Improving Low Compute Language Modeling with In-Domain Embedding Initialisation

Charles Welch, Rada Mihalcea and **Jonathan K. Kummerfeld**
Computer Science & Engineering
University of Michigan
{cfwelch,mihalcea,jkummerf}@umich.edu

Abstract

Many NLP applications, such as biomedical data and technical support, have 10-100 million tokens of in-domain data and limited computational resources for learning from it. How should we train a language model in this scenario? Most language modeling research considers either a small dataset with a closed vocabulary (like the standard 1 million token Penn Treebank), or the whole web with byte-pair encoding. We show that for our target setting in English, initialising and freezing input embeddings using in-domain data can improve language model performance by providing a useful representation of rare words, and this pattern holds across several different domains. In the process, we show that the standard convention of tying input and output embeddings does not improve perplexity when initializing with embeddings trained on in-domain data.

1 Introduction

Language modeling is an essential part of many NLP applications, including predictive keyboards, speech recognition, and translation. Recent work has focused on (1) small constrained datasets, such as the Penn Treebank (Marcus et al., 1993) and WikiText-103 (Merity et al., 2017b), and (2) vast resources with billions of words used to train enormous models with significant computational requirements (Radford et al., 2019). This leaves a gap: when a substantial amount of in-domain data is available, but computational power is limited.

We explore how initialising word embeddings using in-domain data can improve language modeling in English. Testing all valid configurations of weight tying, embedding freezing, and initialisation, we find that the standard configuration is not optimal when rare words are present. Instead, the best approach is to initialise with in-domain data, untie the input and output, and freeze the input.

To understand this difference, we run a series of experiments to measure the impact of changing (a) the threshold for replacing rare words with a special symbol; (b) the source of data for initialisation; (c) the amount of training data for the language model; and (d) the hyperparameters for both the baseline and our proposed approach. We find that the improvement comes from improved representation of rare words. These findings are confirmed through experiments on four additional domains, with similar trends.

We also compare our approach to an n-gram language model and a large-scale transformer model. We find that if a large-scale transformer is inappropriate either for computational or modeling reasons, it is best to train an LSTM-based language model with as much data as possible and initialise the embeddings on all available in-domain data.

2 Proposed Approach

We propose initialising the language model's word embeddings with vectors trained on additional in-domain data. To make this most effective, we make two other key changes to training. First, we prevent embeddings from shifting during training. Without this, the embedding space could become inconsistent as vectors for words seen in training shift while those for words seen only in the additional data stay the same. Second, we do not tie the weights of the input embeddings and final output layer. To understand the impact of these factors, we train models with every valid combination of weight tying, freezing, and pretraining.[1]

We experiment with Merity et al. (2017a)'s AWD-LSTM – a high-performing model that can be trained in under a day on a single GPU (without fine-tuning). We train embeddings using GloVe on Gigaword.[2] For evaluation, we consider two

---

[1]Note, for frozen output embeddings the bias is not frozen.
[2]Embedding size 400 and rare word cutoff 5, the same as in the original AWD-LSTM model and GloVe respectively. All other GloVe hyperparameters were set as specified in the original GloVe paper and trained using the released code.

versions of the Penn Treebank. *Std* is the standard version used in language modeling, with words of frequency less than five converted to UNK, all words lowercase, numbers replaced with a special symbol, and punctuation removed. *Rare* has the same pre-processing but without replacement of rare words.[3]

Table 1 shows the results, with icons to concisely describe the different configurations.[4] Looking first at the standard evaluation set, we can see the value of pretrained embeddings by considering pairs where the only difference is whether the embeddings are random or pretrained. Pretrained embeddings are better in all but one case (comparing the fourth last and second last rows), and there the difference is only 0.5. As for freezing the pretrained input embeddings, keeping all other aspects the same, it is always better to freeze them.

There are also four clear sections of performance in the table: (a) frozen random output embeddings; (b) frozen pretrained output embeddings; (c) frozen random input embeddings; (d) various configurations. These results have an asymmetry. Freezing the output embeddings consistently leads to poor performance, even with pretrained embeddings pretrained. In contrast, freezing with pretrained input embeddings leads to some of the best results. We expected freezing with random initialisation to perform poorly, but the drop is modest for input freezing and dramatic for output freezing. This suggests that the two embedding matrices are serving different purposes in the model. The results do support the practise of tying when the input embeddings are random, but the benefit is half as large when they are pretrained.

For the dataset with rare words we see mostly the same trends. The exception is the bottom six rows. Once rare words are present, random initialisation of the input embeddings is considerably worse than pretraining (third last row). Again, there is an asymmetry between input and output, with the top five models all using pretrained input embeddings, but only three of them using pretrained output embeddings. Tying is also no longer the best approach, with the top three models not tying. Our proposed approach, using pretrained untied embeddings and freezing the input, has the best results.

The only difference between Std and Rare is

	Tied	Input (freeze)	Input (init)	Output (freeze)	Output (init)	Std	Rare
(a)	Tied	Frozen	Random	Frozen	Random	680	1120
	Untied	Frozen	Random	Frozen	Random	680	1120
	Untied	Unfrozen	Random	Frozen	Random	680	431
	Untied	Unfrozen	Pretrained	Frozen	Random	220	372
	Untied	Frozen	Pretrained	Frozen	Random	218	360
(b)	Untied	Frozen	Random	Frozen	Pretrained	121	202
	Untied	Unfrozen	Random	Frozen	Pretrained	95.0	170
	Untied	Unfrozen	Pretrained	Frozen	Pretrained	91.3	147
	Tied	Frozen	Pretrained	Frozen	Pretrained	90.7	136
	Untied	Frozen	Pretrained	Frozen	Pretrained	90.7	136
(c)	Untied	Frozen	Random	Unfrozen	Random	82.2	143
	Untied	Frozen	Random	Unfrozen	Pretrained	81.4	142
(d)	Untied	Unfrozen	Random	Unfrozen	Random	65.3	120
	Untied	Unfrozen	Random	Unfrozen	Pretrained	64.1	113
	Untied	Unfrozen	Pretrained	Unfrozen	Random	62.5	105
	Untied	Unfrozen	Pretrained	Unfrozen	Pretrained	61.7	98.5
	Untied	Frozen	Pretrained	Unfrozen	Pretrained	61.6	**97.1**
	Tied	Unfrozen	Random	Unfrozen	Pretrained	61.3	112
	Untied	Frozen	Pretrained	Unfrozen	Random	61.1	98.1
	Tied	Unfrozen	Pretrained	Unfrozen	Pretrained	**59.8**	98.7

■ = Tied parameters   ❏ = Untied parameters
❄ = Frozen in training   🔥 = Unfrozen in training
⊡ = Random init.   ⊠ = Pretrained init.

Table 1: Perplexity on the PTB for all valid combinations of weight tying, freezing, and pretraining. Results are sorted by perplexity on Std and shown to three significant figures.

the lack of UNKs in Rare. This impacts 5.1% of tokens in the validation set (33% of types). While our pretrained embeddings do not cover all of these rare words, they do cover most. The vocabulary from Gigaword that we build vectors for covers 99.5% of the validation word tokens in Std (98% of word types), and 98.8% of the validation word tokens in Rare (84% of word types).

3 When & Why Does Pretraining Help?

To understand the strengths and limitations of this new approach, we consider a series of experiments, each probing a specific variable. To simulate our target scenario, we use 44 million words of Wall Street Journal data from the North American

---

[3]The script to generate our *Rare* data from the LDC release is available at: http://jkk.name/emnlp20lm/.

[4]Dice Icon by Andrew Doane from the Noun Project. Fire and Snowflake Icons by Freepik from www.flaticon.com.

News Corpus (NANC, Graff, 1995). This provides enough data for pretraining, training, validation, and test sets all in the exact same domain (not even varying the newspaper). We apply similar preprocessing as in the previous section, but break the data down into articles rather than sentences and keep rare words.

We compare the six best configurations from Table 1. In all cases, output embeddings are not frozen, so we leave out the symbol. We use only one symbol for pretraining/random because both embeddings are the same in most cases. The exceptions have ⅌ to indicate pretrained input and random output.

- ■ 🔥 🎲 Standard approach.
- ▣ 🔥 ⅌ Our approach, but with random output embeddings and without freezing.
- ■ 🔥 🌐 Standard approach + pretraining.
- ▣ 🔥 🌐 Our approach, but without freezing.
- ▣ ❄ 🌐 Our approach.
- ▣ ❄ ⅌ Our approach, but with random output embeddings.

Other Domains Show the Same Pattern. First we consider varying the domain to make sure this is not an artifact of news data. Table 2 shows results on Covid-19 research (Wang et al., 2020), Ubuntu IRC chat (Kummerfeld et al., 2019), Reddit, and Wikipedia, tokenised with either Scispacy (Neumann et al., 2019) or Stanza (Qi et al., 2020). Pretraining consistently helps, while freezing is best on all but Wikipedia. Our approach is consistently either the best or very close to the best.

The Improvement is Due to Rare Words. To probe the impact of rare words, we explore replacing them with UNK (using the same UNK symbol as used in embedding pretraining). We consider four variations, each constructed in two steps. First, we make a list of the words in the original training set and how many times each one occurs. Second, we make modified versions of the training and validation sets, replacing words with UNK if their count in our list is lower than K. For this step, any word that does not appear in our list is treated as if it has a count of zero. We consider K = 0, 1, 2 and 5. K is 0 for all other experiments in this section, which means that no words are replaced with UNK. When K is 1, 2, and 5, the introduction of UNKs means all words in the validation set are seen during language model training.

Train Config	Domain				
	NANC	Cord	IRC	Reddit	Wiki
■ 🔥 🎲	106	135	41.3	186	206
▣ 🔥 ⅌	103	125	41.1	166	174
■ 🔥 🌐	97.2	121	39.8	154	142
▣ 🔥 🌐	95.7	111	39.2	152	**141**
▣ ❄ 🌐	90.8	**109**	**37.3**	**146**	144
▣ ❄ ⅌	**90.5**	112	37.6	152	161

Table 2: Results for various domains. All other results in this section are for NANC.

Train Config	Frequency Cutoff			
	0	1	2	5
■ 🔥 🎲	106	106	70.6	55.4
▣ 🔥 ⅌	103	104	72.5	56.8
■ 🔥 🌐	97.2	99.9	68.1	54.1
▣ 🔥 🌐	95.7	97.8	70.2	56.0
▣ ❄ 🌐	90.8	92.1	66.5	54.5
▣ ❄ ⅌	**90.5**	**91.5**	**65.8**	**54.0**
UNK Types Dev	0%	13%	21%	33%
UNK Tokens Dev	0%	2.3%	3.4%	5.5%
UNK Types Train	0%	0%	40%	68%
UNK Tokens Train	0%	0%	1.4%	4.1%

Table 3: Varying the minimum frequency to not be converted into an UNK. The top half shows language model perplexity. The bottom half shows the percentage of word tokens and types that are replaced with UNK in each case.

Dataset	Train		Pretrain		Train in Pre	
	Type	Tok	Type	Tok	Type	Tok
PTB	73	5.3	77	0.11	14	1.3
NANC	71	4.8	63	0.49	13	0.63
Sci	78	6.3	85	1.2	23	1.6
IRC	83	4.2	90	1.3	37	1.4
Reddit	81	6.1	86	0.69	15	0.71
Wiki	78	7.3	78	0.36	5.6	0.43

Table 4: Percentage of word types and tokens that occur five times or fewer in each dataset. The last two columns are the percentage of types/tokens in the training set that occur five or fewer times in the pretraining set. For PTB the pretraining set is Gigaword (as used in Table 1).

Table 3 shows a clear trend: the benefit of our approach grows as more rare words are present (i.e., K is smaller). Note, it may seem odd that perplexity is higher when K=1 than when K=0 since we have removed rare words. This is probably because when K is 1 there are UNKs in the validation set

	Pretraining Source			
Train Config	NANC 43M	Gigaword 5B	GloVe 6B	GloVe 42B
■ 🔥 🚫	97.2	90.9	93.1	93.3
📕 🔥 ⁒	103	99.3	98.3	99.5
📕 🔥 🎲	95.7	**90.0**	91.2	93.6
📕 ❄ 🚫	90.8	90.6	91.1	91.5
📕 ❄ ⁒	90.5	90.7	90.7	91.9

Table 5: Varying similarity and size of pretraining data. Dataset size is shown below the name of each dataset.

	NANC WSJ			Gigaword		
Train Config	1x	2x	4x	1x	2x	4x
■ 🔥 🎲	106	81.0	67.5	106	92.5	86.3
📕 🔥 ⁒	103	83.3	68.7	99.3	91.7	87.2
■ 🔥 🚫	97.2	80.4	67.8	90.9	88.6	85.7
📕 🔥 🚫	95.7	80.0	68.1	**90.0**	86.4	85.5
📕 ❄ 🚫	90.8	73.7	66.8	90.6	84.8	**82.5**
📕 ❄ ⁒	**90.5**	**72.9**	**66.1**	90.7	**83.8**	83.7

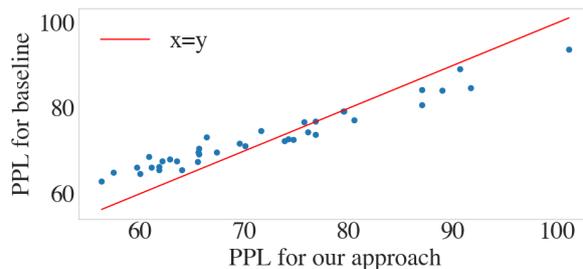Table 6: Expanding the language model training set.

Figure 1: Hyperparameter search results with one point for each configuration. The line separates where our approach is better (left) or worse (right).

but not in the language model training set.

Table 4 shows statistics about rare words in the datasets. 71-83% of word types in the training sets occur fewer than five times, but most of these appear frequently in the pretraining sets (compare the first column with the second last column). The same pattern occurs for word tokens. Comparing the statistics for the training set and the pretraining set, the percentage of rare word types is fairly consistent while the percentage of rare tokens consistently goes down.

Pretraining Data Needs to be from a Similar Domain. We would expect that the effectiveness of pretraining will depend on how similar the data is. Table 5 shows results with different embeddings, and indicates the number of words used in pretraining. We see that the value of additional data depends on the domain. Gigaword is also news text and is able to improve performance. The larger GloVe datasets use Wikipedia and CommonCrawl data, which is a poorer match and so does not improve performance. For GloVe we did have to change the embedding dimensions from 400 to 300, which may impact performance slightly.

The Effect Persists When Language Model Training Data is Increased. So far we have only used the additional in-domain data for pretraining. In this experiment, we expand the training set for the language model. We try two variations, one where the data is an exact domain match (NANC) and one where it is also news, but from different newspapers and from a different year (Gigaword). Table 6 shows that as we increase the amount of data our approach and the variant with random output embeddings continue to do best, but the margin shrinks between them and the standard approach. Note, however, that these results are with hyperparameters tuned for the baseline configuration. With tuning the 0.7 gap between our proposal and the baseline for 4xNANC widens to 6.6.

Hyperparameter Tuning Further Improves Results. All of the previous experiments were slightly tipped in favour of the baseline as we used the hyperparameters from Merity et al. (2017a). We do not have the resources to tune for every condition, so instead we focus on a final set of experiments with the 4xNANC condition from Table 6. We run 37 configurations with randomly sampled hyperparameters, using the same configurations for the baseline and our proposed approach (see the supplementary material for details). Figure 1 shows that our approach is even stronger after tuning, with a score that is 6.6 better than the baseline. Comparing the baseline and tuned hyperparameters, some shifted substantially more than others: the learning rate was halved; word dropout was halved; and the number of layers was increased from 3 to 4. The other parameters shifted by 15-30%.

Test Results Confirm Our Observations. Using the best configuration we train the baseline and our proposed approach using 8xNANC (the most our GPU could support). We compare to an n-gram language model trained on all of the NANC data (Heafield et al., 2013), and a transformer based model trained on a massive dataset, GPT-2 (Radford et al., 2019). While GPT-2 cannot be retrained in a low-compute scenario, it can be used. We compare to GPT-2 without fine-tuning. We evaluate byte-pair encoding (BPE) separately because with

Model	Words		BPE	
	Dev	Test	Dev	Test
N-Gram	92.3	95.0	56.7	55.3
GPT-2 (112m)	-	-	46.4	43.8
Baseline AWD-LSTM	52.8	53.5	37.8	36.7
Our approach	49.0	49.4	38.3	37.2
GPT-2 (774m)	-	-	32.5	33.7

Table 7: Final results, training with 8xNANC.

BPE tokenisation models have additional information when predicting the second or later piece of a token (Merity, 2019).

Table 7 shows that for word-level prediction, our approach improves over the baseline and an n-gram language model. BPE breaks up rare words, leading to no improvement over the baseline and while we do better than the 112m parameter GPT-2, we do not do as well as the 774m parameter one (both untuned). Overall, this indicates that for users who require word-level scores and have limited computational resources our approach is an effective way to use additional data when training LSTM language models.

4 Related Work

Embedding Tying. Tying input and output matrices has consistently increased performance while reducing the number of model parameters (Press and Wolf, 2017; Inan et al., 2017). The improvement is thought to be because otherwise only one input embedding is updated each step and the gradient has to propagate a long way through the model to reach it. Subsequent work has explored more advanced forms of tying, recognising that the role of the input and output matrices are not exactly the same (Pappas et al., 2018). This asymmetry has been found in the actual embedding spaces learned and shown to have a negative effect on performance (Gao et al., 2019; Demeter et al., 2020). These observations match the patterns we observe and provide theoretical justification for not tying when possible.

In-Domain Data Pretraining and Freezing. Word vectors are frequently used in downstream tasks and recent work has shown that their effectiveness depends on domain similarity (Peters et al., 2019; Arora et al., 2020) For language modeling, Kocmi and Bojar (2017) explored random and pretrained embeddings and found improvements, but did not consider tying and freezing. In-domain data is also useful for continuing to train contextual em-

bedding models before fine-tuning (Gu et al., 2020; Gururangan et al., 2020), and for monolingual pretraining in machine translation (Neishi et al., 2017; Qi et al., 2018; Artetxe et al., 2018). This matches our observations, but does not cover the interactions between freezing and tying we consider.

Handling Rare Words. These remain challenging even for large transformer models (Schick and Schütze, 2020). Recent work has explored copying mechanisms and character based generation (Kawakami et al., 2017), with some success. These ideas are complementary to the results of our work, extending coverage to the open vocabulary case. Due to space and computational constraints we only consider English. For other languages, inflectional morphology and other factors may impact the effectiveness of our approach (Shareghi et al., 2019; Cotterell et al., 2018). Our work is also complementary to concurrent work on producing rare words as output (Pappas and Mulcaire, 2020).

Language Model Types. We focus on a single model type for computational budget reasons. We chose an LSTM because while transformer based models such as GPT-2 now dominate transfer learning, LSTMs continue to be competitive in language modeling (Du et al., 2020; Li et al., 2020; Melis et al., 2018; Merity et al., 2017a). Our ideas are orthogonal to this prior work and our findings may apply to transformers as well, but confirming that would require additional experiments.

5 Conclusion

Initialising embeddings with vectors trained on in-domain data can improve performance by providing better representations for rare words. This effect persists even as more in-domain data is used to train the language model. Our work also suggests that standard model components like embedding tying should be retested as we continue to explore the space of language modeling.

Acknowledgements

References

Simran Arora, Avner May, Jian Zhang, and Christopher Ré. 2020. Contextual embeddings: When are they worth it? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2650–2663.

Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. 2018. Unsupervised neural machine translation. In *International Conference on Learning Representations (ICLR)*.

Chris Callison-Burch and Daphne Ippolito. 2020. Homework 4: Fine-tune a neural language model for text generation. http://interactive-fiction-class.org/homeworks/language-model/language-model.html.

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.

David Demeter, Gregory Kimmel, and Doug Downey. 2020. Stolen probability: A structural weakness of neural language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2191–2197.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. 2019. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2185–2194.

Wenyu Du, Zhouhan Lin, Yikang Shen, Timothy J. O'Donnell, Yoshua Bengio, and Yue Zhang. 2020. Exploiting syntactic structure for better language modeling: A syntactic distance approach. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6611–6628.

Jun Gao, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2019. Representation degeneration problem in training natural language generation models. In *International Conference on Learning Representations (ICLR)*.

David Graff. 1995. North american news text corpus LDC95T21. Web Download. Philadelphia: Linguistic Data Consortium.

Yuxian Gu, Zhengyan Zhang, Xiaozhi Wang, Zhiyuan Liu, and Maosong Sun. 2020. Train no evil: Selective masking for task-guided pre-training. *ArXiv*, abs/2004.09733.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. 2020. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360.

Kenneth Heafield, Ivan Pouzyrevsky, Jonathan H. Clark, and Philipp Koehn. 2013. Scalable modified Kneser-Ney language model estimation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 690–696.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *International Conference on Learning Representations (ICLR)*.

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. 2017. Learning to create and reuse words in open-vocabulary neural language modeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1492–1502.

Tom Kocmi and Ondřej Bojar. 2017. An exploration of word embedding initialization in deep-learning tasks. In *Proceedings of the 14th International Conference on Natural Language Processing (ICON-2017)*, pages 56–64.

Jonathan K. Kummerfeld, Sai R. Gouravajhala, Joseph J. Peper, Vignesh Athreya, Chulaka Gunasekara, Jatin Ganhotra, Siva Sankalp Patel, Lazaros Polymenakos, and Walter S. Lasecki. 2019. A large-scale corpus for conversation disentanglement. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3846–3856.

Yinqiao Li, Chi Hu, Yuhao Zhang, Nuo Xu, Yufan Jiang, Tong Xiao, Jingbo Zhu, Tongran Liu, and Changliang Li. 2020. Learning architectures from an extended search space for language modeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6629–6639.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

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 (ICLR)*.

Stephen Merity. 2019. Single headed attention rnn: Stop thinking with your head. *ArXiv*, abs/1911.11423.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017a. Regularizing and Optimizing LSTM Language Models. In *International Conference on Learning Representations (ICLR)*.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. An Analysis of Neural Language Modeling at Multiple Scales. *arXiv preprint arXiv:1803.08240*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017b. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*.

Masato Neishi, Jin Sakuma, Satoshi Tohda, Shonosuke Ishiwatari, Naoki Yoshinaga, and Masashi Toyoda. 2017. A bag of useful tricks for practical neural machine translation: Embedding layer initialization and large batch size. In *Proceedings of the 4th Workshop on Asian Translation (WAT2017)*, pages 99–109.

Mark Neumann, Daniel King, Iz Beltagy, and Waleed Ammar. 2019. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 319–327.

Nikolaos Pappas, Lesly Miculicich, and James Henderson. 2018. Beyond weight tying: Learning joint input-output embeddings for neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 73–83.

Nikolaos Pappas and Noah A. Mulcaire, Phoebe Smith. 2020. Grounded compositional outputs for adaptive language modeling. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Robert Parker, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda. 2011. English gigaword fifth edition LDC2011T07. Web Download. Philadelphia: Linguistic Data Consortium.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 157–163.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 101–108.

Ye Qi, Devendra Sachan, Matthieu Felix, Sarguna Padmanabhan, and Graham Neubig. 2018. When and why are pre-trained word embeddings useful for neural machine translation? 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 529–535.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Timo Schick and Hinrich Schütze. 2020. BERTRAM: Improved word embeddings have big impact on contextualized model performance. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3996–4007.

Ehsan Shareghi, Daniela Gerz, Ivan Vulić, and Anna Korhonen. 2019. Show some love to your n-grams: A bit of progress and stronger n-gram language modeling baselines. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4113–4118.

Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Darrin Eide, Kathryn Funk, Rodney Michael Kinney, Ziyang Liu, William. Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Christopher Wilhelm, Boya Xie, Douglas M. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. 2020. Cord-19: The covid-19 open research dataset. In *ACL NLP-COVID Workshop*.

A Initialisation without additional data

This experiment considers a variation where we pretrain the embeddings only on the PTB (i.e., there is no additional pretraining data). LM uses the embeddings produced by training a baseline model (i.e., train an LM, then reset all parameters except the embeddings and train again).

Tie	Input	Pretraining Method	Val PPL
📑	🔥	GloVe	64.5
📑	❄️	GloVe	66.2
📑	🔥	LM	61.7
◼️	🔥	🎲	61.3
◼️	🔥	GloVe	60.5
📑	❄️	LM	60.3
◼️	🔥	LM	59.4

While pretraining on the training data improves performance here, the improvement does not persist through the finetuning stage.

B Note on model size

When we untie the embeddings it does increase the number of parameters. We ran experiments with 200 dimensional embeddings and found the same trends, but all results were slightly worse.

C Reproducibility Criteria

For each item in the list we have a section below with the relevant information.

C.1 Experimental Results

A clear description of the mathematical setting, algorithm, and/or model.   The model we use is described in Merity et al. (2017a). We modify it to support weight freezing and initialisation.

For embeddings, we used GloVe with the same configuration as described in the original paper. For words in the LM training set that do not appear in the pretraining data, we used a random vector generated in the same way as Merity et al. (2017a).

A link to a downloadable source code, with specification of all dependencies, including external libraries   The code for our work is attached as supplementary material and available at `http://jkk.name/emnlp20lm/`. For the main experiments we used CUDA 10.1 and PyTorch 0.1 to get results consistent with those reported in Merity et al. (2017a).[5]

Description of computing infrastructure used   We used 7 GeFerce GTX TITAN X GPUs with 12212 Mb of RAM each. For the GPT-2-large experiments we used a Tesla T4 via Google Colab based on the notebook from Callison-Burch and Ippolito (2020).

Average runtime for each approach   Time per epoch varied from 60 to 2250 seconds depending on the amount of training data.

Number of parameters in each model   For the language model this varied from 24,221,600 to 105,737,253, depending on the training data used. Those values do not count all of the pretrained vectors though, only the ones that occurred in either the training, development, or test sets. The pretrained vectors depended on the volume of data:

- Cord: 126,386,800
- IRC: 26,050,000

---

[5]Later versions of PyTorch and their code led to slightly worse performance.

- NANC: 29,019,200
- Reddit: 77,719,200
- Wiki: 624,022,000
- Gigaword: 410,236,000

Corresponding validation performance for each reported test result   Only the final table reports test results and it also contains the relevant validation results.

Explanation of evaluation metrics used, with links to code   We use perplexity as implemented in (Merity et al., 2017a). The code is attached and available at `http://jkk.name/emnlp20lm/`.

C.2 Hyperparameter Search

We performed one hyperparameter search as described above.

Bounds for each hyperparameter   These values are named as defined in the AWD-LSTM arguments:

- lr, [10, 25].
- dropouti, [0, 0.5].
- dropoute, [0, 0.5].
- nhid, [650, 1650].
- nlayers, [2, 5].
- dropout, [0.25, 0.55].
- dropouth, [0.1, 0.4].
- wdrop, [0.35, 0.65].

We fixed clipping, batch size, bptt, and wdecay based on observations in prior work. For hyperparameter tuning we reduced the number of epochs to 100 as improvements beyond that point were usually very small. We also stopped early if the loss at epoch N was not lower than in epochs [0: N-10], as proposed in Merity et al. (2018).

Hyperparameter configurations for best-performing models   For most experiments we used the default hyperparameters from the AWD-LSTM, GPT-2 and kenlm. For the final experiments we used tuned hyperparameters as specified below.

Parameter	Tuned	Baseline
lr:	14.0	30
dropouti:	0.28	0.4
dropoute:	0.05	0.1
nhid:	1322	1150
nlayers:	4	3
dropout:	0.28	0.4
dropouth:	0.31	0.25
wdrop:	0.58	0.5

Number of hyperparameter search trials See above.

The method of choosing hyperparameter values (e.g., uniform sampling, manual tuning, etc.) and the criterion used to select among them (e.g., accuracy) All of the hyperparameters were simultaneously varied, sampling all uniformly at random. We selected the final set based on validation perplexity.

Expected validation performance, as introduced in Section 3.1 in Dodge et al, 2019, or another measure of the mean and variance as a function of the number of hyperparameter trials. We use Dodge et al. (2019)'s approach to produce the following plot, where the orange line is the baseline and our approach is the blue line:
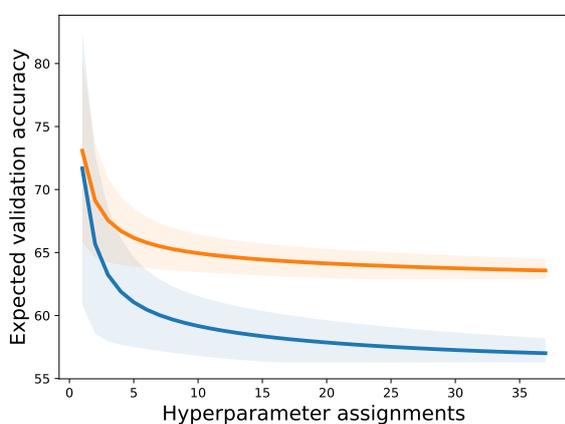
C.3 Datasets

We use several datasets:

- PTB: Mikolov's preprocessed version of the PTB (Marcus et al., 1993).

- PTB-Rare: Our version of the PTB.

- NANC: Wall Street Journal data from the North American News Corpus (Graff, 1995).

- Wiki: English Wikipedia.

- Reddit: A random sample of Reddit messages.

- CORD-19: Covid-19 research articles tokenised by Scispacy (Neumann et al., 2019).

- IRC: Ubuntu IRC dialogue disentangled by prior work (Kummerfeld et al., 2019).

- Gigaword: The Gigaword corpus (Parker et al., 2011).

Relevant statistics such as number of examples See below.

Details of train/validation/test splits

- PTB: We used the standard split: sections 00-20 for training, 21-22 for validation, 23-24 for test. These contain 88,7521, 70,390, and 78,669 tokens respectively.

- PTB-Rare: This is the same split as PTB.

- NANC: We randomly divide the data into splits the same size as PTB. For the additional LM data experiment we expand the training set first with the same amount of data again, then add another two times the data (adding the data each time, so smaller sets are subsets). The samples are not exactly the same size as the PTB training set as we add articles until we have just gone past the number of tokens. In all cases the pretraining, validation, and test set have 43,098,002, 72,356 and 79,408 tokens respectively. The 1x, 2x, and 4x training sets have 887,993, 1,776,407, and 3,551,496 tokens respectively. For the final experiment we use an 8x set (the 4x set plus another 4x) containing 7,101,988 tokens.

- Wiki: The same process as NANC. This gave pretraining, training, validation, and test sets of size 2,247,381,902, 887,933, 70,437, and 80,180 respectively.

- Reddit: The same process as NANC. This gave pretraining, training, validation, and test sets of size 219,940,812, 887,617, 70,439, and 78,796 respectively.

- CORD-19: The same process as NANC. This gave pretraining, training, validation, and test sets of size 194,840,142, 891,989, 73,014, and 81,648.

- IRC: The same process as NANC. This gave us pretraining, training, validation, and test sets of size 53,443,738, 887,716, 70,530, and 78,784.

- Gigaword: The same process as NANC for selecting 2x and 4x data. This gave us 4,790,293,007 tokens for pretraining, 888,869 extra tokens for 2x (added to the base NANC training data) and 2,664,487 extra tokens for 4x.

Explanation of any data that were excluded, and all pre-processing steps For all datasets aside from PTB and Reddit we used entire articles / conversations rather than breaking them into separate sentences. We applied preprocessing similar to the Mikolov PTB data, except that

we do not remove rare words. Our scripts for pre-processing are in the attached code and at `http://jkk.name/emnlp20lm/`.

- PTB: We used the raw data directly.
- PTB-Rare: All data, preprocessed with the `make-non-unk-ptb.py` script.
- NANC: We used all articles from the Wall Street Journal, concatenating lines to form complete articles. For BPE evaluation we used the GPT-2 tokeniser to prepare the data.
- Wiki: Extracted using `https://github.com/attardi/wikiextractor`, then removed text that contained 'colspan' or 'rowspan', as wikiextractor sometimes extracts parts of tables. Also excluded titles.
- Reddit: Used the Stanford CoreNLP tokeniser.
- CORD-19: We use all of the articles with pmc json data that are shorter than 20,000 tokens (99% of the data).
- IRC: We use all of the automatically disentangled conversations.
- Gigaword: We use all of the articles, removing duplicates.

A link to a downloadable version of the data The raw data is available at the links below. The preprocessing we applied is described above.

- PTB: `http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz`
- PTB-Rare: `https://catalog.ldc.upenn.edu/LDC99T42`
- NANC: `https://catalog.ldc.upenn.edu/LDC95T21`
- Wiki: `https://dumps.wikimedia.org/backup-index.html`
- Reddit: `https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/`
- CORD-19: `https://www.semanticscholar.org/cord19`
- IRC: `https://github.com/jkkummerfeld/irc-disentanglement`
- Gigaword: `https://catalog.ldc.upenn.edu/LDC2011T07`

For new data collected, a complete description of the data collection process, such as instructions to annotators and methods for quality control. We did not collect any new data.