Modeling the Unigram Distribution

Irene Nikkarinen[*,ð,ɤ] **Tiago Pimentel**[*,ð] **Damián E. Blasi**[ə,ɘ,ɒ] **Ryan Cotterell**[ð,ʕ]
[ð]University of Cambridge [ɤ]Yle [ə]Harvard University
[ɘ]MPI for Evolutionary Anthropology [ɒ]HSE University [ʕ]ETH Zürich
irene.nikkarinen@gmail.com, tp472@cam.ac.uk
dblasi@fas.harvard.edu, ryan.cotterell@inf.ethz.ch

Abstract

The unigram distribution is the non-contextual probability of finding a specific word form in a corpus. While of central importance to the study of language, it is commonly approximated by each word's sample frequency in the corpus. This approach, being highly dependent on sample size, assigns zero probability to any out-of-vocabulary (oov) word form. As a result, it produces negatively biased probabilities for any oov word form, while positively biased probabilities to in-corpus words. In this work, we argue in favor of properly modeling the unigram distribution—claiming it should be a central task in natural language processing. With this in mind, we present a novel model for estimating it in a language (a neuralization of Goldwater et al.'s (2011) model) and show it produces much better estimates across a diverse set of 7 languages than the naïve use of neural character-level language models.

1 Introduction

Neural networks have yielded impressive gains in sentence-level language modeling across a typologically diverse set of languages (Mikolov et al., 2010; Kalchbrenner et al., 2016; Merity et al., 2018; Melis et al., 2018; Cotterell et al., 2018). Similarly, neural networks constitute the state of the art in modeling the distribution over a language's word types (Pimentel et al., 2020), outperforming non-neural generative models such as Futrell et al.'s (2017) with character-level models. This paper focuses on a less-researched task that is halfway between sentence-level language modeling and word type distributions: Modeling the **unigram distribution**, the distribution over word tokens in a language consisting of the probability of a word's form as well as its frequency in the language. In particular, as opposed to sentence-level modeling, the unigram distribution does not consider contextual information.

[*]Equal contribution

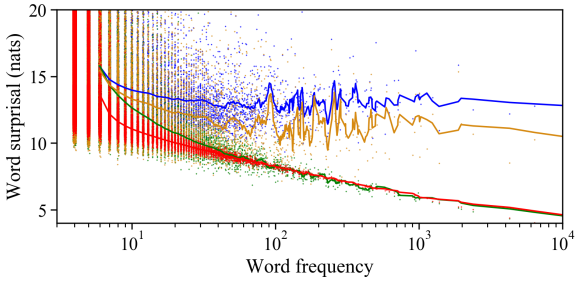

Figure 1: Word-level surprisal in Finnish under our **two-stage** model, two baseline LSTMs trained with either word **type** or **token** data, and another LSTM called the **generator**, trained on an interpolation of both. Lines depict rolling averages.

The unigram distribution is a central object in the science of language from historical linguistics to psycholinguistics and beyond (Baayen et al., 2016; Diessel, 2017; Divjak, 2019). However, the majority of research on unigram distributions is based on identifying this distribution with sample frequency. This approach results in poor estimates, as it assigns zero probability to out-of-vocabulary words.[1] Further, it is highly dependent on sample size (Baayen, 2002)

The core contribution of our work is motivating the unigram distribution as a worthwhile objective for scientific inquiry—one which is currently understudied in the field. With that in mind, we also present a neuralization of Goldwater et al.'s (2011) two-stage model.[2] The gist of this approach is using two components to model the Zipfian distribution of word tokens in a language (Zipf, 1935) separately from its phono- or graphotactic distribution. The first component, termed the **adaptor**, is

[1]In the Turkish Wikipedia, for example, considering a training set of 8 million and a test set of 1 million tokens, 27.4% of test types and 5.3% of test tokens are out-of-vocabulary. Note that, according to Heaps' law, a similar behavior would be expected from corpora of any size (Herdan, 1960; Heaps, 1978).

[2]While Goldwater et al. (2011) acknowledge that their model could be used in various tasks of learning linguistic structure, they only present results in modeling morphology.

based on the Pitman–Yor process (PYP; Pitman and Yor, 1997), and has the ability to model the power-law behavior of word tokens. The second, termed the **generator**, leverages a character-level neural language model to capture structural patterns in written words, e.g. graphotactics and morphology.

Critically, naïvely training a character-level neural model in either types (i.e. unique word forms) or tokens (i.e. word forms in their original frequencies) should lead to degenerate results. Models trained on natural corpora (i.e. token data) should excel in modeling the most common words of a language, but might poorly approximate the set of infrequent word forms which individuals dynamically produce (e.g. through compositional morphology). On the other hand, training models on the collection of unique word forms (i.e. type data) would give equal weight to typical and atypical productions, potentially leading to poor performance on the most frequent forms, which any individual would recognize as part of their language. In the two-stage model, as we will show, our generator is trained on a dataset interpolated between types and tokens—modeling the nuance between frequent and infrequent word forms better.

By testing our model on a set of languages with diverse morphological and phonological characteristics, we find that it is capable of modeling both frequent and infrequent words, thus producing a better estimate of the unigram distribution than a character-level LSTM. The empirical superiority of our two-stage model is shown in Fig. 1, where the surprisal (i.e. the negative log-probability, measured in nats here) of each token is plotted under four different models for Finnish. Our proposed two-stage model achieves a lower or similar surprisal to the baselines on tokens with all frequencies—with similar patterns arising in all analyzed languages.[3]

2 The Unigram Distribution

The unigram distribution is a probability distribution over the possible word forms in a language's lexicon. This probability takes the frequency of a token into account, assigning larger probabilities to word forms which are more likely to be encountered in a language's utterances, thus differing from word type distributions, such as in Pimentel et al. (2020). It is also not conditioned on a word's context, as it considers each word token as a stand-alone unit, as opposed to the task of language modeling, e.g. Mikolov et al. (2010).

2.1 Complex Vocabularies

The composition of spoken vocabularies is structured according to a host of factors. Stemming from articulatory biases, each language has a set of constraints on what sequences of speech sounds can be valid words in it; this is termed the **phonotactics** of a language. Languages also exhibit small but non-negligible biases in the regular match of forms and meanings (Dingemanse et al., 2015; Pimentel et al., 2019, 2021b). Additionally, expectations about morphology can constrain the production or processing of a given word as belonging to a particular word class (as shown for instance in Jabberwocky- and wug-type tasks, Berko 1958, Hall Maudslay and Cotterell 2021).

While individuals often have strong intuitions about these patterns, their judgments are typically gradient rather than categorical (Hayes and Wilson, 2008; Gorman, 2013). The effective set of words that naturally occur in linguistic productions are known to be extremely diverse in their composition. Models deployed to explain and predict typical word forms in a given language might fail at capturing these corners of the space of possible forms. If the goal is to produce ecologically valid models that could approximate actual cognitive processes, these atypical forms should be efficiently learned in addition to the most typical productions.

2.2 Imbalanced Frequencies

Zipf (1935) popularized the observation that the frequency of a word in a corpus is inversely proportional to its rank, approximately following a power-law distribution. As such, a small subset of the most common word types dominate the corpus. These extremely frequent words tend to be short in length and exceptionally archaic, in the sense that they preserve traces of previous phonotactic and phonological profiles that might have ceased to be productive. This is particularly relevant when we consider scenarios where substantial portions of the vocabulary might have been borrowed from different sources over time. English is a textbook example: Williams (1986) reports that French, Latin, Germanic and Greek account for

[3] As a final contribution of our work, the code used in this paper is available at https://github.com/irenenikk/modelling-unigram. We hope this will encourage future work in psycholinguistics to use the model to accurately investigate the effects of unigram probabilities in rare words.

29%, 29%, 26% and 6% of all words' origins in the vocabulary (plus a remaining 10% of diverse origin). The most frequent portion of the vocabulary preserves the most the original West Germanic forms, consisting largely of articles, prepositions, pronouns, and auxiliaries. Further, irregular inflections tend to be more common in these highly frequent words (Ackerman and Malouf, 2013; Cotterell et al., 2019). This observation might invite one to omit frequency information from training data, i.e. to use types, in order to balance out the role of the most frequent words.

On the other side of the frequency scale, however, any natural language data would have plenty of low-frequency words that reflect the open boundaries of the vocabulary. These might include nonce words (*blick*), expressive transformations of other words (*a looooooooooong summer*), specialized terms (*onabotulinumtoxina*), and names, among others. In addition, genuine orthographic misproductions (*langague*) will be present to some degree.

Finally, acronyms (*HTML*) will be present in all frequency bands. These should be particularly problematic to model, since they do not necessarily follow the language's graphotactics to any degree. There are also frequent and infrequent loanwords with different degrees of adjustment to the grapho- and phonotactics of the rest of the vocabulary. For instance, it has been estimated that 96% and 21% of English speakers know the Afrikaans-originated words *aardvark* and *aardwolf*, respectively (Brysbaert et al., 2019).[4] These are the only written word forms in English with a non-negligible frequency that display two letter '*a*'s in word-initial position.

This whimsical nature of the vocabulary of a language makes modeling the unigram distribution challenging: Naïvely training a model to capture word forms at either the token or type level is likely to give disproportionate emphasis to phonotactically unrepresentative words. However, this is also why its modeling is a worthwhile task—it captures both frequent and rare productions, combining form probability with frequency information.

3 Modeling the Unigram Distribution

Our work neuralizes Goldwater et al.'s (2011) two-stage model and employs it to modeling the unigram distribution.[5] The first component, termed the **generator**, is a model used to produce a set of i.i.d. word forms $\{\boldsymbol{\ell}_k\}_{k=1}^{K}$. The second component is termed **adaptor**, and it assigns each instance in the training set to a cluster $\{z_n\}_{n=1}^{N}$. Under this model, each token in a dataset has a corresponding cluster z_n which defines the token's word form $\boldsymbol{w}_n = \boldsymbol{\ell}_{z_n}$. We note that both word forms $\boldsymbol{\ell}$ and clusters $\mathbf{z}$ are latent variables, and only tokens $\boldsymbol{w}$ are observed during training.

Generator. The generator is a model which produces word forms; we use a character-level LSTM here (Hochreiter and Schmidhuber, 1997), as in:[6]

$$\{\boldsymbol{\ell}_k\}_{k=1}^{K} \sim p_\phi(\boldsymbol{\ell}) = \text{LSTM}(\boldsymbol{\ell}) \tag{1}$$

These word forms $\boldsymbol{\ell}_k$ are sampled i.i.d.—thus, the same word may be sampled more than once.

Adaptor. Each word form sampled from the generator corresponds to a cluster. The adaptor then assigns a frequency to each of the clusters according to a Pitman–Yor process:

$$p(z_n \mid \mathbf{z}_{<n}) \tag{2}$$
$$\propto \begin{cases} c_{<n}^{(z_n)} - a & 1 \le z_n \le K_{<n} \text{ (old cluster)} \\ a \cdot K_{<n} + b & z_n = K_{<n} + 1 \text{ (new cluster)} \end{cases}$$

where $0 \le a < 1$ and $0 \le b$ are hyperparameters of the PYP, $\mathbf{z}_{<n}$ are the previous cluster assignments, $K_{<n}$ is the current number of clusters with at least one token and $c_{<n}^{(z_n)}$ is the number of tokens previously assigned to cluster z_n. This adaptor, as a Pitman–Yor process, allows us to model the power-law distribution of word tokens.

Two-stage Model. Given a cluster assignment and the list of word forms, defining a token's form is deterministic: $p(\boldsymbol{w}_n \mid z_n, \boldsymbol{\ell}) = \mathbb{1}\{\boldsymbol{w}_n = \boldsymbol{\ell}_{z_n}\}$. Thus, our model factorizes a new token's probability into two terms:

$$p_{\text{model}}(\boldsymbol{w}) = \tag{3}$$
$$\underbrace{\frac{c_{\boldsymbol{w}} - \overbrace{n_{\boldsymbol{w}} \cdot a}^{\text{smoothing factor}}}{|\mathbf{z}| + b}}_{\text{smoothed 1-gram}} + \underbrace{\frac{(a \cdot K + b)}{|\mathbf{z}| + b}}_{\text{interpolation weight}} \cdot \underbrace{p_\phi(\boldsymbol{w})}_{\text{LSTM}}$$

where $c_{\boldsymbol{w}}$ is the number of occurrences of word form $\boldsymbol{w}$ in our training corpus and $n_{\boldsymbol{w}}$ is the number

[4] Aardvarks and aardwolves are African mammals.

[5] This same model was used in our contemporary work investigating lexicons' (non-)optimality (Pimentel et al., 2021a).

[6] See Pimentel et al. (2020) for more details on this graphotactics generative model.

of distinct clusters to which it has been assigned:

$$c_{\boldsymbol{w}} = \sum_{n=1}^{N} \mathbb{1}\{\boldsymbol{w} = \boldsymbol{\ell}_{z_n}\}, \tag{4}$$

$$n_{\boldsymbol{w}} = \sum_{k=1}^{K} \mathbb{1}\{\boldsymbol{w} = \boldsymbol{\ell}_{k}\} \tag{5}$$

In practice, the two-stage model acts as an interpolation between a smoothed 1-gram model, i.e. corpus frequencies, and an LSTM character model. Notably, this model learns per-word smoothing factors and its interpolation weight in an unsupervised manner through the PYP parameters' inference. The adaptor is fit using Gibbs sampling, and the generator is trained using a cross-entropy loss on the set of non-empty clusters produced by the adaptor. The generator is thus trained using a more balanced corpus where the proportion of the most frequent words is reduced; this can be seen as an interpolation between a type and a token dataset.[7]

4 Experiments.

Dataset. We use Wikipedia data and evaluate our model on the following languages: English, Finnish, Hebrew, Indonesian, Tamil, Turkish and Yoruba. These languages represent a typologically diverse set—with different levels of morphology, ranging from rich (e.g. Finnish) to poor (e.g. Yoruba), as well as distinct scripts and graphotactic patterns. In preprocessing, we first split the data into sentences and then into tokens using spaCy (Honnibal et al., 2020). We then sample 10^6 tokens as our training set for each language (except for Yoruba for which we had less data, see App. F for more details). From these, we build two distinct datasets: a **token dataset**, which corresponds to the list of word forms with their corpus frequency, and a **type dataset** containing the set of unique word forms in the data.

Evaluation. We measure the cross-entropy of our models on a held-out test set; this is the standard evaluation for language modeling. We approximate this cross-entropy using a sample mean estimate

$$\begin{aligned} \mathrm{H}(p) &\leq \mathrm{H}(p, p_{\mathrm{model}}) \\ &\approx -\frac{1}{N}\sum_{n=1}^{N} \log p_{\mathrm{model}}(\boldsymbol{w}_n) \end{aligned} \tag{6}$$

[7] This model's training is detailed in App. E. For a detailed description of the adaptor see Goldwater et al. (2011).

	Evaluated Model			
Language	Type	Token	Two-stage	Generator
English	12.50	9.04	**8.34**	11.86
Finnish	15.07	12.94	**11.85**	14.16
Hebrew	12.62	10.71	**10.20**	11.76
Indonesian	13.28	10.33	**9.55**	11.89
Tamil	14.24	12.66	**11.76**	13.29
Turkish	14.00	11.77	**10.86**	12.95
Yoruba	11.13	10.04	**9.19**	9.88

Table 1: Cross-entropy on the **unigram distribution**.

where we assume instances $\boldsymbol{w}_n$ are sampled from the true unigram distribution $p(\boldsymbol{w})$. Specifically, these token samples $\{\boldsymbol{w}_n\}_{n=1}^{N}$ take the form of the token dataset. The model with the lowest cross-entropy is the one that diverges the least from the true distribution.

Baseline Models. As neural networks yield state-of-the-art performance in language modeling tasks, we expect them to also do well with the unigram distribution. In fact, pseudo-text generated by LSTM-based language models reproduces Zipf's law to some extent (Takahashi and Tanaka-Ishii, 2017; Meister and Cotterell, 2021). Thus, we view state-of-the-art LSTM models as a strong baseline. We train a character-level LSTM language model (Pimentel et al., 2020) to directly approximate the unigram distribution by training it on the token dataset—modeling these tokens at the character level. As a second baseline, we train an LSTM on the type dataset. However, we expect this model to be outperformed by the token one in the unigram distribution task, as the information on word frequency is not available during its training. We do not use a word-level 1-gram model (i.e. the words' sample frequency) as a baseline here, since it results in an infinite cross-entropy for any test set containing oov words. We empirically compare four models: **two-stage**, **generator**, **token**, and **type**.

Modeling Tokens. Cross-entropy on the token test sets can be found in Tab. 1. These results show our two-stage model indeed creates a more accurate estimate of the unigram distribution, producing the smallest cross-entropy across all languages.

Frequent vs Infrequent Words. The weaknesses of the token and type models are evinced by Fig. 1. In line with our hypothesis, the token model achieves lower cross-entropy on the most common words, but fails to model the rare ones accurately.

	Evaluated Model				
Language	Type	Token	Two-stage	Generator	%
English	**18.23**	21.71	20.32	18.59	56%
Finnish	**19.76**	21.42	19.79	19.89	71%
Hebrew	**15.81**	17.76	17.10	16.30	56%
Indonesian	**18.52**	21.20	19.15	19.17	61%
Tamil	**18.77**	20.37	19.26	19.19	71%
Turkish	**18.36**	19.78	18.50	18.62	65%
Yoruba	15.44	17.69	**15.34**	16.28	67%

Table 2: Average surprisal for **singleton** types. Column % represents the ratio of singletons in the type test set.

	Evaluated Model			
Language	Type	Token	Two-stage	Generator
English	14.50	15.94	**13.24**	14.25
Finnish	15.19	14.89	**12.73**	14.80
Hebrew	13.36	13.80	**12.66**	13.20
Indonesian	14.72	15.50	**12.80**	14.64
Tamil	14.65	14.77	**12.95**	14.33
Turkish	14.73	14.41	**12.41**	14.33
Yoruba	11.13	11.97	**10.36**	11.16

Table 3: The average surprisal for **non-singleton** types.

The cross-entropy achieved by the type model does not change as much with word frequency, but is higher than the one achieved by the token model for most of the vocabulary. We also see that the two-stage model performs well across all word frequencies. Indeed, this model appears to behave similarly to the token model with frequent words, but obtains a lower cross-entropy on the rare ones, where the role of the generator in the estimated probability is emphasized. We suspect this is the reason behind the two-stage model's success.

The Long Tail. Fig. 1 also demonstrates that the entropy estimate for the rare words grows quickly and exhibits a large variance across models. This reflects the heterogeneous nature of the words that only appear a few times in a corpus. This part of the vocabulary is where the type model achieves the best results for all languages except Yoruba (see Tab. 2).[8] The fact that singletons (also known as *hapax legomena*), i.e. word forms which occur only once in the test set, form a large portion of the type dataset boosts the type model's performance on rare words. However, in the case of words appearing more than once (see Tab. 3) the two-stage model achieves the best results across languages. Furthermore, in these non-singleton words, the generator outperforms the type and token models in all languages except for Yoruba. This shows the utility of training the generator on an interpolation between types and tokens. In addition, we note that one may justifiably question whether properly modeling singletons is a desirable feature, since they are likely to contain unrepresentative word forms, such as typos, as discussed previously. Indeed, it appears that the two-stage model not only leads to tighter estimates of the unigram distribution, but also allows us to train a better graphotactics model; capable of modeling both frequent word forms as well as new productions.

Future Work. The results we present focus on analyzing the two-stage model. The generator, though, produces interesting results by itself, modeling non-singleton word forms better than the type and token models in most languages. This suggests that it might be better at modeling the graphotactics of a language than either of these baselines. Future work should explore if this indeed is the case.

5 Conclusion

In this work, we motivate the unigram distribution as an important task to both the psycholinguistics and natural language processing communities that has received too little attention. We present a two-stage model for estimating this distribution—a neuralization of Goldwater et al.'s (2011)—which is motivated by the complex makeup of vocabularies: This model defines the probability of a token by combining the probability of its appearance in the training corpus with the probability of its form. We have shown, through a cross-entropy evaluation, that our model outperforms naïve solutions and is capable of accurately modeling both frequent and infrequent words.

Acknowledgements

Damián E. Blasi acknowledges funding from the Branco Weiss Fellowship, administered by the ETH Zürich. Damián E. Blasi's research was also executed within the framework of the HSE University Basic Research Program and funded by the Russian Academic Excellence Project '5-100'.

Ethical Concerns

This paper highlights the importance of modeling the unigram distribution, and presents a model for

[8] We note that we used considerably less training data for Yoruba than for other languages.

the task. We do not foresee any reasons for ethical concerns, but we would like to note that the use of Wikipedia as a data source may introduce some bias into our experiments.

References

Farrell Ackerman and Robert Malouf. 2013. Morphological organization: The low conditional entropy conjecture. *Language*, 89(3):429–464.

R. Harald Baayen. 2002. *Word Frequency Distributions*, volume 18. Springer Science & Business Media.

R. Harald Baayen, Petar Milin, and Michael Ramscar. 2016. Frequency in lexical processing. *Aphasiology*, 30(11):1174–1220.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Jean Berko. 1958. The child's learning of English morphology. *Word*, 14(2-3):150–177.

Phil Blunsom, Trevor Cohn, Sharon Goldwater, and Mark Johnson. 2009. A note on the implementation of hierarchical Dirichlet processes. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 337–340, Suntec, Singapore. Association for Computational Linguistics.

Marc Brysbaert, Paweł Mandera, Samantha F. McCormick, and Emmanuel Keuleers. 2019. Word prevalence norms for 62,000 English lemmas. *Behavior Research Methods*, 51(2):467–479.

Ryan Cotterell, Christo Kirov, Mans Hulden, and Jason Eisner. 2019. On the complexity and typology of inflectional morphological systems. *Transactions of the Association for Computational Linguistics*, 7:327–342.

Ryan Cotterell, Sabrina J. Mielke, Jason Eisner, and Brian Roark. 2018. Are all languages equally hard to language-model? In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 536–541, New Orleans, Louisiana. Association for Computational Linguistics.

Holger Diessel. 2017. Usage-based linguistics. In *Oxford Research Encyclopedia of Linguistics*. Oxford University Press.

Mark Dingemanse, Damián E. Blasi, Gary Lupyan, Morten H. Christiansen, and Padraic Monaghan. 2015. Arbitrariness, iconicity, and systematicity in language. *Trends in Cognitive Sciences*, 19(10):603–615.

Dagmar Divjak. 2019. *Frequency in Language: Memory, Attention and Learning*. Cambridge University Press.

Richard Futrell, Adam Albright, Peter Graff, and Timothy O'Donnell. 2017. A generative model of phonotactics. *Transactions of the Association for Computational Linguistics*, 5(0):73–86.

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. 2011. Producing power-law distributions and damping word frequencies with two-stage language models. *Journal of Machine Learning Research*, 12(68):2335–2382.

Kyle Gorman. 2013. *Generative Phonotactics*. University of Pennsylvania.

Rowan Hall Maudslay and Ryan Cotterell. 2021. Do syntactic probes probe syntax? experiments with jabberwocky probing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 124–131, Online. Association for Computational Linguistics.

Bruce Hayes and Colin Wilson. 2008. A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, 39(3):379–440.

Harold Stanley Heaps. 1978. *Information Retrieval, Computational and Theoretical Aspects*. Academic Press.

Gustav Herdan. 1960. *Type–Token Mathematics: A Textbook of Mathematical Linguistics*, volume 4. Mouton.

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

Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. 2020. spaCy: Industrial-strength natural language processing in python.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aäron van den Oord, Alexander Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. In *arXiv preprint arXiv:1610.10099*.

Clara Meister and Ryan Cotterell. 2021. Language model evaluation beyond perplexity. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, Online. Association for Computational Linguistics.

Gábor Melis, Chris Dyer, and Phil Blunsom. 2018. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the International Speech Communication Association*, pages 1045–1048.

Radford M. Neal. 1993. Probabilistic inference using Markov chain Monte Carlo methods. Technical report, University of Toronto.

Tiago Pimentel, Arya D. McCarthy, Damian Blasi, Brian Roark, and Ryan Cotterell. 2019. Meaning to form: Measuring systematicity as information. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1751–1764, Florence, Italy. Association for Computational Linguistics.

Tiago Pimentel, Irene Nikkarinen, Kyle Mahowald, Ryan Cotterell, and Damián Blasi. 2021a. How (non-)optimal is the lexicon? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4426–4438, Online. Association for Computational Linguistics.

Tiago Pimentel, Brian Roark, and Ryan Cotterell. 2020. Phonotactic complexity and its trade-offs. *Transactions of the Association for Computational Linguistics*, 8:1–18.

Tiago Pimentel, Brian Roark, Søren Wichmann, Ryan Cotterell, and Damián Blasi. 2021b. Finding concept-specific biases in form–meaning associations. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4416–4425, Online. Association for Computational Linguistics.

Jim Pitman and Marc Yor. 1997. The two-parameter Poisson–Dirichlet distribution derived from a stable subordinator. *Annals of Probability*, 25(2):855–900.

Shuntaro Takahashi and Kumiko Tanaka-Ishii. 2017. Do neural nets learn statistical laws behind natural language? *PLOS ONE*, 12(12):1–17.

Greg C.G. Wei and Martin A. Tanner. 1990. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association*, 85(411):699–704.

Joseph M. Williams. 1986. *Origins of the English language*. Simon and Schuster.

George Kingsley Zipf. 1935. *The Psycho-biology of Language: An Introduction to Dynamic Philology*. Houghton Mifflin.

A Tuned hyperparameres for the two-stage model

Language	a	b
English	0.33	3,000
Finnish	0.36	90,000
Hebrew	0.40	55,000
Indonesian	0.48	180,000
Tamil	0.70	37,000
Turkish	0.33	95,000
Yoruba	0.08	156,000

Table 4: The optimized values of a and b for the analyzed languages.

B Hyperparameter Search

The same hyperparameters are used for both our baseline LSTMs and the generator. We use 3 layers, where embedding size is 128, hidden size is 512, and dropout probability is 0.33. Training the two-stage model takes a considerable amount of time (see Tab. 5). We are thus not capable of doing exhaustive hyperparameter tuning. Random search (Bergstra and Bengio, 2012) is used in tuning the values for a and b, where we run five training procedures considering ranges $a \in [0, 1)$, and $b \in [100, 200{,}000)$. We tune the hyperparameters for each language by minimizing the model's cross-entropy on the development set, training them on a subset of the training data with only $100{,}000$ tokens. The found optimal values of a and b are rounded to two decimal places and the thousands respectively. Our two-stage model is trained for five iterations of expectation–maximization.

C Training time with the two-stage model for each language

Language	Minutes
English	164
Finnish	170
Hebrew	175
Indonesian	173
Tamil	166
Turkish	174
Yoruba	56

Table 5: The training times for the two-stage model in each language. These times were obtained with a single NVIDIA Tesla P100 GPU.

D The development set cross-entropies on the unigram distribution

	Evaluated Model			
Language	Type	Token	Two-stage	Generator
English	12.50	9.04	**8.34**	9.18
Finnish	15.08	12.94	**11.88**	13.05
Hebrew	12.62	10.71	**10.20**	10.78
Indonesian	13.27	10.30	**9.53**	10.42
Tamil	14.22	12.65	**11.76**	12.75
Turkish	13.99	11.74	**10.83**	12.92
Yoruba	11.10	10.00	**9.13**	9.83

Table 6: Development set cross-entropy for the baseline models as well as our two-stage model evaluated on the **unigram distribution**.

E Inference

Unfortunately, there is no closed form solution for inferring the parameters of our two-stage model. In order to obtain a sample of cluster assignments and train the generator to match their labels, we estimate the parameters of both the generator and the adaptor concurrently, freezing one's parameters while training the other. We use a regime corresponding to the Monte Carlo Expectation-maximization (EM) algorithm to train the model (Wei and Tanner, 1990), which can be found in Algorithm 1. In the E-step, the function GIBBSSAMPLER returns the cluster assignments $\mathbf{z}$ and the dampened word dataset $\boldsymbol{\ell}$ obtained via Gibbs sampling from the PYP. We then use this dampened dataset to train the generator in the M-step.

Algorithm 1 Training the two-stage model

for i **in** RANGE(# Epochs) **do**
  // E-Step
  $\mathbf{z}, \boldsymbol{\ell} \sim \text{GIBBSSAMPLER}(a, b, p_\phi, \{\boldsymbol{w}_n\}_1^N)$
  // M-Step
  for $t = 1$ **up to** T **do**
    $\boldsymbol{\phi} \leftarrow \eta_t \sum_{k=1}^{|\boldsymbol{\ell}|} \nabla_{\boldsymbol{\phi}} \log p_\phi(\boldsymbol{\ell}_k \mid \boldsymbol{\phi})$
  end for
end for

E.1 Gibbs Sampler For Cluster Assignments

The Pitman–Yor process does not have a well-defined posterior probability. Nonetheless, we can use Gibbs sampling to obtain a sample from this posterior distribution over cluster assignments defined by the two-stage model.[9] We build our sampler after the morphological sampler presented by Goldwater et al. (2011).

Gibbs sampling is a Markov Chain Monte Carlo (MCMC) method which approximates the posterior of a multivariate distribution. It iteratively samples from the conditional distribution of a variable, given the values of the other dimensions (Neal, 1993). We use the conditional distribution defined in eq. (7) (presented in Fig. 2) in the Gibbs sampler where we know the word form $\boldsymbol{w}_n$ of token n—since it is observable in the corpus— and where the values for all other cluster assignments are fixed. Note that, according to eq. (7), we only assign word tokens to clusters with the same form or create a new cluster—and when a new one is created, its word form is assigned to $\boldsymbol{w}_n$. As such, each cluster contains a single shared word form. For each adaptor training iteration, we run the Gibbs sampler for six epochs, and choose the cluster assignments that have the best performance on a development set. Furthermore, we persist the adaptor state across iterations, warm starting the Gibbs sampler with the cluster assignments of the previous iteration.

E.2 Training the generator

In order to train the generator on word form data with more balanced frequency distributions, a new training set is dynamically created. In this dataset, each token appears as many times as it has been assigned as a cluster label, noted with $\boldsymbol{\ell}$ in Algorithm 1.[10] A regime similar to using the inverse-power transformed counts of the tokens in the corpus (Goldwater et al., 2011).

This new training set allows us to train the generator in an interpolation between a purely type- or token-based dataset; this interpolation can be controlled through its parameters a and b. Setting the values of a and b to zero will cause the model to favor existing clusters to creating new ones, resulting in assigning every token with the same form to a single cluster. In this case, the generator parameters would be estimated using the equivalent of a *type* corpus. Similarly, when a approaches one, or in the limit of $b \to \infty$, less tokens will be assigned per cluster and the number of single token clusters grows. This is effectively equivalent to training the generator using *tokens*. Consequently, non-extreme

[9] This is possible due to the exchangeability of the cluster assignments.

[10] We hotstart the generator model by training it on a type-level dataset before the first adaptor training iteration.

$$p(z_n \mid \mathbf{z}_{<n}, \boldsymbol{w}_n) \propto p(z_n, \boldsymbol{w}_n \mid \mathbf{z}_{<n}) \propto \begin{cases} (c_{<n}^{(z_n)} - a) \cdot \mathbb{1}\{\boldsymbol{w}_n = \boldsymbol{\ell}_{z_n}\} & 1 \leq z_n \leq K_{<n} \\ (a \cdot K_{<n} + b) \cdot p_\phi(\boldsymbol{w}_n) & z_n = K_{<n} + 1 \end{cases} \tag{7}$$

Figure 2: The probability of assigning token $\boldsymbol{w}_n$ to cluster z_n in the two-stage model given all other cluster assignments $\mathbf{z}_{<n}$.

value of a and b are a middle ground.

We train the character-level LSTM used as our generator with stochastic gradient descent using a cross-entropy loss function. This model is trained with early stopping; it is evaluated every 200 batches, and training stops when the development set loss has increased for 5 consecutive epochs.

E.3 Training Optimizations

The naïve implementation of the Gibbs sampler for table assignments quickly becomes computationally expensive in practice. Consequently, we use the optimized algorithm designed by Blunsom et al. (2009) for the hierarchical Dirichlet process in our implementation, extending it to Pitman–Yor processes with the additional parameter a.

F Dataset

As noted in the main text, we use Wikipedia data in our experiments. The amount of sentences used in our experiments is capped to one billion after shuffling them. Additionally, we define an upper bound to the amount of tokens used in each experiment. In case the training data exceed this limit, we construct a corpus by re-sampling (with replacement) the desired number of tokens using the corpus frequencies calculated from the original training corpus. The number of types and tokens used in training and evaluation are presented in Tab. 7. Noise in the Wikipedia data is somewhat reduced by hand-defining an alphabet for each language, and removing any sentence which includes words with invalid graphemes in it.[11]

	Train		Test	
	# Types	# Tokens	# Types	# Tokens
English	76,589	10^6	67,148	759,412
Finnish	208,498	10^6	108,020	332,220
Hebrew	131,288	10^6	105,550	619,685
Indonesian	102,739	10^6	72,250	507,848
Tamil	206,512	10^6	116,165	388,257
Turkish	154,185	10^6	85,074	331,072
Yoruba	97,097	329,093	12,117	41,055

Table 7: The amount of tokens and types used in both training and testing for the analyzed languages.

[11] We define the alphabets using the languages' Wikipedia articles and the following website: https://r12a.github.io/app-charuse/.