, which includes both characters and longer elements:

$$p(\cdot \mid c_0, ..., c_{t-1}) = \texttt{softmax}(\mathbf{W}\mathbf{h}_t).$$

Note that we get only one probability distribution over n-grams of different lengths.

2.5 Training procedure

We learn the parameters of our model by minimizing the negative log-likelihood of the training data, using the probability introduced in Eq. (1). We rely on automatic differentiation to compute the gradients, and thus, only need to implement the forward computation, which relies on dynamic programming. Empirically, we observed that training a model from scratch with this objective is sometimes unstable. We thus consider an alternative training objective, used at the beginning of training. For each position, this loss is equal to the sum of the negative log-probabilities of the n-grams corresponding to that position. More formally, given a sequence of length T, this objective is equal to

$$-\sum_{t=1}^{T} \sum_{n=1}^{N-1} \log\left(p(\mathbf{c}_{t:t+n} \mid \mathbf{c}_{1:t-1})\right),$$

and N is the size of the longest n-grams considered (we can pad n-grams when $t + n > T$ or exclude them from this loss).

3 Experiments

In this section, we describe the experiments that we performed to evaluate our approach on character level language modeling.

3.1 Datasets

We consider 3 datasets derived from Wikipedia articles, but with different preprocessing.

Text8. The `text8` dataset of M. Mahoney[1] contains 100 million characters from Wikipedia, and was preprocessed to only contains the lowercase letters `a-z` and nonconsecutive spaces.

[1] http://mattmahoney.net/dc/textdata

Model	Cs	De	En	Es	Fi	Fr	Ru	Avg.
HCLM (Kawakami et al., 2017)	2.035	1.641	1.622	1.555	1.796	1.508	1.810	1.710
HCLM cache (Kawakami et al., 2017)	1.984	1.588	1.538	1.498	1.711	1.467	1.761	1.649
Full (Mielke and Eisner, 2019)	2.240	1.618	1.506	1.469	1.896	1.434	1.969	1.733
Full (tok) (Mielke and Eisner, 2019)	1.928	1.465	1.387	1.363	1.751	1.319	1.709	1.560
BPE (Mielke and Eisner, 2019)	1.897	1.455	1.439	1.403	1.685	1.365	1.643	1.555
BPE (tok) (Mielke and Eisner, 2019)	1.856	1.414	1.386	1.362	1.652	1.317	1.598	1.512
Transformer baseline	1.777	1.406	1.393	1.37	1.525	1.34	1.616	1.489
Our approach	**1.715**	**1.352**	**1.341**	**1.326**	**1.445**	**1.299**	**1.508**	**1.426**

Table 1: Test set bpc on the `MWC` dataset. The hyperparameters for our method are chosen on the validation set of `WikiText2`. Note that Mielke and Eisner (2019) applied the BPE baseline and their method to both tokenized and non-tokenized data. All the other methods were applied on non-tokenized data only.

Model	Test
BN LSTM (Cooijmans et al., 2016)	1.36
HM LSTM (Chung et al., 2016)	1.29
RHN (Zilly et al., 2017)	1.27
Large mLSTM (Krause et al., 2016)	1.27
12L Transf. (Al-Rfou et al., 2018)	1.18
Transformer baseline	1.176
Our approach	**1.156**

Table 2: Test set bpc on the `text8` dataset.

Model	Test
HCLM (Kawakami et al., 2017)	1.670
HCLM cache (Kawakami et al., 2017)	1.500
BPE (Mielke and Eisner, 2019)	1.468
Full (Mielke and Eisner, 2019)	1.455
Transformer baseline	1.417
Our approach	**1.366**

Table 3: Test set bpc on the `WikiText2` dataset.

WikiText2. The `WikiText2` dataset was introduced by Merity et al. (2017) with a different preprocessing from `text8`: numbers, capital letters and special characters are kept. The vocabulary size is 1152.[2] We use the *raw* version of the dataset, which is tokenized but where rare words are not replaced by the `<unk>` token. The training data contains 10.9 millions characters.

MWC. The multilingual Wikipedia corpus (`MWC`) of Kawakami et al. (2017) is very similar in size and preprocessing as `WikiText2`, but contains documents in 7 languages: Czech (`cs`), German (`de`), English (`en`), Spanish (`es`), Finnish (`fi`), French (`fr`) and Russian (`ru`). Unlike `Wikitext2`, the `MWC` dataset is not tokenized. The training sets range from 6.1M characters for Czech to 15.6M characters for English, and we refer the reader to Kawakami et al. (2017) for detailed statistics on this corpus.[3]

[2] As opposed to previous work, we keep all characters that appears in the train, validation or test splits of the data.

[3] Again, we keep all characters that appears in the data.

3.2 Technical details

Following recent work on character language modeling with transformers, we use a model with 12 layers of dimension 512, and 4 attention heads. We use a feedforward block of dimension 2048 for `MWC` and `WikiText2`, and 3072 for `text8`. We set the attention length to 512, and the batch size to 8. We use Adagrad (Duchi et al., 2011) to learn the parameters of our models. Following Vaswani et al. (2017), we start with a learning rate of 0, increase it linearly for k timesteps, then keep it constant, before halving it at every epochs for the last 10 epochs. We use a learning rate of 0.04 and warmup of $16k$ steps for the `text8` dataset, and a learning rate of 0.025 and warmup of $8k$ steps for the `WikiText2` and `MWC` datasets. In order to have an efficient model at inference time, we use the caching mechanism from Dai et al. (2018) to store the hidden representations of the previous batch, as well as relative position weights. We pick a dropout rate in the set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, using the validation set. In the experiments, we use n-grams of size up to 4, excluding n-grams appearing less than 200 times (1000 times on `text8`) to limit the

size of the vocabulary. Thus, segmentations which contain out-of-vocabulary n-grams have a probability equal to zero.

3.3 Results

In Table 1, we report results on the `MWC` dataset, comparing our approach to the models of Kawakami et al. (2017) and Mielke and Eisner (2019). Our approach significantly improves the state of the art on this dataset. Some of the gain is due to the change of architecture for a transformer. However, we observe that marginalizing over segmentations also improves over the character level transformer baseline, showing the benefits of our method. Finally, as opposed to Mielke and Eisner (2019), our approach does not need to tokenize the data to perform well on this dataset.

In Table 2 and Table 3, we report results on the `text8` and `wikitext2` datasets respectively. As for the `MWC` dataset, our approach significantly improves the perplexity compared to our character level transformer baseline. Note that the state of the art on `text8` is 1.08 bpc on the test set with a 24-layer transformer network (Dai et al., 2018). This model is significantly larger than ours, containing almost 8 times more parameters.

4 Conclusion

In this paper, we study the problem of hybrid language modeling, where models can predict n-grams, instead of unigrams only. A technical challenge for learning these models is that a given string can have multiple segmentations, and one needs to marginalize over the set of segmentations. We introduce a simple technique to do so, allowing to apply dynamic programming for learning and inference. Using this approach, we improve the state of the art on the `MWC` and `WikiText2` datasets, used to evaluate hybrid language models.

Acknowledgements

We thank the anonymous reviewers for their helpful comments.

References

Rami Al-Rfou, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones. 2018. Character-level language modeling with deeper self-attention. *arXiv preprint arXiv:1808.04444*.

Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. 1983. A maximum likelihood approach to continuous speech recognition. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):179–190.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *JMLR*.

Jacob Buckman and Graham Neubig. 2018. Neural lattice language models. *TACL*.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. 2016. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*.

Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2018. Transformer-xl: Language modeling with longer-term dependency.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. In *ACL*.

Ben Krause, Liang Lu, Iain Murray, and Steve Renals. 2016. Multiplicative lstm for sequence modelling. *arXiv preprint arXiv:1609.07959*.

Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*.

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

Bart van Merriënboer, Amartya Sanyal, Hugo Larochelle, and Yoshua Bengio. 2017. Multiscale sequence modeling with a learned dictionary. *arXiv preprint arXiv:1707.00762*.

Sebastian J Mielke and Jason Eisner. 2019. Spell once, summon anywhere: A two-level open-vocabulary language model. In *AAAI*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

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)*, 8.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *ACL*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Chong Wang, Yining Wang, Po-Sen Huang, Abdelrahman Mohamed, Dengyong Zhou, and Li Deng. 2017. Sequence modeling via segmentations. In *ICML*.

Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2017. Recurrent highway networks. In *ICML*.