The Raymond and
Beverly Sackler Faculty
of Exact Sciences
Tel Aviv University

TEL-AVIV UNIVERSITY

RAYMOND AND BEVERLY SACKLER

FACULTY OF EXACT SCIENCES

THE BLAVATNIK SCHOOL OF COMPUTER SCIENCE

Models In a Spelling Bee: Language Models Implicitly Learn the Character Composition of Tokens

Thesis submitted in partial fulfillment of the requirements

for the M.Sc. degree of Tel-Aviv University

by

Itay Itzhak

The research work for this thesis has been carried out at

Tel-Aviv University under the supervision of

Dr. Omer Levy, Prof. Nachum Dershowitz and Dr. Kfir Bar

October 2021

Abstract

Standard pretrained language models operate on sequences of subword tokens without direct access to the characters that compose each token's string representation. We probe the embedding layer of pretrained language models and show that models learn the internal character composition of whole word and subword tokens to a surprising extent, without ever seeing the characters coupled with the tokens. Our results show that the embedding layer of RoBERTa holds enough information to accurately spell up to a third of the vocabulary and reach high average character *n*gram overlap on all token types. We further test whether enriching subword models with additional character information can improve language modeling, and observe that this method has a near-identical learning curve as training without spelling-based enrichment. Overall, our results suggest that language modeling objectives incentivize the model to implicitly learn some notion of spelling, and that explicitly teaching the model how to spell does not enhance its performance on such tasks.

Table of Contents

Acknowledgments

This work was supported by the Tel Aviv University Data Science Center, Len Blavatnik and the Blavatnik Family foundation, the Alon Scholarship, Intel Corporation, and the Yandex Initiative for Machine Learning. We thank Avia Efrat for his valuable feedback.

List of Figures

List of Tables

Abstract

Standard pretrained language models operate on sequences of subword tokens without direct access to the characters that compose each token's string representation. We probe the embedding layer of pretrained language models and show that models learn the internal character composition of whole word and subword tokens to a surprising extent, without ever seeing the characters coupled with the tokens. Our results show that the embedding layer of RoBERTa holds enough information to accurately spell up to a third of the vocabulary and reach high average character *n*gram overlap on all token types. We further test whether enriching subword models with additional character information can improve language modeling, and observe that this method has a near-identical learning curve as training without spelling-based enrichment. Overall, our results suggest that language modeling objectives incentivize the model to implicitly learn some notion of spelling, and that explicitly teaching the model how to spell does not enhance its performance on such tasks.

1 Introduction

Contemporary subword tokenization algorithms such as BPE [11] partition a string into contiguous spans of characters. Each span represents a frequent character *n*gram, from individual characters (*a*), through prefixes (*uni*) and suffixes (*tion*), and even complete words (*cats*). The tokenizer then converts each such span into a discrete symbol (a token) with no internal structure, effectively discarding the token's orthographic information. Therefore, a model operating over sequences of subword tokens should be oblivious to the spelling of each token. In this work, we show that despite having no direct access to the subwords' internal character composition, pretrained language models *do* learn some notion of spelling.

To examine what pretrained language models learn about spelling, we present the *SpellingBee* probe. SpellingBee is a generative language model that predicts the character composition of a token given only its (uncontextualized) vector representation from the pretrained model's embeddings matrix. SpellingBee is trained on part of the model's vocabulary, and then tested by spelling unseen token types. If the probe can successfully reconstruct the correct character sequence from an unseen token's embedding, then there must be significant orthographic information encoded in the vector.

We find that the embedding layers of several pretrained language models contain surprising amounts of character information. SpellingBee accurately spells 31.8% of the held-out vocabulary for RoBERTa-Large [7], 32.9% for GPT2-Medium [10], and 40.9% for the Arabic language model AraBERT-Large [1]. A softer metric that is sensitive to partially-correct spellings (chrF) [9] shows a similar trend, with 48.7 for RoBERTa-Large and 62.3 for AraBERT-Large. These results are much higher than the baseline of applying SpellingBee to randomly-initialized vectors, which fails to spell a single token.

Given that subword models learn some notion of character composition to fulfill language modeling objectives, could they perhaps benefit from knowing the exact spelling of each token a priori? To that end, we reverse SpellingBee's role and use it to pretrain the embedding layer of a randomly-initialized model, thus imbuing each token representation with its orthographic information before training the whole model on the masked language modeling objective. We compare the pretraining process of the character-infused model to that of an identical model whose embedding layer is randomly initialized (and not pretrained), and find that both learning curves converge to virtually identical values within the first 1000 gradient updates, a fraction of the total optimization process. This experiment suggests that while language models may need to learn some notion of spelling to optimize their objectives, they can quickly acquire all the character-level information they need without directly observing the composition of each token.

2 Spelling Bee

To measure how much a model knows the character composition of its tokens, we introduce SpellingBee, a generative probe that tries to spell out a token character-by-character. Specifically, SpellingBee probes the original model's *embedding matrix*, since spelling is a property of token *types*, invariant to context. For example, given the embedding of the token *cats*, SpellingBee will try to generate the sequence [*c*, *a*, *t*, *s*]. We do so by modeling SpellingBee as a character-based language model, where the first token is a vector representation of the vocabulary item.[1]

Training We split the vocabulary to train and test sets,[2] and use teacher forcing to train SpellingBee. In the example of *cats*, SpellingBee will compute the following probabilities:

$$
\begin{gathered}
P(x_1 = c \mid x_0 = cats) \\
P(x_2 = a \mid x_0 = cats, x_1 = c) \\
P(x_3 = t \mid x_0 = cats, x_1 = c, x_2 = a) \\
\vdots
\end{gathered}
$$

All of SpellingBee's parameters are randomly initialized. The only parameters that are pretrained are the token embeddings (e.g. the representation of *cats* or *a*), which are taken from the original pretrained language model we intend to probe, and treated as constants; i.e. kept frozen during SpellingBee's training.

Inference & Evaluation Once SpellingBee is trained, we apply it to the test set using greedy decoding. For each vocabulary item w in the test set, SpellingBee is given only the corresponding embedding vector e_w, and is expected to generate the character sequence $w_1, \ldots, w_n$ that defines w. We measure success on the test set using two metrics: *exact match* (EM), and character *n*gram overlap score using *chrF* [9]. While EM is strict, chrF allows us to measure partial success. We also report edit distance using Levenshtein distance ratio in Appendix A.

[1] Some vocabularies have symbols for indicating preceding whitespaces (_) or that the next token is part of the same word (##). SpellingBee learns to predict these symbols too.

[2] We test various train/test splits to ensure the robustness of our findings. See Section 3 for more detail.

SpellingBee for Pretraining Embeddings While we mainly use SpellingBee as a probe, a variation of our method could potentially imbue the embedding layer with character information before training a language model. We could train a probe with randomly-initialized embeddings (instead of pretrained embeddings from another model) to predict the spelling of *all* vocabulary items, and use these trained probe embeddings to initialize any target model's embedding layer (instead of random initialization). We experiment with this method in Section 6, but find that it does not have any significant impact on the convergence of language models.

3 Experiments

We begin with a series of probing experiments, where we apply SpellingBee to the embedding layer of various pretrained models.[1]

Pretrained Models We probe four pretrained models: RoBERTa-Base and Large [7], GPT2-Medium [10], and AraBERT-Large [1]. This set introduces some diversity in vocabulary, objective, and scale: the first three models are trained on English corpora, while AraBERT is trained on text in Arabic; GPT2 is an autoregressive language model, while the rest are masked language models; RoBERTa-Base consists of 125M parameters (with 768 dimensions per embedding), while the other models have approximately 350M parameters (with 1024 dimensions per embedding).

Control Since SpellingBee is a trained probe, one might claim that its performance may partially stem from knowledge acquired during the probe's training. To address this alternative explanation, we wish to establish the probe's baseline performance when provided with inputs with no orthographic information. As an empirical *control*, we train and test SpellingBee on randomly-initialized vectors, in addition to the main experiments where we utilize the pretrained embedding layers.

Training & Testing Data We split the vocabulary into training and testing data using the following protocol. First, we randomly sample 1000 token types as test. We then filter the remaining vocabulary to eliminate tokens that may be too similar to the test tokens, and randomly sample 32000 training examples.We experiment with three filters: *none*, which do not remove tokens beyond the test-set tokens; *similarity*, which removes the top 20 most similar tokens for every token in test, according to the cosine similarity induced by the embedding vectors; *lemma*, which removes any token type that shares a lemma with a test-set token (e.g. if *diving* is in the test set, then *diver* cannot be in the training set).[2] The lemma filter always applies the similarity filter first, providing an even more adversarial approach for splitting the data. To control for variance, we

[1] Hyperparameters are detailed in Appendix B.

[2] We lemmatize using NLTK's WordNet lemmatizer [2] for English and Farasa's Stemmer [3] for Arabic.

create 10 such splits for each model and filter, and report the averaged evaluation metrics over all 10 test sets.

4 Results

4.1 Results

Main Result Table 4.1 shows how well SpellingBee can spell a vocabulary token using only its frozen pretrained embedding. We observe that SpellingBee is able to accurately recover the spelling of up to 40.9% of the test set, while the control is unable to spell even a single word correctly. A similar trend can be seen when considering the finer character *n*gram metric (chrF). Manually analyzing the predictions of the control baselines (see Appendix C) indicate that it primarily generates combinations of frequent character sequences, which mildly contributes to the chrF score, but does not affect EM. These results are persistent across different models and filters, strongly indicating that the embedding layer of pretrained models contain significant amounts of information about each token's character composition.

One may suggest that training SpellingBee over 32000 examples may leak information from the test set. For example, if *dog* was seen during training, then spelling out *dogs* might be easy. We thus consider the similarity and lemma filters, which remove such near-neighbors from the training set. While results are indeed lower (and probably do account for some level of information leakage), they are still considerably higher than the control, both in terms of EM and chrF. Results using the similarity and lemma filters are rather similar, suggesting that embedding-space similarity captures some information about each token's lemma.

Finally, we find that the properties of pretrained models also seem to have a significant effect on the amount of spelling information SpellingBee can extract. Larger models tend to score higher in the probe, and the model trained on text in Arabic appears to have substantially higher EM and chrF scores than those trained on English corpora. One possibility is that Arabic's rich morphology incentivizes the model to store more information about each token's character composition, however it is also possible that AraBERT's different vocabulary, which allocates shorter character sequences to each token type, might explain this difference (we discuss the link between sequence length and accuracy later in this section).

Overall, our probing experiments show that even though subword-based language models do not have direct access to spelling, they *can* and *do* learn a surprising amount of information about the character composition of each vocabulary token.

	Filter	RoBERTa Base	RoBERTa Large	GPT2 Medium	AraBERT Large
EM	None	27.3	31.8	32.9	40.9
	Similarity	15.7	18.2	17.9	21.9
	Lemma	15.7	17.7	16.5	19.5
	Control	0.0	0.0	0.0	0.0
chrF	None	44.7	48.7	51.6	62.3
	Similarity	32.7	35.1	36.4	46.0
	Lemma	32.6	34.8	35.2	43.9
	Control	7.0	7.0	7.0	7.0

Table 4.1: The percent of token types that can be spelled out exactly (EM) from their embeddings by SpellingBee, and the *n*gram overlap between SpellingBee's reproductions and the token types' true spellings (chrF). The first three rows reflect different methods for filtering the training data, and the fourth represents the control experiment, which uses randomly initialized embeddings. All SpellingBee instances in this table are trained on 32000 examples.

Probing with Less Training Data We further examine whether SpellingBee can extract information when trained on less examples. Figure 4.1 shows how well SpellingBee can spell RoBERTa-Large's vocabulary when trained on varying amounts of data, across all filters. We find that more data makes for a better probe, but that even a few thousand examples are enough to train SpellingBee to extract significant character information from the embeddings, which *cannot* be extracted from randomized vectors (the control).[1].

[1] We provide additional analysis on spelling accuracy by subword frequency and length in Section 5

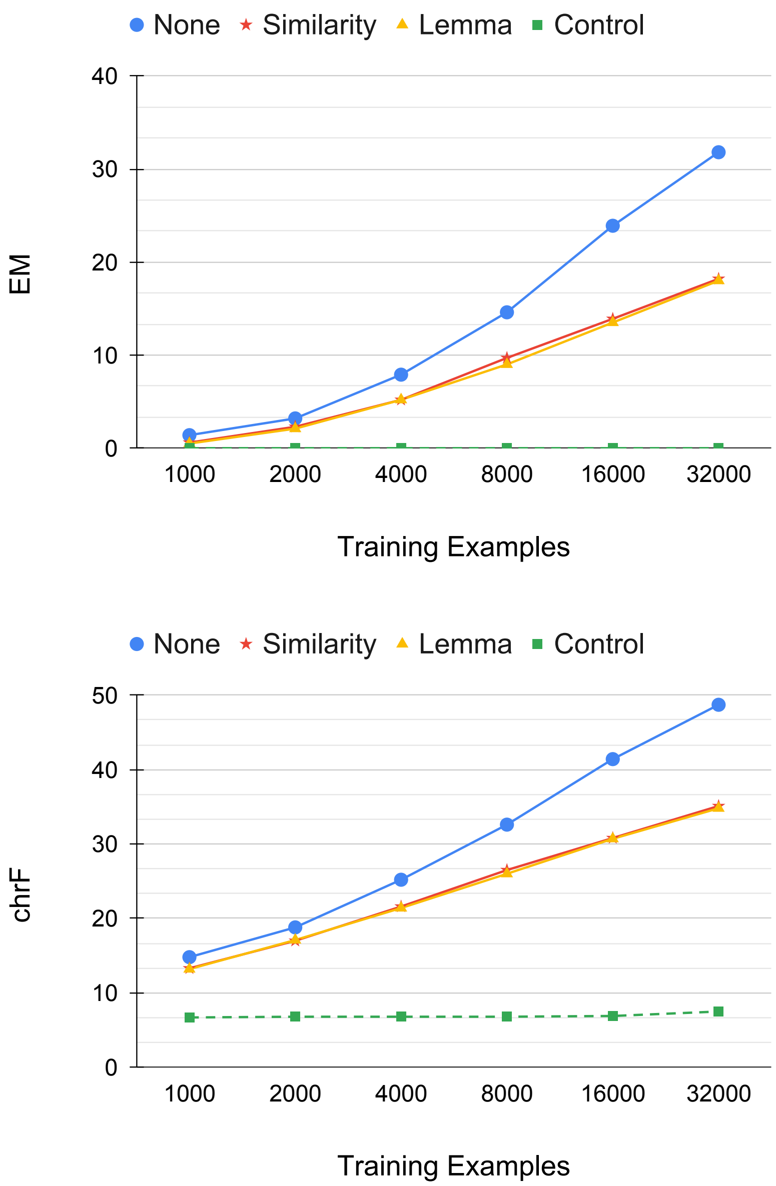

Figure 4.1: The amount of character information SpellingBee is able to extract from RoBERTa-Large, as measured by EM (top) and chrF (bottom), given different quantities of training examples.

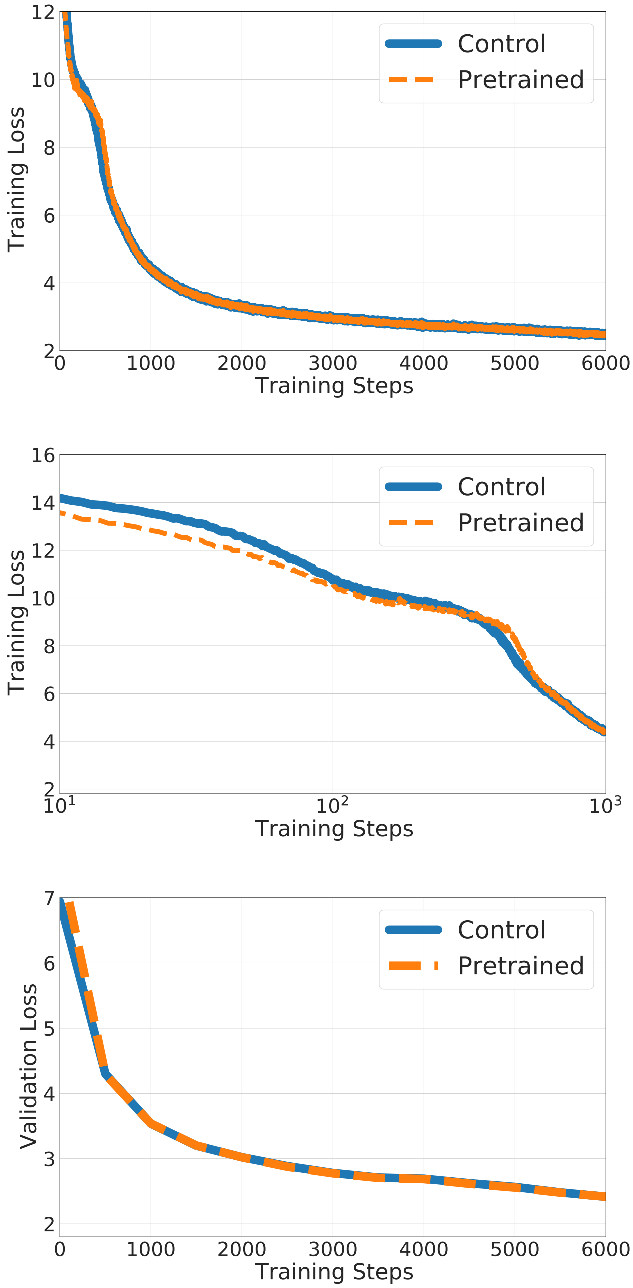

Figure 4.2: The overall training loss (left), first steps of training loss (center), and validation loss (right) of RoBERTa-Large, when training on the masked language modeling objective with embeddings pretrained by SpellingBee (*pretrained*) and randomly-initialized embeddings (*control*).

5 Analysis

5.1 Spelling Accuracy by Frequency

We test whether pretrained models tend to store more spelling-related information in higher-frequency token types. We focus on RoBERTa-Large, and assign each token in the test set to its frequency quintile according to the number of times it appeared in the pretraining corpus – from the 10000 most frequent token types (top 20%) to those ranked 40000-50000 in the vocabulary (bottom 20%) – and measure the average performance of SpellingBee within each quintile. Figures 5.1 and 5.2 shows the results with and without the similarity filter. We observe that SpellingBee is indeed able to extract more information from higher-frequency token types, suggesting that the pretrained model has more information about their character composition.

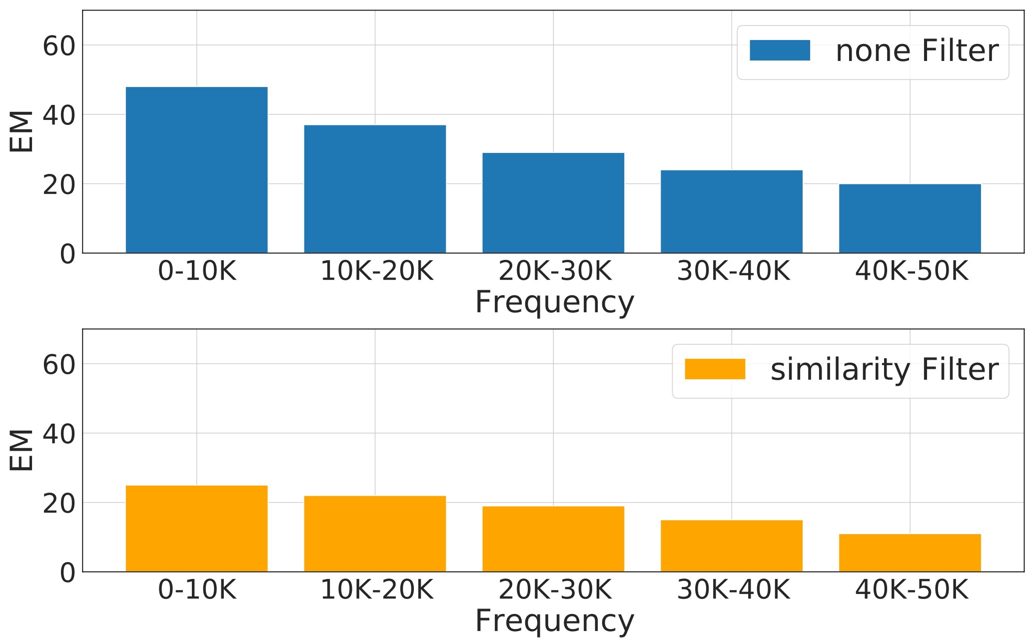

Figure 5.1: The EM scores of SpellingBee on RoBERTa-Large for each frequency quintile with the *none* filter (top) and the *similarity* filter (bottom).

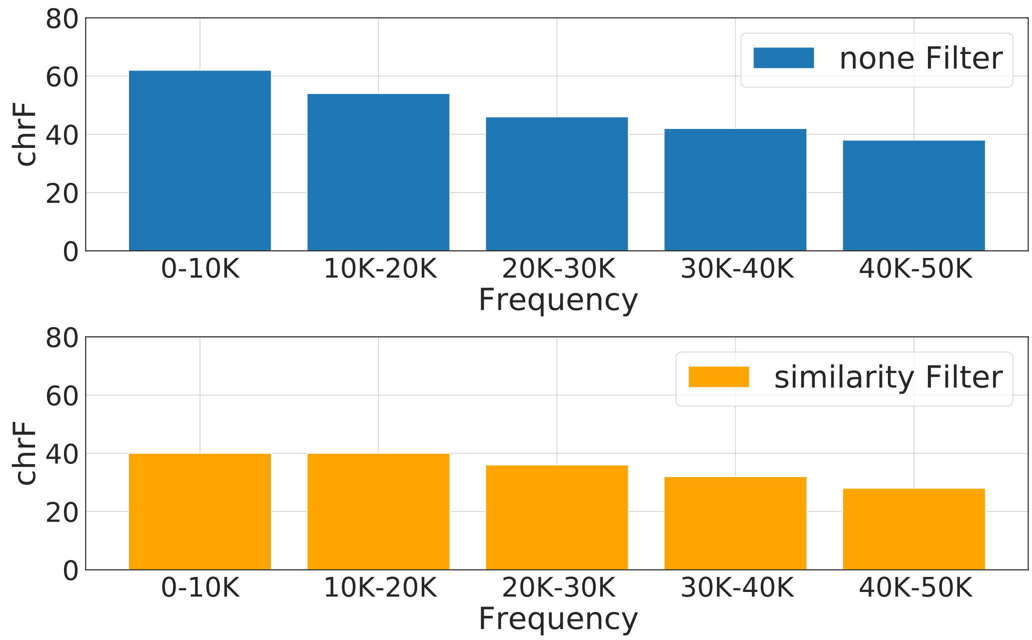

Figure 5.2: The chrF scores of SpellingBee on RoBERTa-Large for each frequency quintile with the *none* filter (top) and the *similarity* filter (bottom).

5.2 Spelling Accuracy by Length

We analyze the effect of token length on the probe's ability to spell. A priori, it is reasonable to assume that it is easier for the probe to spell shorter tokens, since less information needs to be extracted from the embedding and there are less discrete decisions to be made while decoding. Indeed, Figure 5.3 shows that with the none filter most vocabulary tokens with 2-4 characters can be accurately reproduced from their vector representations, while longer tokens are harder to replicate. This trend is particularly sharp when the similarity filter is applied, as the probe is hardly able to spell tokens with 6 or more characters accurately; having said that, the probe is able to generate many *partially correct* spellings, as measured by chrF (Figure 5.4). Perhaps a less intuitive result is the probe's failure to spell single-character tokens. A closer look reveals that many of these examples are rare or non-alphanumeric characters (e.g. *ç* and *$*), which are probably very difficult for the probe to generate if it had not seen them during training. While these results show strong trends with respect to length, token length is also highly correlated with frequency, and it is not necessarily clear which of the two factors has a stronger impact on the amount and resolution of character-level information stored in the embedding layer of pretrained models.

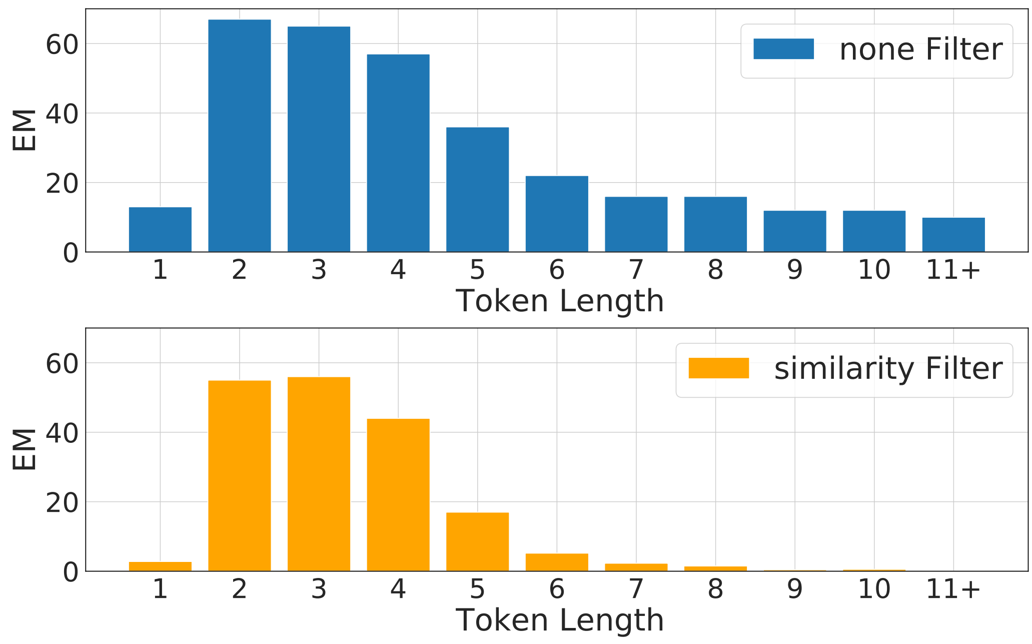

Figure 5.3: The EM scores of SpellingBee on RoBERTa-Large for each token length with the *none* filter (top) and the *similarity* filter (bottom). The rightmost column groups together tokens with length of 11 or above.

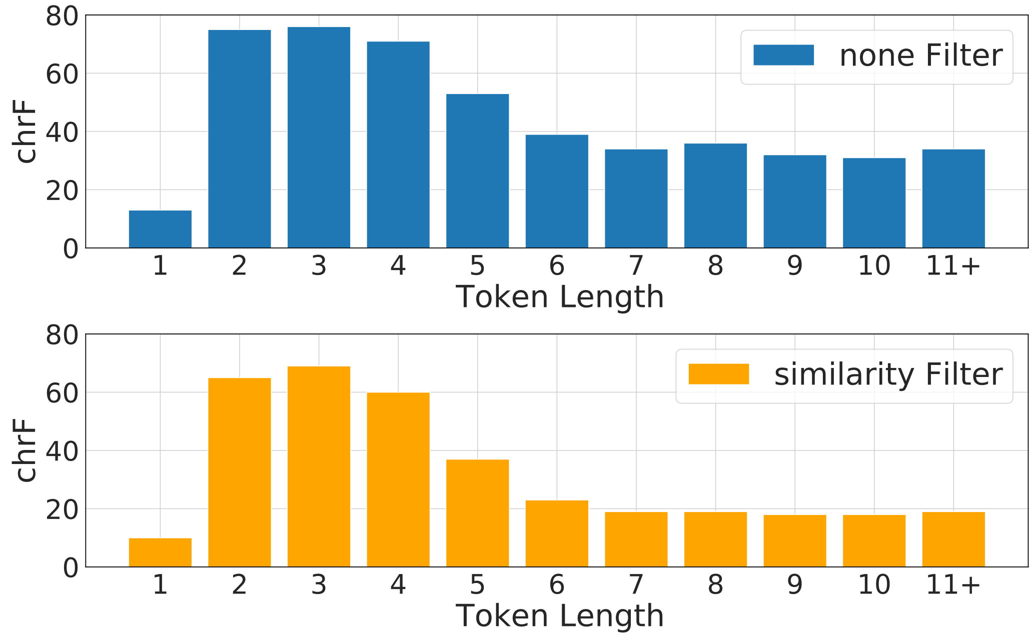

Figure 5.4: The chrF scores of SpellingBee on RoBERTa-Large for each token length with the *none* filter (top) and the *similarity* filter (bottom). The rightmost column groups together tokens with length of 11 or above.

6 Pretraining Language Models to Spell

Our probing experiments reveal that language models learn some partial notion of spelling, despite the lack of direct access to characters. Therefore, we hypothesize that learning to spell is beneficial for language models, and propose pretraining the embedding layer using a variant of the SpellingBee probe described in Section 2. Here, the goal is to imbue each embedding with enough information for SpellingBee to accurately generate its surface form, and then initialize the language model with the pretrained embeddings before it starts training on the language modeling objective.

We apply this process to RoBERTa-Large, training the model's embedding layer with SpellingBee using the same hyperparameter settings from Appendix B, with the key difference being that the embeddings are now tunable parameters (not frozen).[1] We train RoBERTa-Large on English Wikipedia using the hyperparameter configuration of 24hBERT [4], and cease training after 24 hours ($\sim$16000 steps). For comparison, we train exactly the same model with a randomly-initialized embedding layer.

Figure 4.2 shows the masked language modeling loss with and without pretrained embeddings. We see that the curves quickly converge into one. After only 1000 training steps, the difference between the validation losses never exceeds 0.01. This result indicates that the model does not utilize the character information injected into the tokens' embeddings. Along with the results from Section 4.1, we conjecture that the model learns an implicit notion of spelling during pretraining, which is sufficient for masked language modeling, and does not benefit from explicitly adding orthographic information.

[1] To verify that this process does indeed encode the tokens' spellings into the embeddings, we apply a SpellingBee *probe* (using a different random initialization) to the learned embeddings, which yields 93.5% EM on held-out token types.

7 Conclusion

This work reveals that pretrained language models learn, to some extent, the character composition of subword tokens. We show that our SpellingBee probe can spell many vocabulary items using their uncontextualized embedding-layer representations alone. Trying to explicitly infuse character information into the model appears to have a minimal effect on the model's ability to optimize its language modeling objective, suggesting that the model can independently learn all the character-level information it needs for the task.

References

[1] Wissam Antoun, Fady Baly, and Hazem Hajj. AraBERT: Transformer-based model for Arabic language understanding. In *Proceedings of the 4th Workshop on Open-Source Arabic Corpora and Processing Tools, with a Shared Task on Offensive Language Detection*, pages 9–15, Marseille, France, May 2020. European Language Resource Association. ISBN 979-10-95546-51-1. URL `https://www.aclweb.org/anthology/2020.osact-1.2`.

[2] Steven Bird and Edward Loper. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL `https://aclanthology.org/P04-3031`.

[3] Kareem Darwish and Hamdy Mubarak. Farasa: A new fast and accurate Arabic word segmenter. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1070–1074, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL `https://aclanthology.org/L16-1170`.

[4] Peter Izsak, Moshe Berchansky, and Omer Levy. How to train bert with an academic budget. *arXiv preprint arXiv:2104.07705*, 2021.

[5] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[6] Vladimir I Levenshtein et al. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[7] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[8] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling.

In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-4009. URL `https://www.aclweb.org/anthology/N19-4009`.

[9] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the Tenth Workshop on Statistical Machine Translation*, pages 392–395, Lisbon, Portugal, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-3049. URL `https://aclanthology.org/W15-3049`.

[10] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[11] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL `https://www.aclweb.org/anthology/P16-1162`.

A Levenshtein Distance

Levenshtein distance [6] is an edit distance metric that, given two strings, calculates the minimal number of changes needed to be done in order to make the two strings identical. Levenshtein distance *ratio* is the length-normalized version, which is computed by adding the sum of lengths of both strings to the edit distance and dividing by the same sum of lengths. We report the main experiment's results using this ratio in Table A.1.

Filter	**RoBERTa**		**GPT2**	**AraBERT**
	Base	**Large**	**Medium**	**Large**
None	69.7	72.7	74.4	83.6
Similarity	61.5	63.7	64.5	75.8
Lemma	61.4	63.3	63.7	74.8
Control	25.6	26.4	27.0	25.7

Table A.1: Levenshtein distance ratio. The first three rows reflect different methods for filtering the training data, and the fourth represents the control experiment, which uses randomly initialized embeddings. All SpellingBee instances in this table are trained on 32000 examples.

B SpellingBee – Hyperparameters

We implement SpellingBee with a 6-layer encoder-decoder model, with 512 model dimensions. The model parameters are optimized with Adam [5] for 1000 steps with up to 1024 tokens per batch, a learning rate of 5e-4, and a dropout rate of 0.1. These are the default hyperparameters for training a transformer language model in Fairseq [8].

C Manual Error Analysis

We manually analyze 100 random tokens that SpellingBee spelled incorrectly with the lemma filter to understand the nature of the spelling mistakes. Out of those 100 we display 20 mistakes in Table C.1 alongside the spelling prediction of the control baseline. SpellingBee's mistakes vary from single-character typos to completely different words. Having said that, the vast majority of mistakes have significant overlap with the correct spelling, such as shared prefixes and capitalization.

Token	**SpellingBee**	**Control**
_Issa	_Asey	_kinston
_Rhod	_Rob	_hoedn
Memory	Mathinge	_entically
_metals	_metrys	_leaved
_Reed	_Redd	_fomparing
_break	_breach	_promoters
_summit	_mosump	_seasons
Catholic	Cravital	_tonversal
_cleanup	_lamed	_paclus
_Winner	_Womer	_purden
_LIM	_LUM	_Send
Copy	Cople	_providers
_voicing	_relicing	_walking
_Stab	_Stamb	_hoviders
_356	_353	_budiance
find	wive	_malding
_Psychic	_Syptanc	_joacter
_Looking	_Lowing	parging
CLOSE	DEFIC	_tuldence
_prolific	_promistic	_complexement

Table C.1: Sampled SpellingBee errors with the lemma filter alongside the control baseline's spelling for the same tokens. The underscore (_) represents a preceding whitespace.

תקציר

טוקניזציה המבוססת על תתי-מילים (כגון BPE) הפכה לנוהג מקובל ב- pretraining של מודלי שפה עבור מגוון של שפות ומשימות. אולם, עבודות שפורסמו לאחרונה על parsing עבור שפות עשירות מורפולוגית (MRLs) טוענות כי שיטות טוקניזציה בדוגמת BPE אינן הולמות שפות אלה, ומציעות ייצוגי קלט המכילים מידע מורפולוגי. בעבודה זו, אנו משערים כי טוקניזציה המבוססת על תווים, ללא כל עיבוד מקדים, מספקת עבור המודלים את כל המידע שהם זקוקים לו על מנת ללמוד מאפיינים מורפולוגיים. כדי לבחון השערה זו, אנו מאמנים masked language model בסגנון BERT, הפועל על רצפי תווים - TavBERT. ניסויים על תיוג חלקי-דיבר וניתוח מורפולוגי בשלוש שפות: עברית, טורקית וערבית, מראים שגישתנו עדיפה על פני שיטות עם מידע מורפולוגי. בנוסף, אנו משווים את המודל שאימנו עם מודלים מתחרים דומים שאומנו עם טוקניזציית תתי-מילים, ומוצאים כי TavBERT משיג ביצועים טובים במעט עבור תיוג חלקי-דיבר, אך משיג יתרון משמעותי עבור ניתוח מורפולוגי. תוצאות אלה מעידות כי מודלי שפה בקנה מידה עצום מסוגלים ללמוד דפוסים מורפולוגיים מורכבים כאשר ניתנת להם גישה לתווים גולמיים.

הפקולטה למדעים
מדויקים ע"ש ריימונד
ובברלי סאקלר
אוניברסיטת תל אביב

אוניברסיטת תל אביב

הפקולטה למדעים מדוייקים ע"ש ריימונד ובברלי סאקלר

בית הספר למדעי המחשב ע"ש בלווטניק

TavBERT –
למידת מורפולוגיה עשירה באמצעות מיסוך רצפי תווים

חיבור זה הוגש בעבודת מחקר לקראת התואר "מוסמך אוניברסיטה" במדעי המחשב
על ידי:

עמרי קרן

העבודה נעשתה בבית הספר למדעי המחשב
בהנחיית ד"ר עמר לוי

תשרי תשפ"ב