Efficient Training of Language Models with Compact and Consistent Next Token Distributions

Ashutosh Sathe **Sunita Sarawagi**
Indian Institute of Technology, Bombay
`{absathe,sunita}@cse.iitb.ac.in`
https://github.com/ashutoshbsathe/CoCoNTs

Abstract

Maximizing the likelihood of the next token is an established, statistically sound objective for pre-training language models. In this paper we show that we can train better models faster by pre-aggregating the corpus with a collapsed n-gram distribution. Previous studies have proposed corpus-level n-gram statistics as a regularizer; however, the construction and querying of such n-grams, if done naively, prove to be costly and significantly impede training speed, thereby limiting their application in modern large language model pre-training.

We introduce an alternative compact representation of the next token distribution that, in expectation, aligns with the complete n-gram distribution while markedly reducing variance across mini-batches compared to the standard next-token loss. Empirically, we demonstrate that both the n-gram regularized model and our approximation yield substantial improvements in model quality and convergence rate compared to existing methods. Furthermore, our approximation facilitates scalability of gains to larger datasets and models compared to the straightforward n-gram regularization method.

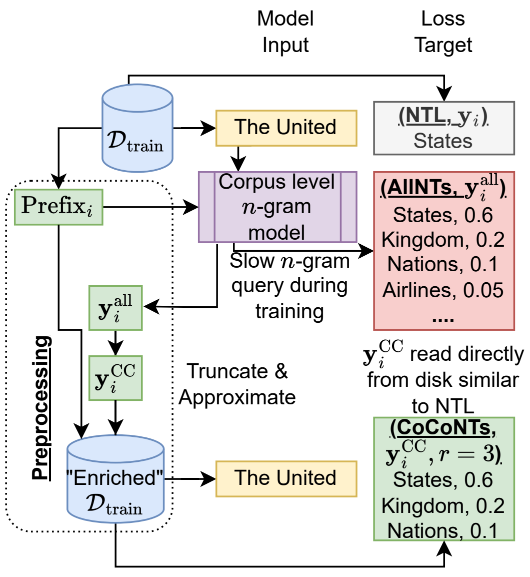
Model Input
Loss Target
$\mathcal{D}_{\text{train}}$
The United
(NTL, $\mathbf{y}_i$)
States
Prefix$_i$
Corpus level n-gram model
(AllNTs, $\mathbf{y}_i^{\text{all}}$)
States, 0.6
Kingdom, 0.2
Nations, 0.1
Airlines, 0.05
....
Slow n-gram query during training
$\mathbf{y}_i^{\text{all}}$
$\mathbf{y}_i^{\text{CC}}$
Truncate & Approximate
$\mathbf{y}_i^{\text{CC}}$ read directly from disk similar to NTL
Preprocessing
"Enriched" $\mathcal{D}_{\text{train}}$
The United
(CoCoNTs, $\mathbf{y}_i^{\text{CC}}, r = 3$)
States, 0.6
Kingdom, 0.2
Nations, 0.1

Figure 1: **Comparison of various training methods.** Standard next-token likelihood reads inputs as well as targets from the disk. n-gram augmented methods (AllNTs) obtain targets ($\mathbf{y}_i^{\text{all}}$) by querying an n-gram model which can be slow during training. Our proposed method, CoCoNTs, truncates and approximates the $\mathbf{y}_i^{\text{all}}$ and stores the preprocessed distribution ($\mathbf{y}_i^{\text{CC}}$) along with the dataset itself for faster retrieval during training.

1 Introduction

Since the advent of the first neural language models (NLM) (Bengio et al., 2003; Mikolov et al., 2013), a standard approach to training NLMs has been to maximize the likelihood of the next token (NT) given the preceding tokens in randomly sampled token sequence from the dataset. In this paper we show that LLM pre-training can be made significantly more efficient by supervising with a distribution over multiple possible next tokens, instead of a single next token. Earlier studies (Neubig and Dyer, 2016; Zhao et al., 2017; Yang et al., 2019) have explored using corpus-level n-gram statistics to improve the quality of RNN based language models. This can also be thought of as label-smoothing (Szegedy et al., 2016; Müller et al., 2019) with n-gram estimated distribution.

In this paper we show that regularizing with corpus level n-gram statistics continues to be beneficial even on current language models, both when pre-training from scratch or fine-tuning. However, a major hurdle is scaling up such techniques to today's corpus and model size. We address these challenges by (1) proposing a compact representation of the next token distribution while continuing to be statistically consistent. and (2) designing data structures to efficiently make use of these distributions without stalling throughput-optimized linear algebra accelerators (XLA devices) such as TPUs. In particular, we find that our proposed data handling strategy and objective (named "CoCoNTs", pronounced "coconuts") often matches the performance of n-gram models with full (non-truncated) distribution (Sec. 5.2). Notably, while the prior n-gram augmented methods use storage proportional to the size of training dataset, our storage overheads are *constant* with respect to size of the training dataset (Sec. 5.3). We also find that n-gram augmented training methods (prior works as well as CoCoNTs) can reach the same validation perplexity as the NTL objective in nearly 50% less optimization steps (Sec. 5.3). Our proposal is useful for improving model quality in both pre-training (Sec. 5.6) as well as fine-tuning – all parameter (Sec. 5.2) or parameter efficient (Sec. 5.5).

Our key contributions can be summarized as: (1) We highlight the usefulness of training with n-gram statistics for faster convergence of language models, and propose a statistically consistent truncation strategy of making their implementation scalable and practical for current data and model sizes (Sec. 4). (2) We show how this truncated distribution can be efficiently retrieved during training time without slowing down XLA devices (Sec. 4.1, Sec. 4.2). (3) We discuss practical strategies and their effects on model performance of scaling this strategy to potentially very large scale datasets via sharding (Sec. 5.3). (4) We thoroughly test our proposal through ablation studies, comparisons with prior works on both fine-tuning as well as pre-training of language models (Sec. 5).

2 Related Work

Language modeling. Early neural language models proposed in (Bengio et al., 2003) and (Mikolov et al., 2013) were trained to maximize the likelihood of the next token in randomly sampled training batches. While the Transformer architecture (Vaswani et al., 2017) has been the powerhouse of modern LLMs (Black et al., 2022; Zhang et al., 2022; Touvron et al., 2023), the key training objective has remained the same. Several studies have explored alternate objectives such as unlikelihood training (Welleck et al., 2020) or contrastive loss (Su et al., 2022; Jain et al., 2023) to improve text generation from these LMs. However, these objectives are not statistically consistent, and have not been adopted in large scale pre-training.

Augmenting language model training. Some prior work (Mikolov et al., 2011; Chelba et al., 2014; Jozefowicz et al., 2016) have proposed extending RNN based LMs (Bengio et al., 2003; Mikolov et al., 2013) with KN smoothed n-gram LMs, and shown that these result in a better language model. This sparked interest (Neubig and Dyer, 2016; Zhao et al., 2017; Yang et al., 2019) to introduce corpus level n-gram statistics into an otherwise local training procedure. Frydenlund et al. (2022) replaced the standard cross-entropy loss with a ranking based loss for which they used n-grams as a weak teacher to obtain "gold" rankings. Li et al. (2020) explore additional KL divergence penalty similar to Yang et al. (2019) and our work. However, they obtain ground truth distribution by similarity between word embeddings trained on the corpus.

Our work is closest to the n-gram regularized loss of Yang et al. (2019) but we propose a modified compact supervision that scales to large corpus. We also compare our models against (Li et al., 2020) to study whether KL divergence from learned word vectors can serve as a better proxy for relatively expensive count based n-gram models.

3 Language Modeling

Let $\mathcal{V}$ denote a vocabulary of tokens. Given a token sequence $\mathbf{t} : t_1, t_2 \ldots, t_i$ where each t_j in an integer index into the vocabulary $\mathcal{V}$, a language model $P_\theta(t|t_1 \ldots t_n) \in \mathbb{R}^{|\mathcal{V}|}$ assigns a multinomial distribution of the probability of the possible next token that could follow $\mathbf{t}$. The size of the token sequence $\mathbf{t}$ is limited to a maximum length L. For training θ we are given a corpus $\mathcal{D}_{\text{train}} = x_1, \ldots, x_N$ where N is typically very large $N \gg L$.

A standard method of training is to maximize the likelihood of the next token (NTL or NT) given the preceding tokens in a randomly sampled snippet of length L from $\mathcal{D}_{\text{train}}$. Let $x_j, \ldots, x_{j+L}$ denote such a snippet sampled at position j of $\mathcal{D}_{\text{train}}$. The training objective then becomes

$$\hat{\theta}_D = \text{argmax}_\theta \sum_{n=1}^{L} \log P_\theta(x_{j+n+1}|x_j, \ldots, x_{j+n}) \tag{1}$$

When a prefix $\mathbf{x}_{j:n} = x_j, \ldots, x_{j+n}$ has multiple occurrences in the corpus, then for the same context depending on the sampling position j, the target may be different. But the model P_θ needs to converge to a consistent distribution. For example, a prefix "The United" could occur multiple times within a corpus with different possible next tokens as shown in Figure 1. If $\mathbf{t} = t_1, \ldots, t_n$ denotes the tokens in such a prefix, it can be seen that at convergence, the next token likelihood $P_\theta(t|t_1,\ldots,t_n)$ for a token sequence is expected to be equal to the empirical distribution of tokens following $\mathbf{t}$ over the entire corpus $\mathcal{D}_{\text{train}}$. Let $\mathbf{y}_\mathbf{t}^{\text{all}}$ denote the fractional frequency of the occurrence of tokens of $\mathcal{V}$ following all different positions where $\mathbf{t}$ appears in the corpus $\mathcal{D}_{\text{train}}$. At convergence we expect:

$$P_{\hat{\theta}_D}(t|\mathbf{t}) \longrightarrow \mathbf{y}_\mathbf{t}^{\text{all}}, \quad \text{where,}$$
$$\mathbf{y}_\mathbf{t}^{\text{all}}[w] = \frac{\sum_{j\in D}\delta(\mathbf{x}_{j:n}=\mathbf{t}, x_{j+n+1}=w)}{\sum_{j\in D}\delta(\mathbf{x}_{j:n}=\mathbf{t})} \tag{2}$$

In this paper we investigate if the above convergence to the corpus-level next-token distribution can be sped up via changing the training objective to directly match the target distribution $\mathbf{y}_\mathbf{t}^{\text{all}}$. For long prefixes $\mathbf{t}$ we do not expect too much repetition, and also maintaining the $\mathbf{y}_\mathbf{t}^{\text{all}}$ proportions for all possible prefixes may incur too much overhead. So, we fix a maximum prefix length k, and instead optimize for a mixture of these two objectives.

The AllNTs objective.

$$\min_\theta \sum_{n=k+1}^{L} -\log P_\theta(x_{j+n+1}|\mathbf{x}_{j:n}) + \sum_{n=1}^{k} \text{KL}\left(\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}; P_\theta(\cdot|\mathbf{x}_{j:n})\right) \tag{3}$$

The above loss is reminiscent of the use of the corpus-level n-gram statistics to regularize LM training (Yang et al., 2019; Neubig and Dyer, 2016). The second term goes over all n-grams $\mathbf{t}$ of length from 1 to k, and attempts to match the learned model distribution to the observed fraction of next tokens in the corpus following that n-gram $\mathbf{t}$.

Benefits of AllNTs. When we supervise the model to match empirical distribution on all possible next tokens after a prefix, the convergence of the model is expected to be faster. In the empirical section we will show that training with even small n-grams ($k=4$), gives rise to much lower perplexity for the same computation budget than the original training only for next token likelihood (Eq 1).

Overheads of AllNTs. For imposing the AllNTs loss we need to create a data structure like a trie, which for each possible prefix can provide the distribution of next tokens (Heafield, 2011; Heafield et al., 2013). Querying the trie for every sampled mini-batch is inefficient. These inefficiencies are especially noticeable (Sec. 5.3) when scaling to larger datasets (>1B tokens) as the batch creation (which includes trie lookup) on CPU is slow. In the next section we show an alternative method for supervising the next token distribution that significantly reduces these overheads.

4 Compact and Consistent Next Tokens: CoCoNTs

We propose to approximate the full empirical next token distribution $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}$ with a more compact and consistent supervision $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}}$ at each sampled prefix $\mathbf{x}_{j:n}$. Unlike $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}$ which can be of size as large as the vocabulary size, the alternative we propose is of size at most $r+1$ where r is a chosen hyperparameter, like 4 or 8 in our experiments. We design $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}}$ so that in expectation over the minibatches, $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}}$ matches the AllNTs supervision but where the variance across loss terms is significantly smaller than via the supervision in the NT objective.

Let $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{topr}}$ denote a truncation of the $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}$ where only the top r largest fractions are retained and the rest of the truncated to zero. Let $\mathbf{1}_\mathcal{V}(t_i)$ be one-hot encoding of size $|\mathcal{V}|$ with component for t_i as 1. We approximate $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}$ with $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}}$ as follows:

$$\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}} = \begin{cases} v\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{topr}} & \text{if } x_{j+n+1} \in \mathbf{y}_{\mathbf{x}_{j:n}}^{\text{topr}} \\ u\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{topr}} + \mathbf{1}_\mathcal{V}(x_{j+n+1}) & \text{otherwise} \end{cases} \tag{4}$$

Choosing u, v. We choose u, v such that $\mathbb{E}(\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}}) \approx \mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}$ and $|\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}} - \mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}|$ is minimized. Let $p = \sum_t \mathbf{y}^{\text{topr}}[t]$ denote the total probability mass covered by the top-r highest probability tokens. An example appears in Figure 1. For a token $t \in \mathcal{V}$ that is outside the top-r tokens in $\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}$, it is clear that $\mathbb{E}(\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}})[t] = \mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}[t]$. Now consider a token t in the top-r set. We want to determine values of u, v s.t. $\mathbb{E}(\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}})[t] = \mathbf{y}_{\mathbf{x}_{j:n}}^{\text{all}}[t]$

$$\mathbb{E}(\mathbf{y}_{\mathbf{x}_{j:n}}^{\text{CC}}[t]) = \mathbf{y}^{\text{topr}}[t](vp + u(1-p)) = \mathbf{y}^{\text{topr}}[t]$$
$$\implies v = \frac{1-(1-p)u}{p}$$

This shows that we are "stealing" some probability mass from the top-r token positions and assigning them to the rare token positions, so that across all repetitions of the prefix we have consistent supervision on the next token distribution. By choosing u carefully we can control this consistency. The value of u has to be in the range $[0, 1/(1-p)]$ for

$\mathbf{y}^{\text{CC}}$ to be non-zero for all positions. The distance $|\mathbf{y}^{\text{CC}}_{\mathbf{x}_{j:n}} - \mathbf{y}^{\text{all}}_{\mathbf{x}_{j:n}}|$ can be computed as

$$|\mathbf{y}^{\text{CC}}_{\mathbf{x}_{j:n}} - \mathbf{y}^{\text{all}}_{\mathbf{x}_{j:n}}| \begin{cases} = (1-p)u & \text{if } x_{j+n+1} \in \mathbf{y}^{\text{topr}}_{\mathbf{x}_{j:n}} \\ \leq 2 - up & \text{otherwise} \end{cases}$$

As u increases from zero, the supervision at the rare token positions is made closer to the ideal, but at the cost of the frequent token positions. We define a hyper-parameter $\gamma > 1$, and choose $u = \frac{1}{\gamma - p}$. While p varies with prefixes $\mathbf{x}_{j:n}$ and datasets, we found that a value of $\gamma = 1.5$ performs well across diverse settings. We show effectiveness of the approximation via an example in Appendix A.2.

4.1 Pre-enriching the dataset with $\mathbf{y}^{\text{CC}}$

The above design of $\mathbf{y}^{\text{CC}}$ allows us to tackle the core computational inefficiency of AllNTs. Instead of incurring the CPU overheads of trie-lookups during training, we propose to pre-enrich the training corpus with the compact $\mathbf{y}^{\text{CC}}$ distributions stored along with the corpus. The total storage cost becomes only $(L+2kr)/L$ times the original storage cost but we avoid the expensive trie lookup operations during training. Also, we can build the entire trie in-memory only once during the pre-processing phase and discard after the data enrichment phase.

We implement our trie in C, similar to Heafield (2011); Heafield et al. (2013). Each TrieNode consists of the count and a HashMap where key is the next token ID and value is a pointer to the next TrieNode. The HashMap is implemented using an AVL tree. We implement a Top-r query method which returns Top-r token IDs and Top-r probabilities for a given prefix of upto k tokens using Eq. 2.Concretely, we first construct the trie by reading sequences of k tokens from the dataset and inserting it in the trie. At every level i, we increment the count by 1 to implicitly record prefix $[t_1 \ldots t_i]$. Once the trie is constructed, we start reading (disjoint) sequences of L tokens from the dataset and writing the "enriched" sequence back to disk. To "enrich" a sequence $x_j, \ldots, x_{j+L}$, we first look up the prefix in the trie and get pointers to k nodes i.e. one at each level. At each of these nodes, the HashMap is storing $\mathbf{y}^{\text{all}}$. We can efficiently traverse this HashMap in descending order of fractions to get $\mathbf{y}^{\text{topr}}$. Once we get Top-$r(\mathbf{y}^{\text{all}}_{j:i})$ $\forall i \in [0, k)$, we can write the "enriched" sequence to disk. Note that, this operation still requires disk storage of $(L + (L + 2kr)) \times N$ tokens where N is the total number of sequences. We present additional discussion and a sample walkthrough of this procedure for more clarity in Appendix A.3.

4.2 Building the mini-batch with $\mathbf{y}^{\text{CC}}_i$

To build the mini-batch of size B in standard NTL, one simply reads sequences of L tokens B times and concatenates them. In standard NTL, the sequence of L tokens itself is both input and target but in CoCoNTs, we have 2 targets that need to be built. In CoCoNTs we need to read sequence of $L + 2kr$ tokens from the disk and use the first L tokens to form $\mathbf{x}, \mathbf{y}$ similar to NTL. Next $2r$ "tokens" correspond to $\mathbf{y}^{\text{topr}}$ where the first token is the actual token ID and the next token is really the count of that token appearing after prefix $[t_1]$. The $2r$ tokens following this would correspond to Top-$r(\mathbf{y}^{\text{all}}_2)$ which encodes counts and tokens appearing after prefix $[t_1, t_2]$. This would repeat for k times to give k distributions as targets for KL divergence in Eq. 3. We provide additional discussion on memory footprint in Appendix A.4.

5 Experiments

Through the experiments, we empirically validate the effectiveness of our approximation (Sec. 5.2) along with the ablation studies on various hyperparameters. We also evaluate the training efficiency of CoCoNTs in Sec. 5.3. Finally, we present two case studies on CoCoNTs in pre-training (Sec. 5.6) and parameter-efficient fine-tuning (Sec. 5.5) to further demonstrate the usefulness and relevance of CoCoNTs even in modern LLM usecases. Hyperparameters for all the experiments are presented in Appendix A.1.

5.1 Datasets, Baselines and Metrics

We explore the effectiveness of CoCoNTs in fine-tuning existing models on WikiText-103 (Merity et al., 2017), MiniPile (Kaddour, 2023; Gao et al., 2020) and a subset of PubMed-Abstracts (Luo et al., 2022). The WikiText-103 training split consists of $\approx$ 114M tokens while MiniPile and PubMed splits consist of $\approx$ 1.6B and 2.6B tokens respectively.

For our full fine-tuning experiments, we compare CoCoNTs against AllNTs (Yang et al., 2019) and NTL objectives on WikiText-103. We compare against D2GPO baseline (Li et al., 2020) as well since they also use a KL divergence based augmentation of training loss.

Since Su and Collier (2023) raised concerns about isotropy of the base gpt2-125m (Radford et al., 2019) model, we also study effects of CoCoNTs objective on gpt-neo-125m (Black et al., 2021) and opt-125m, opt-1.3B (Zhang et al., 2022).

Metrics. Following (Su et al., 2022), we evaluate each fine-tuning method several on model quality and text quality metrics.

- Perplexity (**ppl**) of the model on the test set.
- Prediction accuracy (**acc**) of the model. Given a sample with inputs $[\mathbf{x}_1 \ldots \mathbf{x}_L]$ and labels $[\mathbf{y}_1 \ldots \mathbf{y}_L]$ from the test set, we take argmax of each of the L predicted distribution at time step t to get top-1 predicted token and match it against $\mathbf{y}_t$ to calculate the prediction accuracy.
- Repetition (**rep**) measures the fraction of top-1 next-token predictions that occur in the prefix.
- Expected calibration error (**ECE**) measures how over/underconfident is the model when making correct/incorrect predictions. Lower ECE indicates better calibrated models.
- **MAUVE (MVE)** (Pillutla et al., 2021) measures the similarity between the generated text and reference text using the embeddings of another large pretrained model.
- Repetition within a generated single text sequence: (**rep-n**) $100 \times (1 - \frac{|\text{unique } n\text{-grams}|}{|\text{total } n\text{-grams}|})$.
- Diversity (**div.**) measures repetition at different n-gram levels: $\prod_{n=2}^{4}(1 - \frac{\text{rep-}n}{100})$.
- Number of unique bigrams (**#uniq**) in the generated text.

We also compare the Zipf coefficient of the generated text to gold text as suggested by (Meister et al., 2023). For generating text, we either use greedy decoding or nucleus (Holtzman et al., 2020) sampling. To compare training efficiency, we use following metrics:

- Number of optimization steps (**steps-to-ppl**) taken to reach NTL's perplexity on the val set.
- Total wall clock time (**TWT**) to finish the entire training. We exclude the time for preprocessing (trie building and storage) as the preprocessing is a one-time operation which many times did not take very long. Additionally, one could always start with datasets that are preprocessed by someone else. We do include the time it takes to load the trie and training time retrieval as these operations will often stall the XLA devices.
- Total disk usage (**disk**) required for training (includes storage of prefix trie).
- Maximum CPU RAM (**max-RAM**) used for pretraining i.e. loading and using the trie.

5.2 Model performance

CoCoNTs is comparable to AllNTs and better than NTL. In Table 1, we compare performance of CoCoNTs with various other objectives. We find that CoCoNTs is able to provide consistent gains over the NTL baseline. We also notice that CoCoNTs outperforms D2GPO (Li et al., 2020) which indicates that count based conditional n-gram models are able to provide a stronger training signal as compared to word embedding similarity. Maximum gains with CoCoNTs are observed on gpt2-125m, however this could be related to isotropy of gpt2-125m checkpoint as discussed by (Su and Collier, 2023). Across all metrics and models, AllNTs is generally the best performing model while CoCoNTs comes close to it. CoCoNTs offers gains (albeit modest as compared to small models) to larger models (opt-1.3B) as well.

5.3 Training efficiency

CoCoNTs is significantly more efficient than AllNTs. Figure 2 compares our training efficiency metrics across various training methods on WikiText-103 and PubMed datasets. AllNTs uses Python's `defaultdict` [1] to implement the trie HashMap while CoCoNTs uses our efficient implementation of HashMap as described in Sec. 4.1. AllNTs serializes the resultant trie to disk using the Python's `pickle`[2] library. We also explore using an existing n-gram implementation (Heafield, 2011) with Python FFI as denoted by AllNTs-CFFI. Because AllNTs stores the n-gram model on disk for later retrieval, its disk overhead grows with the size of the dataset as opposed CoCoNTs's constant (dependant on k, r only) disk overhead. Measuring the total wall clock time, we find that CoCoNTs ($k = 4$) is *faster* than AllNTs ($k = 2$) which highlights the benefits of our approximation and pre-enriching of the dataset.

Results on PubMed demonstrate challenges in scaling up vanilla AllNTs to large datasets. On our machine with 256GB RAM, the prefix trie for $k = 8$ with vanilla AllNTs did not fit in memory. It is possible that AllNTs-CFFI is able to fit everything in memory similar to CoCoNTs but we could not explore in depth since our n-gram implementation (Heafield, 2011) kept crashing on our system. While AllNTs-CFFI saves disk and RAM,

[1] https://docs.python.org/3/library/collections.html#collections.defaultdict

[2] https://docs.python.org/3/library/pickle.html

Model	Model Quality				Generations [greedy]			Generations [nucleus]			
	PPL	Acc.	Rep.	ECE	Div.	#Uniq	Zipf	Div.	#Uniq	Zipf	MVE
					gpt2-125m						
NTL	24.279	39.667	52.597	0.075	14.051	87756	0.990	94.504	112768	0.952	0.708
D2GPO	22.151	42.540	52.270	0.068	14.823	93563	0.945	95.289	122590	0.866	0.774
AllNTs	20.634	44.170	48.922	0.068	30.678	127266	0.958	99.415	161138	0.911	0.900
CoCoNTs	20.717	42.192	50.300	0.069	30.030	121498	0.959	95.751	156432	0.932	0.863
					gpt-neo-125m						
NTL	22.746	43.528	49.843	0.055	28.937	99326	0.956	96.809	118061	0.976	0.636
AllNTs	20.188	46.308	47.632	0.049	32.603	101725	0.927	96.066	121729	0.910	0.685
CoCoNTs	20.208	44.147	49.014	0.051	31.178	99080	0.950	96.919	120084	0.942	0.699
					opt-125m						
NTL	19.374	44.583	51.707	0.067	26.789	90484	1.022	94.771	114472	0.978	0.648
AllNTs	19.371	49.282	50.027	0.054	33.767	95887	0.994	97.690	135104	0.951	0.670
CoCoNTs	18.558	49.059	50.071	0.052	34.722	99168	0.972	95.111	141256	0.934	0.703
					opt-1.3B						
NTL	16.554	49.686	45.617	0.052	29.137	101649	0.939	97.723	164789	0.912	0.861
CoCoNTs	15.679	50.010	45.617	0.051	29.866	102492	0.941	98.111	166846	0.941	0.869
Gold					89.036	171076	0.925	89.036	171076	0.925	1.000

Table 1: **Results on WikiText-103.** We find that CoCoNTs is competitive with far more expensive (as we show in Sec. 5.3) AllNTs objective. For both AllNTs and CoCoNTs, $k = 8$ was used to build the prefix trie. CoCoNTs additionally used $r = 8$. Best results are highlighted with green while second-best are highlighted with yellow.

it still is not as efficient as CoCoNTs on WikiText-103. Moreover, the trie loading time as well as the (somewhat) slow CFFI interface serialization overheads significantly increase total wall time of AllNTs-CFFI over CoCoNTs.

Effects of sharding. As compared to datasets used in this work, modern LLMs (Black et al., 2022; Touvron et al., 2023; Groeneveld et al., 2024) are often trained on far bigger web corpora (Gao et al., 2020; Together Computer, 2023; Soldaini et al., 2024) for which building an n-gram model in-memory may not be feasible. In such cases, we show that such datasets can be sharded into multiple small datasets of several few billion tokens with each shard being enriched with its own n-gram index. We study the effect of sharding by simulating it on MiniPile and PubMed datasets and seeing effect on perplexity as shown in Fig. 3. We find that after a certain threshold of shards, the number of tokens per shard decreases so much that KL penalty can become overly optimistic and result in worse perplexity. In general, we found that having more than billion tokens per shard was sufficient to get results close to AllNTs while still using modest amount of RAM.

5.4 Understanding CoCoNTs

Ablation studies on k and r. By default we choose $k = 8, r = 8$. We report ablations on these values when fine-tuning gpt2-125m on WikiText-103 and compare using validation perplexity. As indicated by trends in Fig. 4, we find that increasing k and r leads to predictable improvements in perplexity as compared to the NTL baseline. We do notice a significant difference in perplexities between AllNTs and CoCoNTs for lower values of k. This could be potentially due to poorer $\mathbf{y}_i^{\text{CC}}$ approximation since the fan-out is expected to be significantly higher for initial few tokens. Empirically, in the first few levels of the trie, the branching factor is the highest often leading to $\mathbf{y}_i^{\text{all}}$ that has a support size much larger than r. This is further supported by the significant improvement observed when going from $r = 2$ to $r = 4$ for a fixed k. Moreover, as k increases, the support of $\mathbf{y}_i^{\text{all}}$ naturally decreases. In fact for $k > 2$, the average support size is less than 4 on WikiText-103. This implies that $p = 1$ in Eq. 4 leading to $v = 1$ which effectively reduces CoCoNTs to AllNTs objective.

Using larger/domain specific data sources to build n-gram models can help. We show that using n-gram statistics from alternative corpus is also useful. To study this, we augmented our existing WikiText-103 n-gram trie with n-grams from MiniPile and PubMed datasets independently. As shown in Table 2, model trained with WikiText-103 + MiniPile index improves perplexity on both WikiText-103 *and* MiniPile. On the other hand, if the augmenting dataset (PubMed) is both large *and*

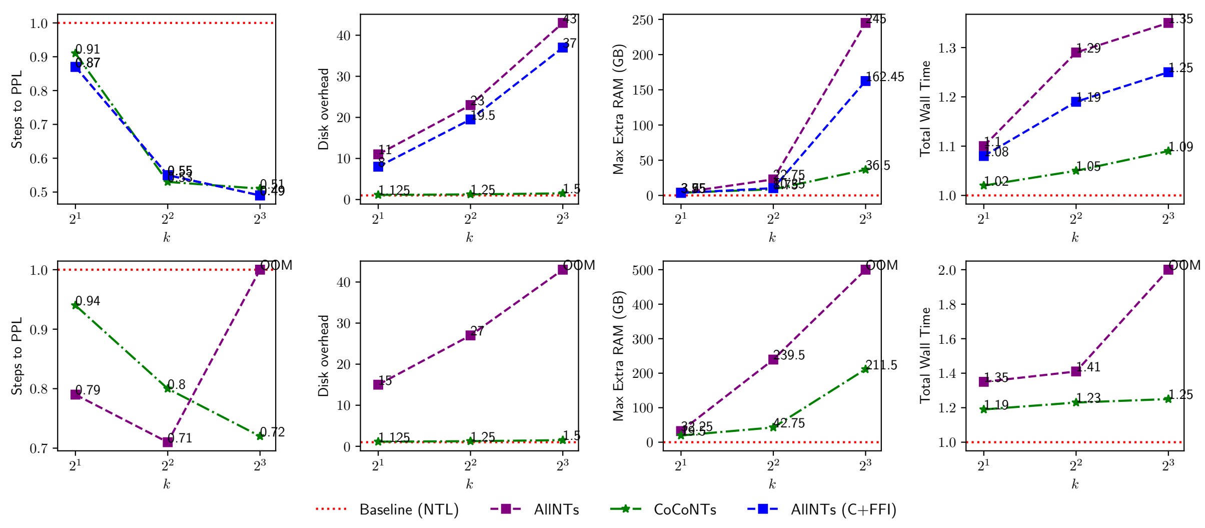

Figure 2: **Comparison of training efficiency on WikiText-103 (top) and PubMed (bottom).** AllNTs with higher values of k can easily go out of memory from a naive implementation. Both AllNTs and CoCoNTs converge faster to NTL's validation perplexity as compared to NTL. The total wall time (TWT) to finish the entire training is also significantly lower with CoCoNTs as compared to AllNTs due to lack of any n-gram querying during training. gpt2-125m model is used for all experiments with $r = 8$ for CoCoNTs.

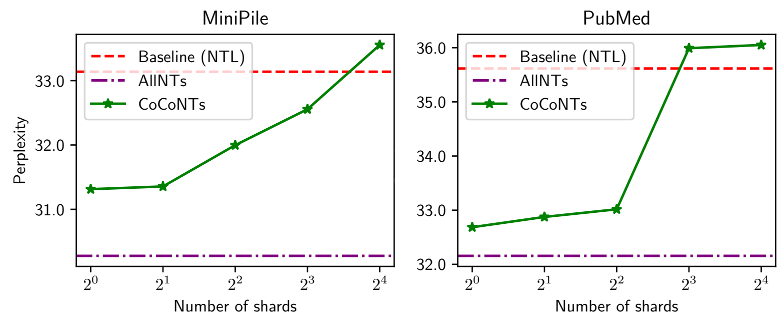

Figure 3: **Effect of sharded CoCoNTs on large datasets.** Oversharding can make the n-gram distribution unreasonably sparse. This can lead to overly optimistic approximation and KL penalty which can hurt the performance on extremely small indices.

	Wiki-103	MiniPile	PubMed
gpt2-125m	24.279	43.456	37.819
AllNTs	20.634	41.261	38.538
CoCoNTs	20.717	42.486	39.016
+MiniPile	**20.223**	**36.899**	38.564
+PubMed	20.926	39.432	**36.175**

Table 2: **Perplexity improvements when augmenting n-gram index with larger/domain specific data.** Each model (row) is trained on WikiText-103 and evaluated on validation splits of (column) WikiText-103, MiniPile and PubMed. CoCoNTs with indices built on additional MiniPile or PubMed data improves performance on respective datasets.

domain specific, the resultant model improves on augmenting dataset *at the cost* of performance on the original (WikiText-103) dataset.

Perplexity reduction as a function of prefix length. We show that even though we impose the CoCoNTs loss only in a small prefix of an overall sequence of length 256, the improvement in model quality is throughout the length of the sequence. For this, we measure the negative log-likelihood (NLL) separately at each position from 1 to 256 in the test data and show our findings in Figure 5.We observe that we reduce NLL at all sequence positions over the two existing methods. This suggests that in today's NLM architectures with shared parameters, enforcing the correct loss in only a subset of the positions is sufficient for overall gains.

Effect of longer pretraining. We study that pretraining for more steps flattens the benefits of CoCoNTs slightly but not completely. We continued pretraining of gpt-neo-125m from our experiments (Sec. 5) for 20k more steps (50% more than original pretraining steps) and report our findings in Table 3. Looking at the validation perplexities, we do see that the difference between baseline and CoCoNTs perplexities does decrease from 2.5 to 2.3

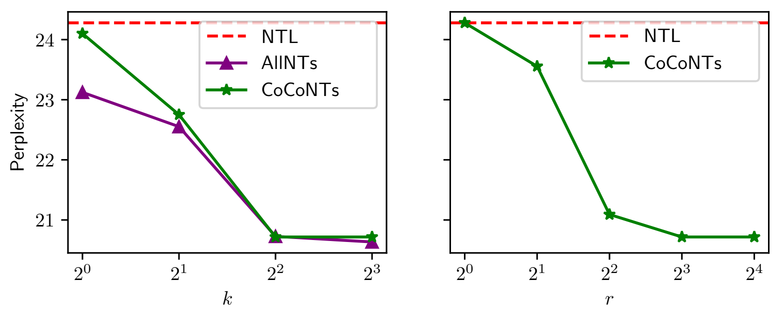

Figure 4: **Ablations studies on k and r for AllNTs and CoCoNTs.** All experiments fine-tune a gpt2-125m model on WikiText-103. Higher values of both k and r improve perplexity before plateauing. $r = 8$ is fixed when varying k and $k = 8$ is fixed when varying r for CoCoNTs.

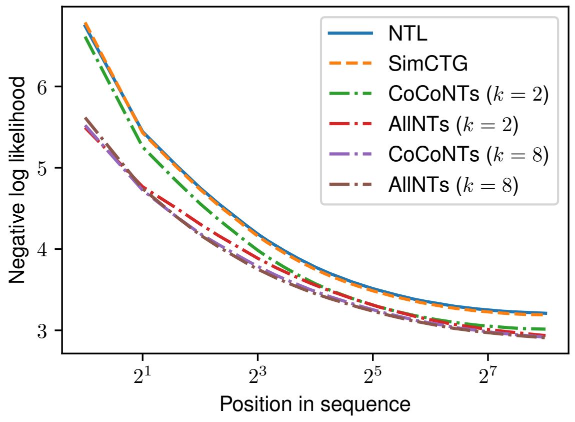

Figure 5: **Perplexity as a function of position in sequence.** Both AllNTs and CoCoNTs show smooth changes in perplexity despite applying loss to only a small k token prefix.

Method	Pretraining Step					
	10k	20k	30k	40k	50k	60k
NTL	26.3	23.9	23.0	22.7	22.5	22.4
AllNTs	25.5	22.4	21.2	20.2	20.2	20.0
CoCoNTs	25.9	22.6	21.3	20.2	20.2	20.1

Table 3: **Effect of pretraining for more steps on validation perplexity.** Both AllNTs and CoCoNTs use $k = 8$. CoCoNTs also uses $r = 8$. The difference in CoCoNTs and the baseline slightly reduces but does not become completely zero.

over the 20k additional steps. To correctly assess if this difference goes to zero, significantly longer training is required.

5.5 Case Study: PEFT Domain Adaptation

Parameter efficient fine-tuning (PEFT) of LLMs (Hu et al., 2022; Dettmers et al., 2023) has become a popular method to adapt general purpose LLMs such as LLaMA-2 (Touvron et al., 2023) on a specific domain. To study benefits of CoCoNTs objective in this direction, we first train a LoRA (Hu et al., 2022) to fine-tune LLaMA-2-7B on our

	PubMedQA	MedQA	TWT
NTL	56.025	36.911	1x
AllNTs	**56.976**	**38.343**	1.57x
CoCoNTs	56.251	37.732	**1.32x**

Table 4: **Results with LoRA domain adaptation.** The base LLM (LLaMA-2-7B) is finetuned on PubMed abstracts and subsequently finetuned on each of the QA tasks with LoRA. CoCoNTs continues to outperform NTL while staying close to AllNTs. The Total Wallclock Time (TWT) of AllNTs is much higher than ours.

	CoLA (MCC)	MRPC (F1)	RTE (Acc)	TWT
NTL	0.288	0.735	0.591	1x
AllNTs	**0.347**	**0.771**	**0.629**	1.35x
CoCoNTs	0.339	0.742	0.621	**1.19x**

Table 5: **Results on the BabyLM Strict Challenge.** The base model (opt-125m) trained with CoCoNTs performs similar to AllNTs while being significantly faster than AllNTs, as measured by the Total Wallclock Time (TWT). Both methods are better than NTL on quality.

split of the PubMed dataset using each of the fine-tuning method. Both AllNTs and CoCoNTs use $k = 4$ while CoCoNTs uses $r = 8$. With this LoRA as the starting point, we fine-tune the resultant (Medical LLaMA) model on 2 medical QA (PubMedQA (Jin et al., 2019) and MedQA (Jin et al., 2021)) tasks independently. As summarized by accuracy of these QA tasks in Table 4, we find results similar to full fine-tuning i.e. CoCoNTs performs better than NTL but worse than AllNTs. However, AllNTs incurs an almost 60% increase in pre-training time, whereas our methods reduces the overheads by half as seen by the last total wallclock time (TWT) column in the table.

5.6 Case Study: The BabyLM Challenge

Can CoCoNTs be used to pre-train a better language model for downstream tasks? We pre-train an opt-125m architecture model from scratch on the data from the BabyLM challenge (Warstadt et al., 2023). We use the nearly 100M word data from the "strict" track with standard preprocessing. Once all (NTL, AllNTs, CoCoNTs) the models are pre-trained, we finetune each on 3 downstream tasks (with recommended hyperparameters) from the evaluation suite – CoLA (Warstadt et al., 2019), MRPC (Dolan and Brockett, 2005) and RTE (Dzikovska et al., 2013) which are subsets of the SuperGLUE benchmark (Wang et al., 2019b,a) for the BabyLM challenge. Task specific fine-tuning does not use any custom objective. Table 5 show metrics on respective tasks. Both AllNTs and CoCoNTs trained base LMs show significant improvement over standard NTL in CoLA which is about linguistic acceptability. This could be due to grammatically incorrect sentences/completions automatically being suppressed in the n-gram index. On other downstream tasks such as paraphrase detection (MRPC) or entailment (RTE), the performance of all models is much closer in comparison. Also, observe that the total wall clock time (TWT) for AllNTs is 35% higher than baseline, and CoCoNTs reduces the overhead down to 19%.

6 Conclusion

In this work, we revisited the benefits of regularizing language model training with corpus-level n-gram statistics, and proposed ways to scale up their implementation on current scales of data and model sizes. Our proposal truncates the n-gram estimated next-token distribution and introduces a novel method of mixing with occurrences of frequent and rare tokens so as to provide low-variance supervision of the distribution of next tokens. The distributions are designed to be compact and can be stored with the corpus so that their retrieval is as simple as a disk read operation. We empirically show that CoCoNTs performs comparable to AllNTs objective but is significantly more efficient than AllNTs. Notably, while the AllNTs storage costs scale with dataset, CoCoNTs storage costs depend only on the hyperparameters k and r. Our fine-tuning experiments suggest that CoCoNTs based training benefits smaller models the most with larger models seeing only modest improvements. We also observe that imposing AllNTs or CoCoNTs loss only on a small k token prefix was sufficient to improve the overall model performance. Case study on the BabyLM challenge highlights that CoCoNTs trained base LMs are better than standard NTL trained base LMs on downstream tasks as well.

Limitations and Future Work

In this section, we discuss the limitations of CoCoNTs objective and provide insights into potential challenges and areas for improvement. The one-time preprocessing step required by our method for very large-scale datasets requires sharding. Effects of such sharding on social biases of the model must be studied carefully. When applying our method to very large datasets like C4 or The Pile, the implementation of the prefix-trie using better optimized libraries as discussed in (Jurafsky and Martin, 2023) may become necessary. This could require significant engineering efforts to optimize access times and ensure efficient training. To address scalability concerns further, a possible suggestion in addition to sharding would be to "sparsify" the trie for such large-scale datasets. By pruning branches with low counts, we can significantly reduce the overall memory footprint while still maintaining the essence of (idea of "heavy hitters" (Misra and Gries, 1982; Woodruff, 2016; Braverman et al., 2017)) the next token distribution.

When applying our method to very large datasets like C4 or The Pile, the implementation of the prefix-trie using better optimized libraries as discussed in (Jurafsky and Martin, 2023) may become necessary. This could require significant engineering efforts to optimize access times and ensure efficient training. To address scalability concerns further, a possible suggestion in addition to sharding would be to "sparsify" the trie for such large-scale datasets. By pruning branches with low counts, we can significantly reduce the overall memory footprint while still maintaining the essence of the empirical next token.

Furthermore, it is important to acknowledge that our CoCoNTs objective aims to match the empirical next-token distribution, and thus inherits any biases present in the training data. However, an advantage of our approach is that the prefix trie allows for detailed exploration and identification of these biases. If such biases are observed, it should be possible to edit the prefix trie to mitigate their influence. Our experimental setup included the largest models and datasets that could run comfortably on our compute resources. Future work can explore effectiveness of CoCoNTs on even larger datasets (e.g. ThePile (Gao et al., 2020), RedPajama (Together Computer, 2023), Dolma (Soldaini et al., 2024) etc.) and larger scale models such as LLaMA2 (Touvron et al., 2023).

Ethics Statement

We acknowledge that our objective entails preprocessing and handling large-scale datasets for creating the prefix trie. This necessitates careful attention to privacy concerns and the implementation of robust data protection measures. It is vital to thoroughly examine and mitigate any biases that may emerge in the training data prior to the application of our proposed objective.

Moreover, given the improved text generation capabilities demonstrated by our approach, it is imperative to address ethical considerations regarding the responsible use of language models trained using our proposed objective. In this context, we underscore the significance of ensuring that the deployment and utilization of such models align with ethical standards, including but not limited to mitigating the potential for malicious use, promoting fairness in algorithmic decision-making, and safeguarding user privacy.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(null):1137–1155.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. GPT-NeoX-20B: An open-source autoregressive language model. In *Proceedings of the ACL Workshop on Challenges & Perspectives in Creating Large Language Models*.

Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. 2021. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow. If you use this software, please cite it using these metadata.

Vladimir Braverman, Stephen R. Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P. Woodruff. 2017. Bptree: An $l2$ heavy hitters algorithm using constant memory. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, page 361–376, New York, NY, USA. Association for Computing Machinery.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Myroslava Dzikovska, Rodney Nielsen, Chris Brew, Claudia Leacock, Danilo Giampiccolo, Luisa Bentivogli, Peter Clark, Ido Dagan, and Hoa Trang Dang. 2013. SemEval-2013 task 7: The joint student response analysis and 8th recognizing textual entailment challenge. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 263–274, Atlanta, Georgia, USA. Association for Computational Linguistics.

Arvid Frydenlund, Gagandeep Singh, and Frank Rudzicz. 2022. Language modelling via learning to rank. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(10):10636–10644.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The Pile: An 800GB dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. 2024. Olmo: Accelerating the science of language models.

Kenneth Heafield. 2011. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland. Association for Computational Linguistics.

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, Sofia, Bulgaria. Association for Computational Linguistics.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Nihal Jain, Dejiao Zhang, Wasi Ahmad, Zijian Wang, Feng Nan, Xiaopeng LI, Ming Tan, Baishakhi Ray, Parminder Bhatia, Xiaofei Ma, Ramesh Nallapati, and Bing Xiang. 2023. Contraclm: Contrastive learning for causal language model. In *ACL 2023*.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2021. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Applied Sciences*, 11(14):6421.

Qiao Jin, Bhuwan Dhingra, Zhengping Liu, William Cohen, and Xinghua Lu. 2019. PubMedQA: A dataset for biomedical research question answering. 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 2567–2577, Hong Kong, China. Association for Computational Linguistics.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling.

Daniel Jurafsky and James H. Martin. 2023. *N-gram Language Models (Chapter 3) in Speech and language processing*, 3rd. ed. edition.

Jean Kaddour. 2023. The minipile challenge for data-efficient language models. *arXiv preprint arXiv:2304.08442*.

Zuchao Li, Rui Wang, Kehai Chen, Masso Utiyama, Eiichiro Sumita, Zhuosheng Zhang, and Hai Zhao. 2020. Data-dependent gaussian prior objective for language generation. In *International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Renqian Luo, Liai Sun, Yingce Xia, Tao Qin, Sheng Zhang, Hoifung Poon, and Tie-Yan Liu. 2022. BioGPT: generative pre-trained transformer for biomedical text generation and mining. *Briefings in Bioinformatics*, 23(6). Bbac409.

Clara Meister, Tiago Pimentel, Gian Wiher, and R̃yan Cotterell. 2023. Locally Typical Sampling. *Transactions of the Association for Computational Linguistics*, 11:102–121.

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

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, page 3111–3119, Red Hook, NY, USA. Curran Associates Inc.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531.

J. Misra and David Gries. 1982. Finding repeated elements. *Science of Computer Programming*, 2(2):143–152.

Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. 2019. When does label smoothing help? In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Graham Neubig and Chris Dyer. 2016. Generalizing and hybridizing count-based and neural language models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1163–1172, Austin, Texas. Association for Computational Linguistics.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaïd Harchaoui. 2021. MAUVE: measuring the gap between neural text and human text using divergence frontiers. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 4816–4828.

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

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. 2024. Dolma: An Open Corpus of

Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*.

Yixuan Su and Nigel Collier. 2023. Contrastive search is what you need for neural text generation. *Transactions on Machine Learning Research*.

Yixuan Su, Tian Lan, Yan Wang, Dani Yogatama, Lingpeng Kong, and Nigel Collier. 2022. A contrastive framework for neural text generation. In *Advances in Neural Information Processing Systems*.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019a. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019b. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

Alex Warstadt, Aaron Mueller, Leshem Choshen, Ethan Wilcox, Chengxu Zhuang, Juan Ciro, Rafael Mosquera, Bhargavi Paranjabe, Adina Williams, Tal Linzen, and Ryan Cotterell. 2023. Findings of the BabyLM challenge: Sample-efficient pretraining on developmentally plausible corpora. In *Proceedings of the BabyLM Challenge at the 27th Conference on Computational Natural Language Learning*, pages 1–34, Singapore. Association for Computational Linguistics.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2020. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*.

David P. Woodruff. 2016. New Algorithms for Heavy Hitters in Data Streams. In *19th International Conference on Database Theory (ICDT 2016)*, volume 48 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:12, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Yiben Yang, Ji-Ping Wang, and Doug Downey. 2019. Using large corpus n-gram statistics to improve recurrent neural language models. 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 3268–3273, Minneapolis, Minnesota. Association for Computational Linguistics.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big Bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297. Curran Associates, Inc.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models.

Zhe Zhao, Tao Liu, Shen Li, Bofang Li, and Xiaoyong Du. 2017. Ngram2vec: Learning improved word representations from ngram co-occurrence statistics. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 244–253, Copenhagen, Denmark. Association for Computational Linguistics.

A Appendix

A.1 Hyperparameters

Most of our experiments are performed on a single TPUv2-8 core VM with TPU VM architecture. We also sometimes used 4x NVIDIA A100 GPUs with FlashAttention for a fraction of all experiments.

For fine-tuning experiments (Sec. 5.2), we start with publicly available checkpoints for gpt2-125m[3], gpt-neo-125m[4], opt-125m[5] and opt-1.3B[6]. Each 125m parameter model is trained for 40k steps with AdamW (Loshchilov and Hutter, 2019) optimizer with effective batch size of 192. The maximum learning rate was set to 10^{-4} with 10% of maximum steps as warmup followed by cosine decay to zero. For the 1.3B parameter model, we set the maximum steps to 10k and reduce the batch size to 32. Each fine-tuning run took roughly 5-6 hours on TPUs and 8-10 hours on GPUs.

For pre-training on the BabyLM challenge (Sec. 5.6), we set the batch size, optimizer and learning rate schedule similar to fine-tuning and trained for total of 5 epochs. For downstream tasks, we use the hyperparameters mentioned in the BabyLM evaluation pipeline[7]. The pre-training took 5.5 hrs on TPUs while fine-tuning on downstream tasks took 1-1.5hrs each.

For parameter-efficient fine-tuning of LLaMA (Sec. 5.5), we set the batch size to 32 and fine-tune on PubMed with hyperparameters similar to WikiText-103 fine-tuning. We use the same parameters as BabyLM downstream tasks for LLaMA PEFT downstream tasks as well. The pre-training took close to 6 hours on TPUs while fine-tuning on downstream tasks took 2-2.5hrs each.

A.2 Effectiveness of $\mathbf{y}^{\text{CC}}$

Example. We show a simple example to illustrate how $\mathbf{y}^{\text{CC}}$ manages to reduce variance in the next-token distribution across sampled mini-batches. Assume vocab size is 5, and $\mathbf{y}_{\mathbf{t}}^{\text{all}} = [0.6, 0.3, 0.1, 0.05, 0.05]$. Let $r = 2$. Assume $\gamma = 1.5$. For this case we have $p = 0.9, u = 1.66, v = 0.5$. Everytime we sample a mini-batch where next token is from the top-2 set: $\{1, 2\}$, we supervise with $\mathbf{y}^{\text{CC}} = v[0.6, 0.3, 0, 0, 0]$ which has a distance of 0.166 from $\mathbf{y}^{\text{all}}$. Whereas, when we sample the last three tokens we are at most 0.51 from $\mathbf{y}^{\text{all}}$. Contrast this to the baseline NT case where for rare token the distance to $\mathbf{y}^{\text{all}}$ could be as high as 1-0.05=0.95! With $\mathbf{y}^{\text{CC}}$ we reduce this distance to 0.51. Even for frequent tokens the distance has been reduced from a maximum of 0.7 to 0.166.

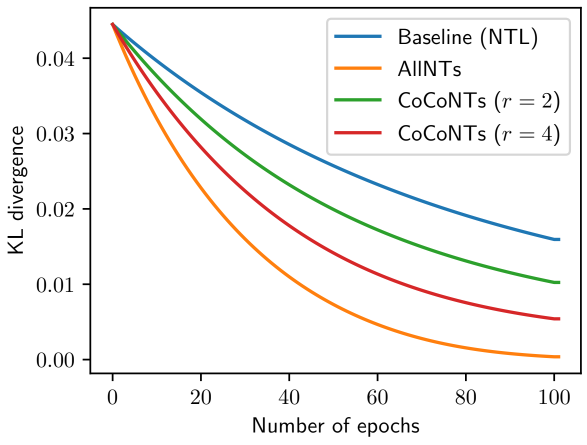

Figure 6: **Studying convergence rates of our approximation when learning a single 10-class multinomial.** The trajectories are averaged over 10 independent runs for all methods. Higher values of r yield better approximations as well as convergence rates.

Effect of r. A crucial hyperparameter in our approximation is r. While we see the effects of r on overall model quality in Sec. 5.4, we study the effect of r in a more controlled way when learning a single 10-class multinomial. By increasing r, as shown in Fig. 6, we find that both convergence rate as well as KL divergence between learned and actual multinomial consistently improves.

A.3 Additional Discussion on Pre-enriching the Dataset

In this section, we provide a step-by-step walkthrough of Pre-enriching process for the dataset.

First, we assume that the prefix-trie is already created and available in-memory for $k = 2$. This trie has a crucial property: the "count" attribute (64 bit unsigned integer in our implementation) of each node N_i indicates how many times the prefix, which corresponds to the path from the root to N_i, appears in the training dataset $\mathcal{D}_{\text{train}}$. The root node stores the count of total prefixes in the dataset. Figure 7 shows an example trie, where the node associated with the token "United" has a count of 1000, while the root node has a count of 40000. This means that there are 1000 sequences in total

[3] https://huggingface.co/gpt2
[4] https://huggingface.co/EleutherAI/gpt-neo-125m
[5] https://huggingface.co/facebook/opt-125m
[6] https://huggingface.co/facebook/opt-1.3B
[7] https://github.com/babylm/evaluation-pipeline

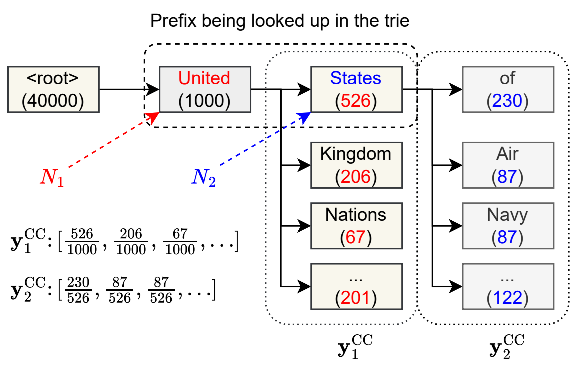
Prefix being looked up in the trie
<root> (40000)
United (1000)
States (526)
Kingdom (206)
Nations (67)
... (201)
of (230)
Air (87)
Navy (87)
... (122)
N_1
N_2
$\mathbf{y}_1^{\text{CC}}: [\frac{526}{1000}, \frac{206}{1000}, \frac{67}{1000}, \dots]$
$\mathbf{y}_2^{\text{CC}}: [\frac{230}{526}, \frac{87}{526}, \frac{87}{526}, \dots]$
$\mathbf{y}_1^{\text{CC}}$
$\mathbf{y}_2^{\text{CC}}$

Figure 7: **Example of trie lookup for prefix "United States".** The number in the paranthesis denotes the "count" property of the TrieNode as described earlier. Notice the denominator terms of $\mathbf{y}_1^{\text{CC}}, \mathbf{y}_2^{\text{CC}}$ carefully. These will be truncated to only include Top-r values.

that begin with the word "United". Additionally, the child node "States" has a count of 526, indicating that there are 526 sequences that start with "United States".

Let's assume that a block (length L) beginning with "United States of America . . . " is selected in the pre-enriching second pass over the dataset. Since $k = 2$, we retrieve from the prefix trie with prefix "United States" as shown in Figure 7. At every level (i.e. highlighted N_i), first $\mathbf{y}_i^{\text{all}}$ will be created in floating point representation with the same bitwidth as token IDs. In our case, this was fp16. Then, we sort the distribution to get top r token IDs and top r probability values. Storing these on disk will require space equivalent to $2kr$ more tokens.

After a block of L tokens is read from the input file, it is immediately written as is to the output file. Then we find top r token IDs and probabilities for each k and sequentially write these values to the output file. After all k such distributions are written, we would have written equivalent of $L+2kr$ tokens to the output file.

A.4 Additional Discussion on Minibatch building with $\mathbf{y}_i^{\text{CC}}$

Since we know that the maximum support of all the $\mathbf{y}_i^{\text{CC}}$ distributions is $r + 1$, we can ideally easily pass them as key-value pairs to the training loop and calculate KL divergence more efficiently. This causes only $O(kr)$ increase in memory footprint of a batch. However, this can be slightly inefficient since we need to run "`gather`" operation to obtain correct components of the predicted distribution $P_\theta(y_i)$ based on token indices. "`gather`" operation is often slow[8] on TPU/XLA devices which rely on a predictable dataflow in order to optimize their compute graph. Prior works such as BigBird (Zaheer et al., 2020) have resorted to special resizing and converted the operation to continuous memory access.

Such tricks are harder to implement here without ascertaining an upper bound on the maximum token ID in the support of $\mathbf{y}_i^{\text{CC}}$. Ideally, obtaining such bounds may be useful and possible since the tokenizer (such as WordPiece or BPE) are expected to assign lower token IDs (earlier "merges") to frequent tokens anyway. In our implementation, we initialize a $k \times |\mathcal{V}|$ size zero vector and use the "`scatter`" [9] operation to populate counts at correct places. While the "`scatter`" operation is also slow, we perform it during batch creation on CPU which is latency optimized as opposed to previous proposal which was doing "`gather`" operation on accelerators which are throughput optimized. While this increases the memory footprint of the batch by $O(k|\mathcal{V}|)$, we found that using such dense vectors for KL divergence resulted in the model running slightly faster on both TPUs and GPUs.

Since we pack the $[\mathbf{y}_1^{\text{CC}}, \dots, \mathbf{y}_k^{\text{CC}}]$ directly into the batch as $|\mathcal{V}|$-dimensional vectors, the memory used by the labels in the CoCoNTs objective is considerably higher than in the baseline. Despite this, we occupy only 0.5% of the total TPU/GPU RAM used by the trainer. The baseline (NT) method takes only $128 \times 256 \times 2 = 64$KB of memory to store labels for a sequence length of 256 and a batch size of 128, assuming16-bit integer token IDs. In the CoCoNTs objective, we provide additional k distributions. Assuming each number in the distribution is a 16-bit float, and considering $k = 8$ and GPT2's vocabulary size of 50257, we occupy approximately $128 \times 8 \times 50257 \times 2 = 98.16$MB of additional GPU/TPU RAM per batch during training. On our GPUs, the NT objective utilizes around 72GB of RAM out of the total available 80GB, leaving more than enough room to accommodate all the $k = 8$ extra distributions per sequence per batch.

[8] `https://github.com/pytorch/xla/issues/898`, `https://github.com/pytorch/xla/issues/3587`

[9] `https://pytorch.org/docs/stable/generated/torch.scatter.html`