The Scaling Law for Continual Pre-trained Language Models Across Different Subset Selection Algorithms

Jayden Lim
University of North Carolina Chapel Hill/National University of Singapore

Abstract:

The Neural scaling law, as described in prior literature, asserts that as model size, amount of training data, and compute resources are increased, the model's performance will improve as a result. In this work, we investigate the nature of the scaling law with respect to the amount of available training data. We analyze the robustness of the scaling law as it relates to Large Language Models continually pre-trained on subsets curated with different data sampling algorithms, including random selection, perplexity-based pruning selection, and top-K TracIn scores subset selection. We demonstrate that the presence of the scaling law is ubiquitous across all 3 subset selection techniques, yet it is also contingent on the batch size and how noisy the gradients are. Additionally, we demonstrate an inverse Scaling Law relationship between training and evaluation loss in the context of top-K TracIn subset selection. Moreover, we also show how training using strategic subset selection can contribute to lower loss compared to merely training with a random subset and how predicting the perplexity distribution of the evaluation set can improve model efficacy.

1. Introduction + Background & Methods

1.1 Behavior of the Scaling Law

There is a rich history of the Neural scaling law's application to efficient deep learning. By adding too much data, the model takes too long to train. Conversely, by adding too little data, the loss will be too high for the model to yield an acceptable performance on the downstream task. Hence, by following the scaling law of adding more data to the training set to improve performance, is it possible to locate exactly when the model begins to yield diminishing returns? As shown by the description of the scaling law from Kaplan, Jared, et al. (2020)[1], it is possible to predict the behavior of the evaluation loss as one continues to logarithmically add more data to the training set using the given function:

$$L(D) = (D_c/D)^{\alpha_D}\ ;\ \ \alpha_D \sim 0.095, \quad D_c \sim 5.4 \times 10^{13}\ \text{(tokens)}$$

When fully graphed (Figure 1), we observe that the loss begins to visually converge around 10^{26}. However, after 10^{18}, the loss decreases by less than 0.1 between consecutive dataset sizes.

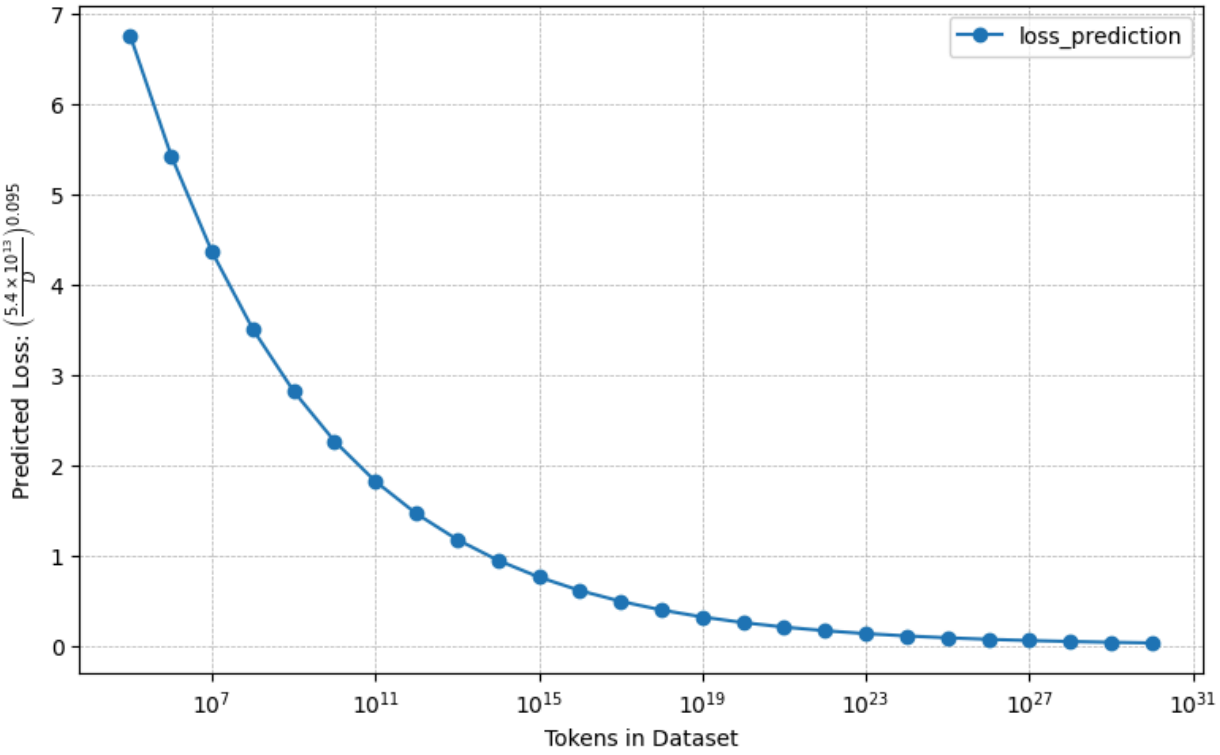

Figure 1.

Given the option of unlimited compute and training time, training to convergence would allow the model to reach optimal performance. Yet, with the limitations of both of those resources, we clearly observe that the precise point to stop adding more tokens depends solely on what loss is deemed "acceptable" given time and compute limits allocated to a given institution. Given our limited resources, we decided to use subsets of the C4 dataset between 10^5 and 10^8 tokens, as this range appears to show significant improvements in loss according to Kaplan, Jared, et al. (2020)[1]. We anticipated this would allow for a thorough demonstration of the scaling law without being overly costly in terms of training time.

1.2 C4 (Common Crawl)

The C4 dataset itself, or "Colossal Clean Crawled Corpus" as introduced by Raffel et al. (2019), is a cleaned version of the Common Crawl dataset known to produce better performance on a variety of NLP tasks than the original Common Crawl dataset[2]. Common Crawl is a giant corpus of web-crawled data. However, even after cleaning, C4 remains a relatively noisy dataset[3]. Additionally, by loading C4 from Hugging Face via the allenai/c4 repository, one should note that this dataset is incredibly large, consisting of 156 billion tokens and 365 million documents. Hence, our implementation utilizes a streaming approach, which avoids having to download the entire dataset and thus optimizes local disk space.

1.3 Implementation & Setup

We implement a Python script that enables each example to be streamed dynamically during the tokenization process using a combination of Hugging Face libraries and PyTorch. During the streaming, we include the option to apply our custom perplexity filter or TracIn filter. We load the pre-trained GPT-2-small, the smallest version of GPT-2 with only 124 million parameters, as it is one of the more lightweight large language models made available by Hugging Face. We then use the Hugging Face Trainer on the pre-trained model to effectively tune all current parameters during each training step and train on 4 NVIDIA L40 GPUs. As observed by Que, Haoran, et al. (2024), the Scaling Law remains just as evident during continual pre-training as in pre-training the model from scratch[4]. Lastly, we log both the training and evaluation cross-entropy loss for each subset size and plot using Matplotlib. For more on our implementation, our code is available on GitHub: https://github.com/jaylim13/data-selection-project0**.**

1.4 Types of Data Selection Algorithms

Random

The entire C4 dataset is shuffled with seed 42, and we take the first n streamed samples.

Perplexity Limited

As each sample is streamed in, we take the perplexity (exponentiated e^{loss}) of each sample using the out-of-the-box pretrained GPT2. We set an upper and lower bound that the perplexity must fall within. If the perplexity falls within this limit, we accept the sample into the dataset; otherwise, we exclude it.

Perplexity Limited AND Bucketed

Run the perplexity function for a large number of samples and plot the distribution. This selection technique is almost the same as the perplexity-limited approach, but we control the proportions of each perplexity by splitting into 3 buckets based on the perplexity

distribution of the training set: low, mid, and high, with more emphasis on the mid bucket to generalize to the majority of examples. We also eliminate outlier examples by limiting the perplexity of the "high" bucket to 400, as our goal with bucketing is to remove or lessen the impact of noisy samples.

TracIn Function

We first implemented a custom callback that computes the gradient for every batch of the trainer and saves the list of gradients as a .pt file in your workspace. We then implemented a custom Trainer that inherits from the Huggingface Trainer so that you can pass the inputs and get the loss for each sample. Next, we use the .pt gradients file to compute the influence scores of each sample and sort the target dataset based on the highest influence scores. Lastly, we retrain the base Hugging Face pre-trained GPT-2 model with the top k % samples and compare the loss to when we solely used a randomly selected subset.

2. Empirical Results

Our initial approach aimed to conserve memory, and hence only trained on a single A100 GPU. Additionally, we set the batch size, gradient_accumulation_steps, and epoch count to 1. We trained with a learning rate of 1e-4. We specifically started the experiment with a low batch size to save GPU compute and generalize better with smaller subsets. However, some trade-offs to consider include slower runtime and potentially noisy gradients. For the perplexity filter, we settled on limiting the perplexities between 50-100 (or ~75-92 percentile) as this showed the lowest training loss among various trials. However, we later observe that using the perplexity range of 22-55 (or ~25-75 percentile) suggested by Marion, Thomas et al.(2023) would produce better performance on the validation set[5]. We run the training workflow for both random and perplexity-pruned datasets of a set number of samples, increasing by 10^k, letting the number of batches = total gradient steps = number of samples to observe any influence on the scaling law (Figure 2).

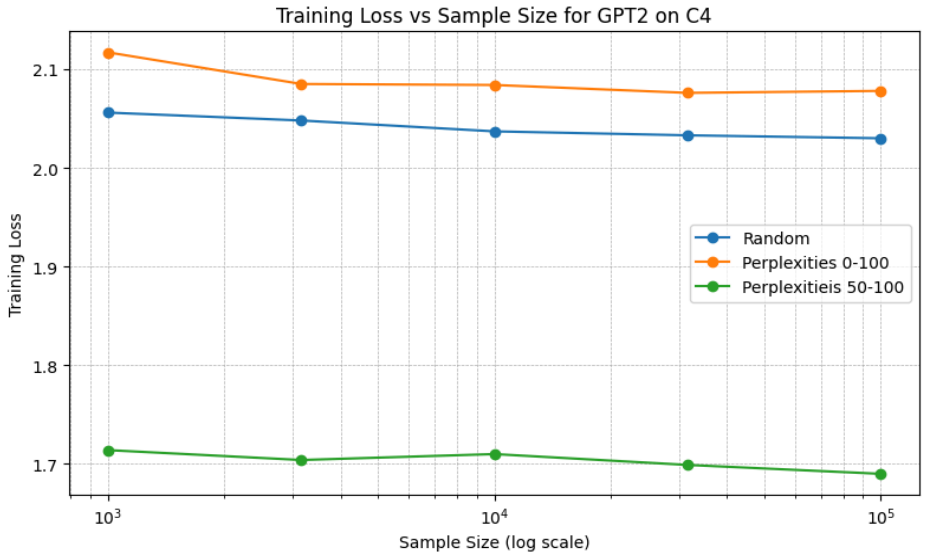

Figure 2.

Following the experiment on the number of samples, we do observe that when adding more samples and using the formula gradient steps = num_epochs x dataset_size / batch_size to let gradient steps equal batch size and data set size, there will be a generally decreasing trend, yet the graph is not guaranteed to decrease in loss at every step. We propose that this is either because the samples vary too much in length or because there is too much noise from a low batch size. Hence, we continue the experiment with a fixed token size to guarantee absolute consistency between each increment of the x-axis:

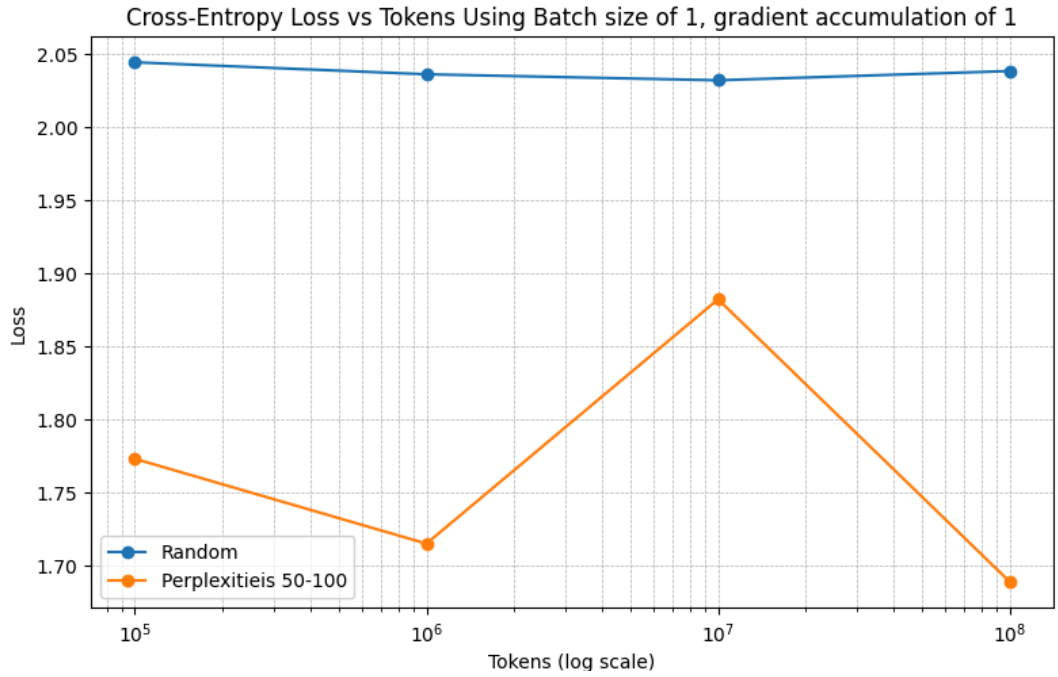

Figure 3.

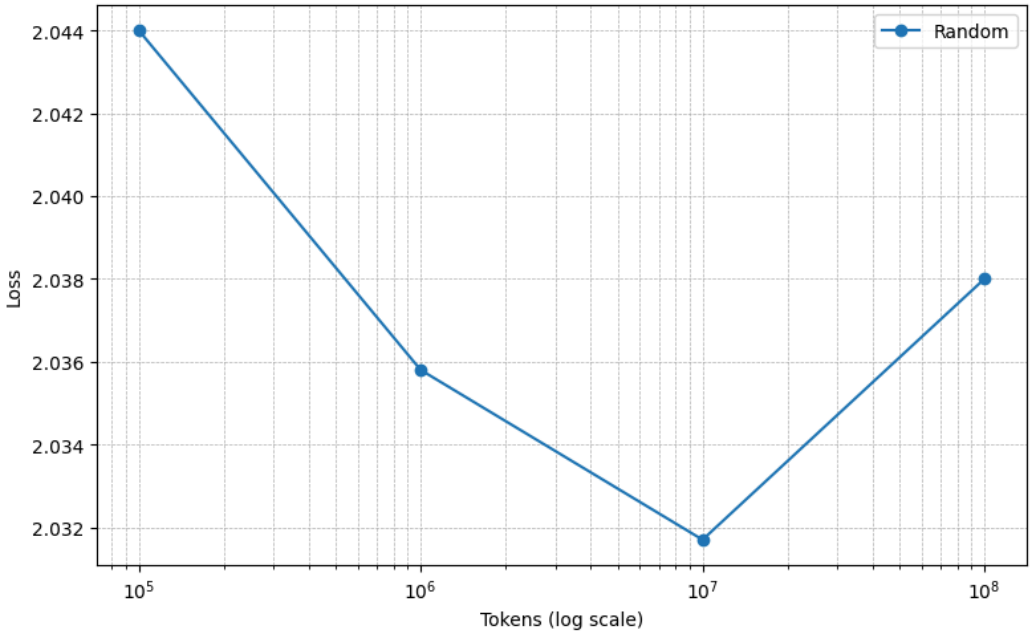

Figure 4.

Given Figures 3 and 4, we see a sudden spike when training with a dataset of 10^7 total tokens, showing that this deviation from the scaling law is likely due to noisy gradients caused by a low batch size. Furthermore, to address the noisy gradient problem, we modified the training setup to work with multiple GPUs (4) and increased the batch to 16 with gradient_accumulation_steps = 2, giving an effective batch size of 32 which is the upper-end optimal batch size according to Masters, Dominic, and Carlo Luschi (2018) and thus resulting in relatively fast training[6].

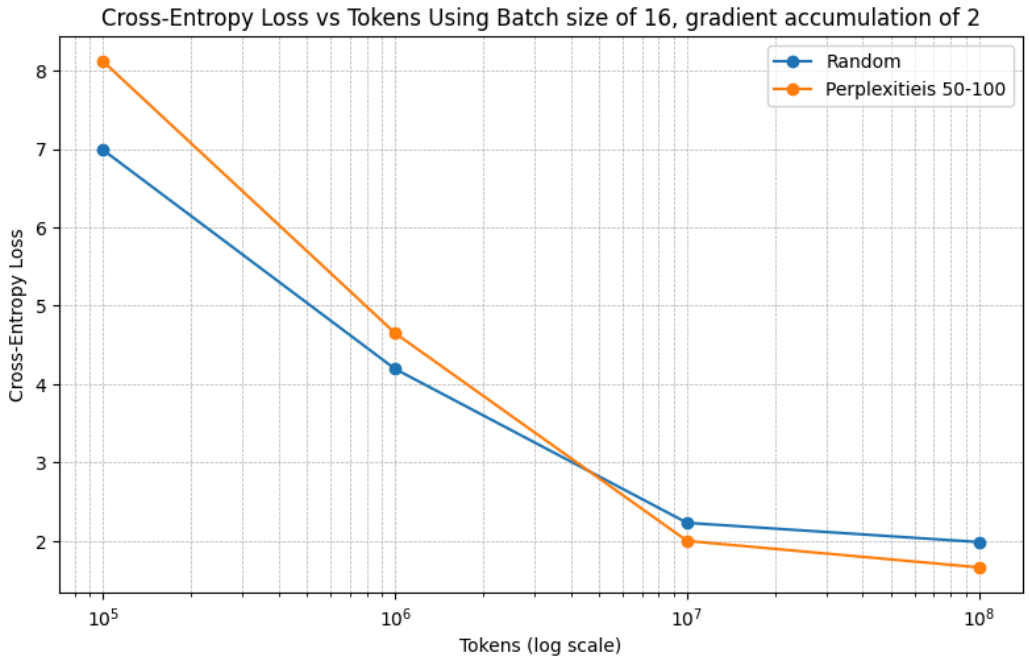

Figure 5.

From Figure 5, we observe that the perplexity-limited model loss being lower than the random selection model loss may imply that as we limit the perplexities of the training data, the model naturally learns quicker since the perplexities are less diverse. Additionally, we see at 10^7 tokens, both models have practically the same loss. Furthermore, perhaps even with less data, the high-perplexity-limited model will perform better on a high-perplexity dataset than a model trained on random data.

On the evaluation data, we see that the random selection model still outperforms the perplexity-limited model in Figure 6.

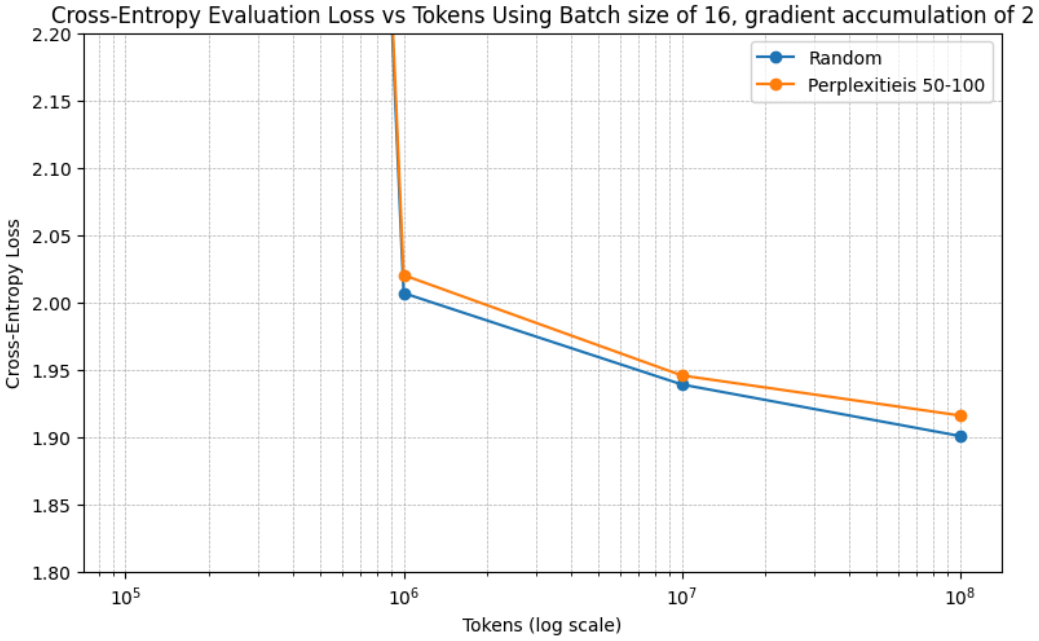

Figure 6.

The discrepancy between the two models means that 50-100 perplexities actually generalize well to the majority of data. However, by not including samples below 50 or above 100 at all, we run the risk of not generalizing at all to the extremely easy examples or extremely hard examples. Under this assumption that models trained on a set perplexity range are good at predicting other samples in that perplexity range, we attempt to get our perplexity training dataset perplexity distribution to match the evaluation set better than the randomly selected training dataset.

Therefore, we implement a perplexity bucketed approach to manually set the perplexity distribution of our training subset and exclude points that appear to be extreme outliers based on the distribution of the entire C4 dataset. The perplexity distribution we used incorporates the suggestion from Marion, Thomas et al.(2023) of 25-75 percentiles for the majority of samples[5]. The buckets of the training subset are as follows:
Low 0.05: perplexity <22,
Mid 0.8: 22 <= perplexity < 55
High 0.15: 55<= perplexity <= 400

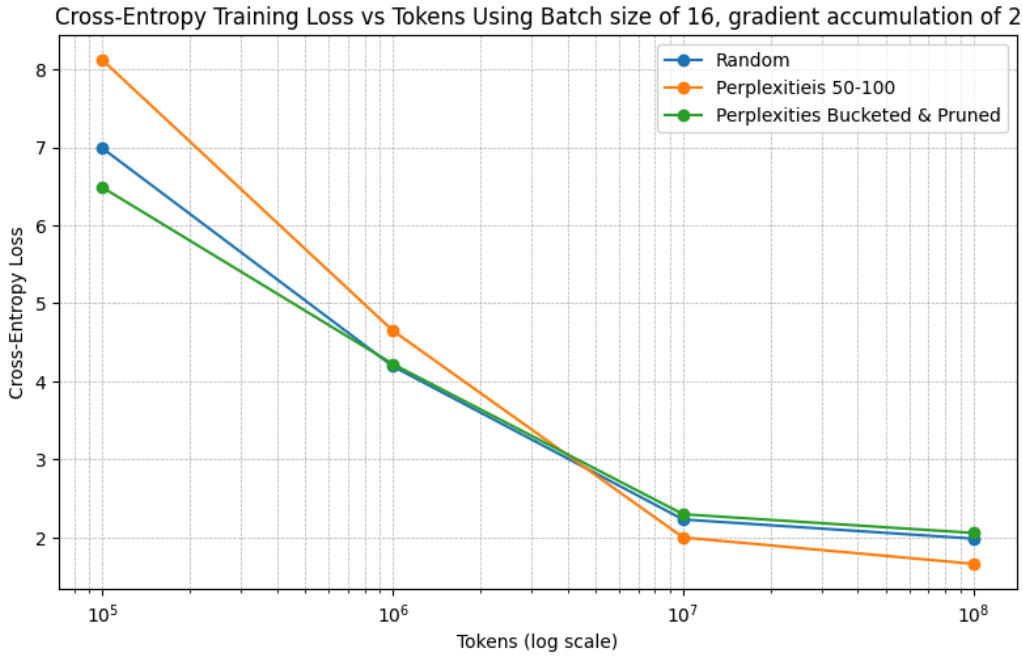

Figure 7.

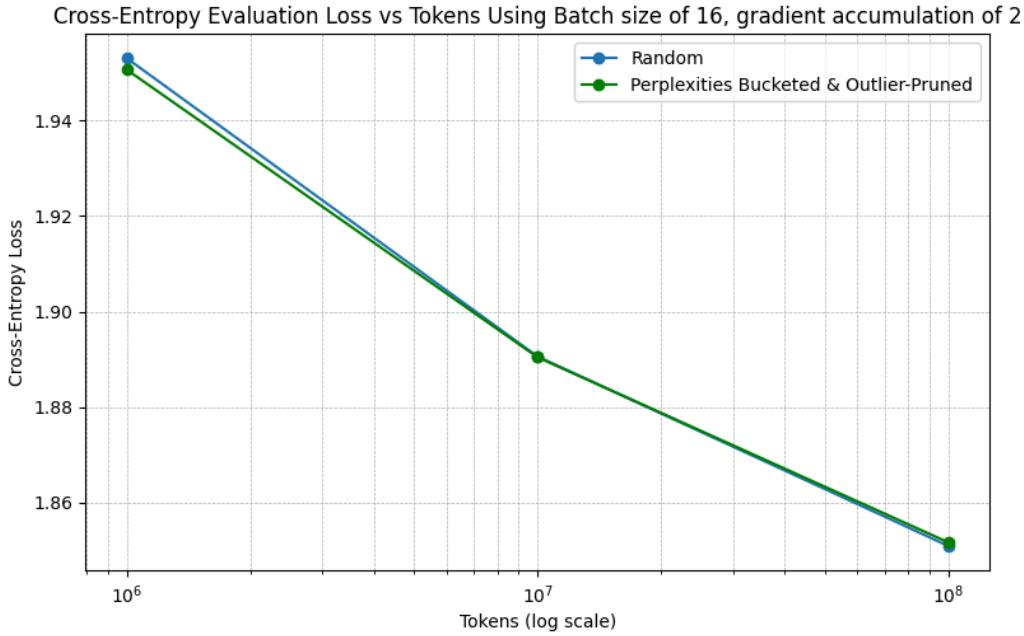

Figure 8.

As seen in Figure 8, the perplexity bucketed model does better with a dataset of 10^6 than the random selection model, but worse at 10^8. As you continue to train the model with more diverse data, not only does the model learn new patterns, but the perplexity distribution of the training set changes as well. Moreover, the perplexity distribution of the random 10^8-token training subset may now be more similar to the test set compared to the

perplexity-bucketed 10^8-token training subset, resulting in better generalization by the random selection model.

Lastly as observed in Figure 9, when all 3 models are evaluated on a test set of perplexities 50-100 (~75-92 percentile), we see that the model limited to those exact perplexities begins to significantly outperform the other two, hence verifying our earlier assumption that models trained on a set perplexity range are good at predicting other samples in that perplexity range.

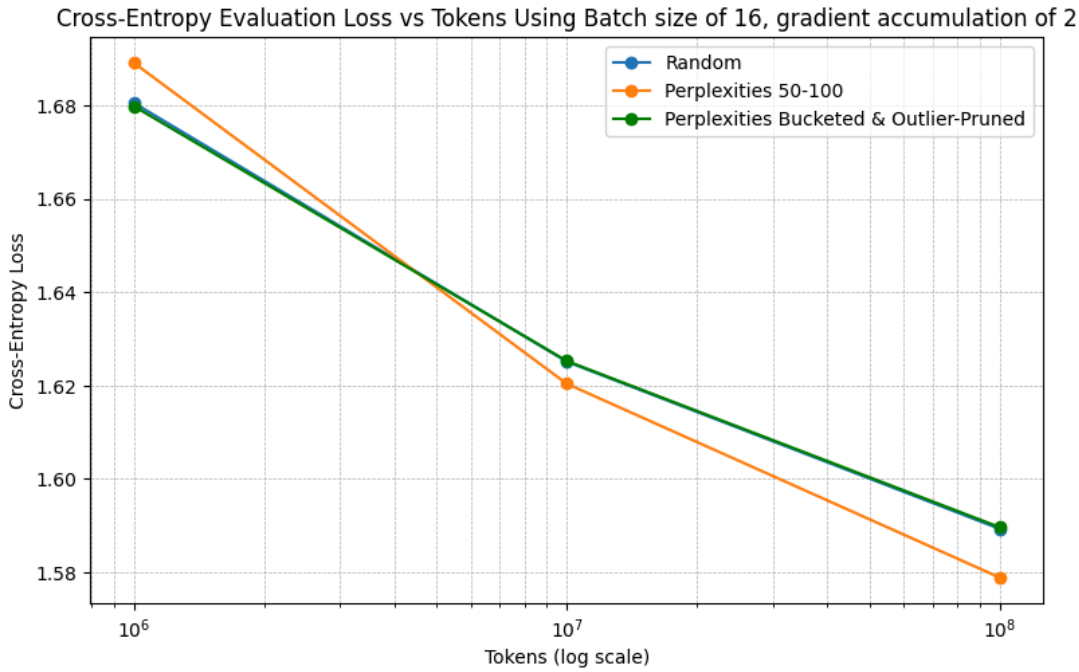

Figure 9.

Tracin

To observe the scaling law on a much more effective gradient-based subset selection technique, we implemented our own custom TracIn callback and trainer and integrated it with our existing Hugging Face workflow. However, due to the size of the gradient .pt file, we were only able to run the TracIn procedure for a small subset (10^5tokens).

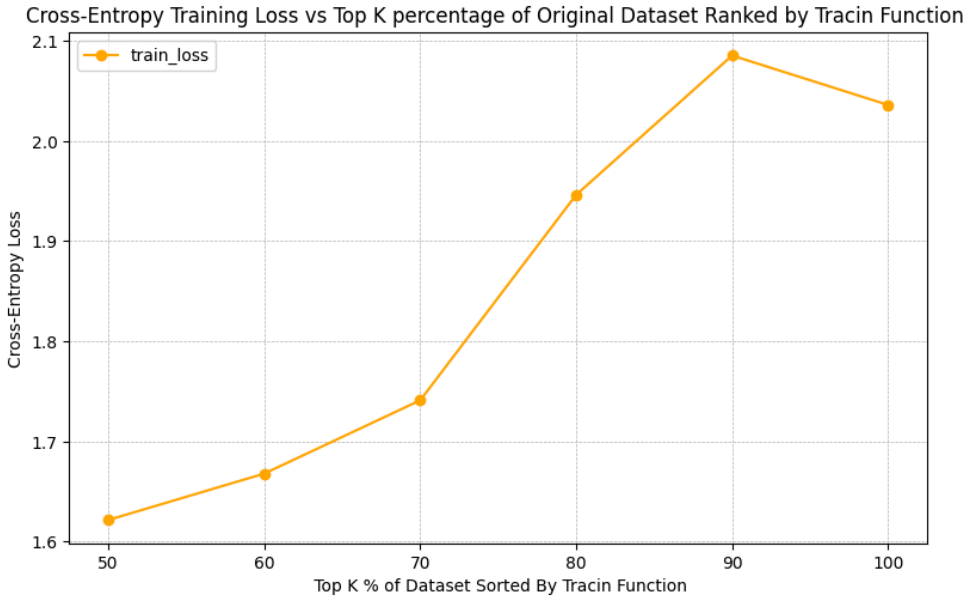

Figure 10.

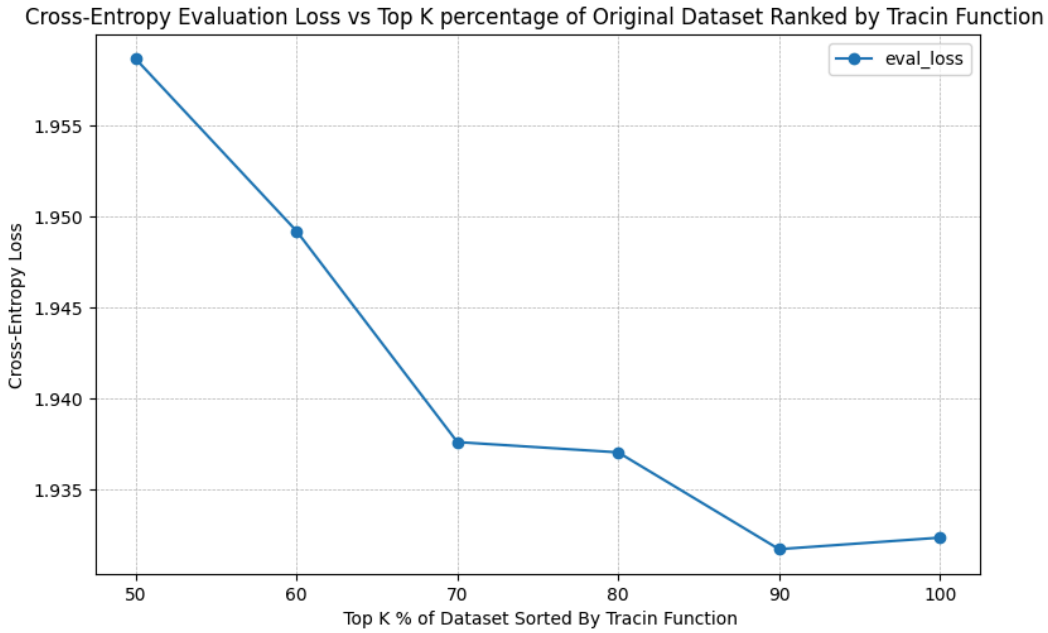

Figure 11.

At 90% of the Tracin ranked dataset, the loss finally goes below the original dataset size, showing that we can use the influence function to get a better loss for the model. The training loss has an inverse relationship with the evaluation loss since the dataset we are sampling from is sorted based on higher influence scores. By adding more samples from this sorted dataset into our training set, we are adding more samples with lower influence

scores, and hence, the training loss increases. However, adding more samples gives us a better validation loss as the model becomes able to generalize to a higher diversity of examples.

3. Future Works

As described by Kaplan, Jared, et al. (2020), there exists an additional loss prediction function with respect to both dataset size and number of parameters [1]. It would be beneficial to run the training and testing on both larger and smaller models on varying batch sizes to confirm if deviations from the scaling law are truly due to noise or can be overcome with additional parameters. Another point to consider for future work is running more trials with different shuffling seeds and comparing the plotted behavior of the loss across these different seed shufflings. This would enable us to see if the observations are consistent across diverse training orders and a greater variety of different examples from the C4 dataset.

4. Conclusion

We empirically validate that the Neural scaling law is evident throughout all training procedures, regardless of whether there was a special subset selection algorithm incorporated into the workflow, with the exception of cases where low batch sizes contribute to extremely noisy gradients. In the case of deciding what effective batch size to use, one should consider that although low batch sizes tend to generalize better and save more GPU memory, they may also jeopardize the scaling law.

References:

[1] Kaplan, Jared, et al. *Scaling Laws for Neural Language Models*. arXiv preprint arXiv:2001.08361, 2020. https://arxiv.org/abs/2001.08361

[2] Raffel, Colin, et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. arXiv preprint arXiv:1910.10683, 2020. https://arxiv.org/abs/1910.10683

[3] Maini, Pratyush, et al. *Rephrasing the Web: A Recipe for Compute and Data-Efficient Language Modeling*. Apple Machine Learning Research, August 2024. https://machinelearning.apple.com/research/recipe-for-compute

[4] Que, Haoran, et al. *D-CPT Law: Domain-specific Continual Pre-Training Scaling Law for Large Language Models*. Advances in Neural Information Processing Systems, 2024. arXiv preprint arXiv:2406.01375. https://arxiv.org/abs/2406.01375

[5] Marion, Thomas, et al. *When Less is More: Investigating Data Pruning for Pretraining LLMs at Scale*. arXiv preprint arXiv:2309.04564, 2023. https://arxiv.org/abs/2309.04564

[6] Masters, Dominic, and Carlo Luschi. *Revisiting Small Batch Training for Deep Neural Networks*. arXiv preprint arXiv:1804.07612, 2018. https://arxiv.org/abs/1804.07612