Finetuning Can Be Efficiently Approximated With Unigram Bias

Richard Antonello
UT Austin
rjantonello@utexas.edu

Shivang Singh
UT Austin
shivang.singh@utexas.edu

Alexander Huth
UT Austin
huth@cs.utexas.edu

Abstract

Finetuning a pretrained language model on a new text corpus requires specialized hardware and significant computational expenditure. We observe that a substantial proportion of the finetuning task can be reduced to learning token unigram probabilities in the new corpus. Based on this observation, we propose an easily implementable technique that gives a first-order approximation of standard finetuning at little computational cost. Combined with some basic statistical guardrails, this technique can be used to immediately and substantially reduce perplexity of a pretrained language model without requiring any gradient backpropagation.

1 Introduction

Language modeling is the task of generating language from context. This task is typically made differentiable by allowing language models to output a probability distribution over candidate tokens. Recent developments in natural language processing have produced substantial improvements in the effectiveness of language models (LMs), culminating in powerful attention-based Transformer architectures (Vaswani et al., 2017; Radford et al., 2019). These new architectures significantly exceed the performance of older RNN and LSTM-based architectures as measured by perplexity, a common loss metric for the language modeling tasks (Irie et al., 2019). However, Transformer-based language models are notoriously difficult to train from scratch, and generally require the use of many high-end GPUs, which is infeasible for most computational settings and for many researchers (Kitaev et al., 2020; Popel and Bojar, 2018). To partly alleviate these concerns, the paradigm of pretrained models has been developed, where pretrained language models are released by researchers or corporations with the resources required to generate them. These models can be downloaded and finetuned through backpropagation to better capture a specific corpus or task. This avoids the difficulties involved in training a state-of-the-art language model from scratch, as finetuning is comparatively fast and requires less data (Howard and Ruder, 2018). However, even finetuning can be resource-intensive, as it still requires some expensive hardware and significant time/energy expenditure. In settings where this computational power does not exist or cannot be efficiently invoked, such as mobile devices, these pretrained models cannot be specialized through finetuning and therefore have limited value (Yu et al., 2018). Additionally, recent state-of-the-art language models such as GPT-3 (Brown et al., 2020) are so heavily parameterized that even finetuning these models is financially infeasible for all but the most well-funded researchers.

In this paper, we propose an alternative to standard finetuning of pretrained models. We show that merely adjusting the unigram probability distribution outputted by a language model is enough to significantly reduce perplexity on a target corpus. Using this observation, we design a simple method for finetuning a language model on a new corpus without requiring significant computational expenditure or specialized hardware. While this technique is not as effective as full finetuning, its computational efficiency enables better use of pretrained models in environments and systems with scarce computational resources.

2 Motivation

To motivate our approach, we first show that much of the benefit of finetuning pretrained language models comes from learning the unigram probability distribution of the target corpus. To isolate the effect of learning only this distribution

and no other information about the target corpus, we performed a simple experiment using the pretrained language model GPT-2 Small (Radford et al., 2019). Since the exact unigram token probabilities in the source corpus for GPT-2 Small model are unknown, we had to first finetune on a defined source corpus of Reddit comments (Huth et al., 2016). We define this finetuned model as our "pretrained model", for which we know the exact source unigram distribution. We then finetuned this model on a target corpus, the Toronto Books corpus. For this second finetuning step, we froze the weights of all layers in the network except the final weight layer, which feeds into the softmax operator that outputs the predicted probability distribution over tokens. This final layer constitutes a small percentage of the total number of the free parameters of the LM architecture. It is only capable of modifying the relative probability of tokens and cannot alone encode more nuanced information about the propensity of certain tokens to cooccur or specific phrase structures. We then finetuned the model via backpropagation, only permitting this last layer to be modified with the loss gradient.

Figure 1 compares the effectiveness of finetuning on just the last layer (*blue*) compared to finetuning the model without frozen weights (*orange*). While it is clear that finetuning the entire model performs better, a substantial fraction of the benefit of finetuning can be gleaned by only learning the final weight layer. This suggests that much of what is learned during finetuning is the relative probabilities of tokens in the new target corpus compared to the source corpus. If we could modify the predicted token probabilities without using backpropagation, then we could receive this benefit without requiring any expensive training. This is the motivation for our approach, which we call *instantaneous finetuning* or *IF*. The red line in the figure shows the result of using this technique on this same pair of corpora, showing that it is indeed capable of matching the effectiveness of finetuning on the last layer of the network.

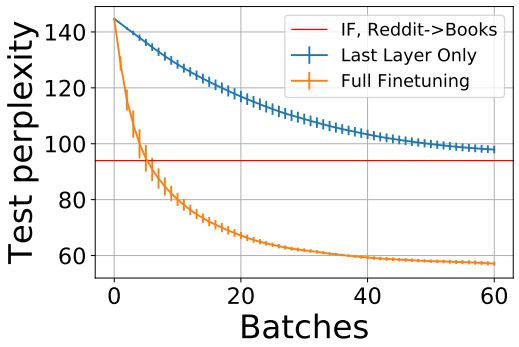


Figure 1: *A Substantial Portion of LM Finetuning is Learning Relative Unigram Probabilities*. Freezing all but the output weight layer of GPT-2 during finetuning (*blue*) only cuts the improvement to perplexity by half compared to full finetuning (*orange*). This suggests that a significant part of the learning that occurs during LM finetuning is simply reconfiguring unigram probabilities to match the new target corpus. Our method inexpensively captures this portion of the learning. The finetuning curves are averaged over 50 runs. Error bars show the standard error.

3 Instantaneous Finetuning

3.1 Basic Methodology

To generate a probability distribution over tokens, neural network language models typically apply the softmax operation to a vector of continuous real values that represent the unscaled relative log likelihoods of observing each token. Our method modifies these relative log likelihood values through the addition of a bias vector that is designed to optimally match token probabilities on the target corpus. To do this, we compare the unigram probability distribution of source corpus for the original pretrained model with that of the target corpus. We define the unigram distribution over the source corpus as P_{old}, and the unigram distribution over the training set for the target corpus as P_{new}. We then use the tokenwise log-ratios between P_{old} and P_{new} as the output bias layer, forcing the unigram distribution of the pretrained model to match that of the target corpus. More formally, let $P(X = x)$ be the probability of token x in distribution P. For every token x_i, we define the output bias as

$$B(x_i) = \log \frac{P_{new}(X = x_i)}{P_{old}(X = x_i)} \quad (1)$$

This simple and computationally inexpensive update constitutes the entirety of instantaneous finetuning. To understand why this procedure works, let us examine the effect of this update on the result of the softmax operation. Recall the softmax operation:

$$\textbf{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Let $x'_i = x_i + B(x_i)$ be the updated logit value for token x_i. When we update x_i to x'_i, we modify the

final softmax probability distribution accordingly:

$$\textbf{Softmax}(x'_i) = \frac{e^{x'_i}}{e^{x'_i} + \sum_{j \neq i} e^{x_j}}$$

In the event when the numerator is negligible compared to the size of the denominator (as it will be for most x_i), we can approximate and simplify:

$$\begin{aligned}\textbf{Softmax}(x'_i) &\approx \frac{e^{x'_i}}{\sum_j e^{x_j}} \\ &= \frac{e^{x_i + \log \frac{P_{new}(X=x_i)}{P_{old}(X=x_i)}}}{\sum_j e^{x_j}} \\ &= \frac{P_{new}(X = x_i)}{P_{old}(X = x_i)} \cdot \textbf{Softmax}(x_i)\end{aligned}$$

This method only requires a single read of the training sets of the source and target corpora in order to generate the token probability distributions and thus the values for the final bias layer. It also has the benefit of treating the more than 99.99% of the LM that precedes the final softmax as a black box. This allows this method to easily be combined with neural network compression techniques that significantly reduce the parameter space and memory footprint of the LM at the expense of trainability (Ullrich et al., 2017; Tang and Lin, 2018), providing some of the functionality of finetuning to compressed models.

3.2 Improving Stability

Applying basic instantaneous finetuning without using any statistical guardrails to ensure stability of the updated language model can cause issues. In particular, one issue arises when a token appears very rarely in either the source corpus or the target corpus. For example, if a token appears with probability 10^{-8} in P_{old} and 10^{-6} in P_{new}, then direct application of IF would require that the final bias for that token is modified by $\log 100$. For rare tokens the estimated probabilities can be noisy, as they are based on a very small number of examples. Exactly modifying the bias based on noisy probabilities can lead to poor LM performance. For tokens that do not have large enough sample sizes in either the source or target distribution relative to the size of the respective corpora, we recommend not modifying the corresponding bias term.

An additional concern arises when a given token is vastly overrepresented in the target corpus compared to the source corpus. This may arise when this token takes on new significance in the target corpus, or when its use in the target corpus has changed entirely from the source corpus. Instantaneous finetuning is unsuitable for these circumstances, as it does not allow the LM to learn new contextual information about tokens, so we also suggest bounding the maximum update for a given token by a reasonably sized value.

Section 4 shows that including these stability heuristics is critical for the performance of instantaneous finetuning.

4 Results

For our tests, we used the publicly available Transformer-based pretrained 124 million parameter language model GPT-2 Small, as implemented by the open-source `transformers` library (Wolf et al., 2019). Since we did not have access to the precise preprocessing and tokenization steps used to generate the training corpus for GPT-2 Small, we first performed standard finetuning using SGD with Adam to finetune the LM to a corpus of Reddit comments (Huth et al., 2016). We then treated this corpus as the de facto "source" corpus from which P_{old} is computed. For our target corpus, we used the Toronto Books corpus (Zhu et al., 2015), from which we computed P_{new}. Our first experiment compares the performance of instantaneous finetuning between these two corpora to the performance of standard finetuning. Figure 1 shows the relative performance of each of the two methods as measured by perplexity. For each application of standard finetuning in this figure, we used SGD with Adam (Kingma and Ba, 2015) with a learning rate of 5×10^{-5}, $\beta_1 = 0.9$, and $\beta_2 = 0.999$ with a mini-batch size of 16 contexts of 32 tokens each per batch. We only modified token biases using instantaneous finetuning if there were at least 1 instance of the token in the target corpus, and we set minimum and maximum bounds of $(\log 1/10, \log 10)$ on the size of the update to the final pre-softmax bias layer.

4.1 Analyzing LM Stability

As noted in Section 3.2, due to noisy estimation of probabilities for rare tokens, it is necessary to either clip the size of the bias update or limit the bias update only to tokens that are sufficiently com-

[1]For min sample sizes of zero, a bound of at least one was still applied to the target corpus to prevent division by zero.

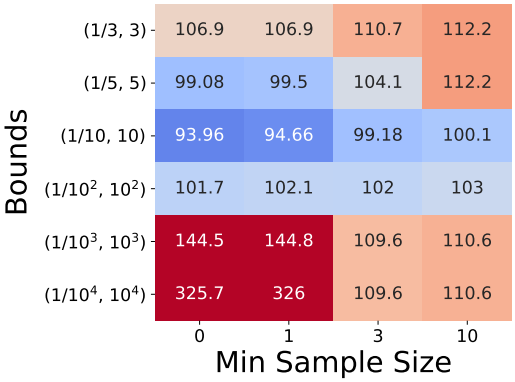


Figure 2: The evaluation perplexity of the result of instantaneous finetuning starting from the Reddit corpus and training to the Books corpus for a variety of constraints is shown above. The rows show the effect of bounding the maximum change to the bias term by the log of the bounds given in the leftmost column. The columns show the effect of changing the constraint on the minimum number of tokens of that type, where each number in the bottom row represents the minimum number of tokens that must be observed in both the source and target corpora for that token to be modified by a non-zero bias term.[1] These perplexities can be compared to the baseline model before applying instantaneous finetuning, which has a perplexity of **144.6**. The perplexity of instantaneous finetuning with *no constraints* was extremely high at **1909.2**. It is clear from this that the use of stability constraints are essential for the effective instantaneous finetuning.

mon in both the source and target corpora. Adding these limitations requires us to set several free parameters, specifically the bounds on the bias updates and the minimum number of token samples required to modify a given token. We show the results of a simple experiment that illuminates the importance of these restrictions on the method. Figure 2 shows the final perplexity for different choices of these free parameters. We see that relaxing these parameters,which allows rare tokens to be modified, or allows the modifications to be large, results in worse LM performance. One of these free parameters *Min Sample Size* controls the rarity of tokens in the bias term that get modified. The other parameter *Bounds* controls the degree to which the bias layer can get modified We see that relaxing these parameters results in worse LM performance. The best performance comes from a moderate bound on the size of the log ratio $(0.1, 10)$, and no threshold on the number of samples. Figure 3 shows the statistics of the effects of these bounds over the corpora and suggest that these bounds mostly work to limit the effect of rarest 20% of tokens. Similar results as in Figure 2 have been replicated on other corpora pairs and are included as supplementary material. These tests suggest that the ideal parameters may change slightly from corpus to corpus, however using the bounds $(\log 1/5, \log 5)$ is close to optimal for all tested corpora pairs. When a bound on the bias size is used, then a bound on the minimum sample size is not necessary. This may be because bounding the update size implicitly limits the effect of small sample sizes.

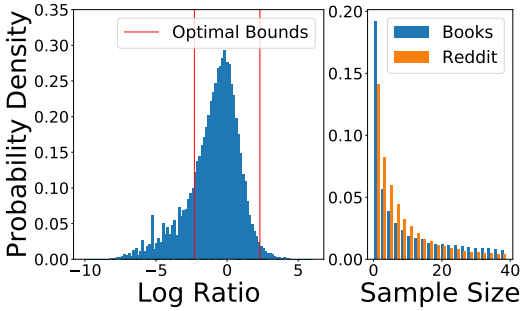


Figure 3: *Distribution of Log Ratio and Sample Size of Rare Tokens*: The left-hand figure shows the distributions of log ratios for each token bias term, with red lines delineating where the empirically optimal (0.1, 10) bounds lie for the Reddit to Books corpora pair. We can see that the red lines roughly match where the distribution of log ratios begins to become noisy, suggesting that the accuracy of the log ratio computation begins to break down beyond these points. The right-hand figure shows a truncated distribution of number of occurrences of each token for each corpus.

5 Conclusion and Future Work

We have proposed a simple and easy-to-implement method for approximating LM finetuning that requires minimal computation and yields substantial improvements over a baseline pretrained model. Our method only requires the token probability distributions for the source corpus that was used to pretrain the model and the target corpus. Including simple clipping and sample size bounds makes this method an effective and efficient alternative to language model finetuning in settings where computational resources are scarce.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin

Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Jeremy Howard and Sebastian Ruder. 2018. Fine-tuned language models for text classification. *ArXiv*, abs/1801.06146.

Alexander G Huth, Wendy A De Heer, Thomas L Griffiths, Frédéric E Theunissen, and Jack L Gallant. 2016. Natural speech reveals the semantic maps that tile human cerebral cortex. *Nature*, 532(7600):453–458.

Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. 2019. Language modeling with deep transformers. In *INTERSPEECH*.

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

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *ArXiv*, abs/2001.04451.

Marie Lebert. 2008. Project gutenberg (1971-2008).

Martin Popel and Ondrej Bojar. 2018. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110:43 – 70.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8).

Raphael Tang and Jimmy Lin. 2018. Adaptive pruning of neural language models for mobile devices. *ArXiv*, abs/1809.10282.

Karen Ullrich, Edward Meeds, and Max Welling. 2017. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Seunghak Yu, Nilesh Kulkarni, Haejun Lee, and Jihie Kim. 2018. On-device neural language model based word prediction. In *Proceedings of the 27th International Conference on Computational Linguistics: System Demonstrations*, pages 128–131, Santa Fe, New Mexico. Association for Computational Linguistics.

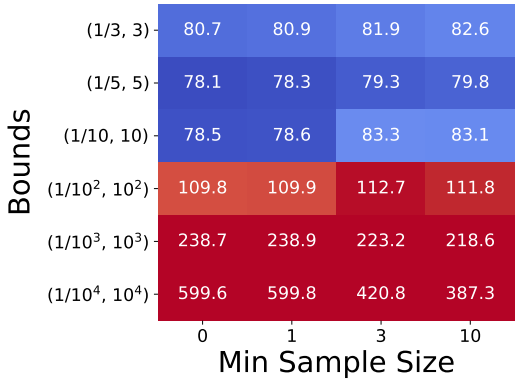
Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.

A Supplementary Material

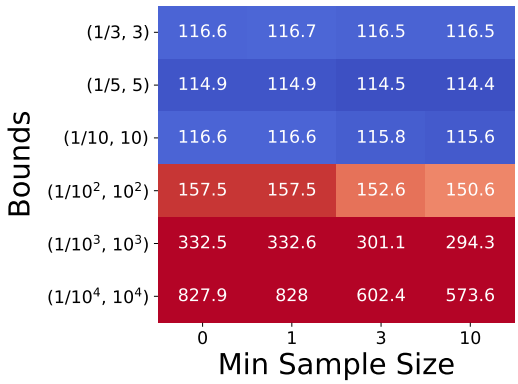
A.1 Miscellaneous Figures

Below are some additional stability constraint tables similar to Figure 2, for different pairs of source and target corpora. For two of these, we used as an additional corpus a set of books from the Project Gutenberg public domain database (Lebert, 2008).

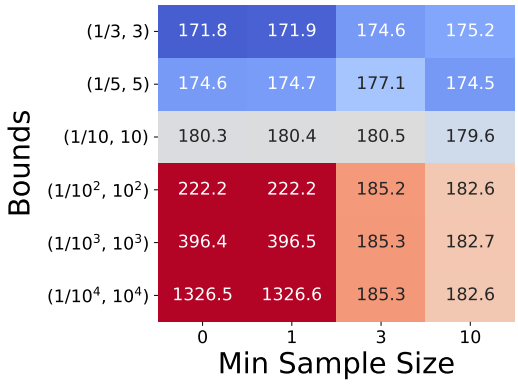
Books to Reddit (Baseline Perplexity: **101.2**)



Books to Gutenberg (Baseline Perplexity: **144.4**)



Reddit to Gutenberg (Baseline Perplexity: **181.6**)



A.2 Code Outline

The supplementary material also includes a Python code outline that goes about implementing instantaneous finetuning. It can be found in the attached `if.py` file.