Worth the Weight? Entropy adjustment may not improve performance of Code LLM fine-tunes that use a KL component

Kyle Domingos
domingos.k@northeastern.edu

Luna Phipps-Costin
phipps-costin.l@northeastern.edu

Abstract

Catastrophic forgetting (McCloskey and Cohen, 1989), the abrupt loss of previously acquired knowledge when a model is trained on new tasks, has been extensively documented in neural networks during supervised fine-tuning. We seek to address this in the context of large language models for generating code (code-LLMs). We investigate the application of a Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) loss term for this setting. We also investigate the use of an entropy-based KL weight proposed by (Su et al., 2023) and compare the results. Our study systematically varies the weight of the KL term and the presence and sign of an entropy-based weighting strategy based on EA-KD. We conduct experiments on the OSS-Self-Instruct dataset with StarCoder2-3B.

1 Introduction

Supervised fine-tuning (SFT) can add capabilities and improve performance of large language models that generate code (code-LLMs). However, such fine-tuning can also degrade performance on certain code generation tasks (1989). Some remedies to this problem include adding a Kullback-Leibler (KL) divergence loss term (1951) and more recent techniques such as EA-KD proposed by Su et al. (2023). In this work we seek to determine which of these strategies best preserve or improve performance in fine-tuning small code LLMs.

2 Background and Related Work

2.1 Key Paper: Catastrophic forgetting

Catastrophic Interference (or Catastrophic Forgetting as it is interchangeably known as) is the main concept put forth in the seminal paper (1989). Human cognition is generally understood to be a sequential process. From basic building blocks, we add additional knowledge and build upon concepts in order to construct better understanding. The connectionist model at the time hypothesized that computers could be made to learn in much the same way; namely by giving examples and training up knowledge sequentially. The paper (1989) dealt a serious blow to the existing connectionist model of cognition, by showing the degrading performance of standard Neural Networks on its trained task after further training on an adjacent task.

2.2 Distillation loss / KL terms

Kullback-Leibler divergence is an effective measure of how the output distribution of a fine-tuned student model differs from a base teacher model. It penalizes the fine-tuned model for diverging too far from the original base model prediction, which reduces the risk of catastrophic forgetting.

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log \left(\frac{P(x)}{Q(x)} \right) \quad (1)$$

Where $P(x)$ is the probability assigned to token x by the teacher model, and $Q(x)$ is the probability assigned to token x by the student model.

2.3 Key Paper: Entropy-weighted KL

Standard Knowledge distillation has an issue where all samples are given the same weight, which has the unfortunate side effect of prioritizing low entropy samples. This in turn, means that any useful data like learning values are completely ignored. Entropy based Adaptive Knowledge Distillation (EA-KD) as proposed in (Su et al., 2023) is a solution to this that utilizes the entropy from the model to weight the samples, by giving the high entropy samples higher weight, and lowering the weight of low entropy samples. This simple concept results in a plug-and-play solution applicable wherever KD is already being used that gives SOTA improvements with negligible additional computational costs.

$$H = -\sum p_{n,j}\left(T'\right) log(p_{n,j}\left(T'\right)) \quad (2)$$

2.4 Pass@K

It is a relatively recent innovation to evaluate models by their success on a test suite rather than their confidence in predicting the next token. (Sumith Kulal, 2019) developed the success rate at B metric, which was improved and popularized by the Codex paper as Pass@k.

$$\text{pass@k} = \frac{1}{N}\sum_{i=1}^{N}\left[1 - \frac{\binom{n_i - c_i}{k}}{\binom{n_i}{k}}\right] \tag{3}$$

2.5 StarCoder 2

StarCoder2 (Lozhkov et al., 2024) is an open-access family of large language models (LLMs) for code, developed by the BigCode community in collaboration with ServiceNow, Hugging Face, and NVIDIA. Trained on The Stack v2 and some natural language such as Wikipedia, Arxiv, and GitHub issues, it encompasses over 600 programming languages. StarCoder2 models range from 3B to 15B parameters. We use the 3B model.

2.6 OSS-Instruct

It is popular to improve the performance of large language models (LLMs) for code by supervised fine-tuning (SFT). Usually, we generate small datasets of "high-quality" code synthetically, and then utilize these datasets to adjust the weights of a base model that was trained on a larger, more noisy dataset. It is also useful to fine-tune models to complete question-answer pairs rather than raw text (Ouyang et al., 2022), commonly known as instruction tuning.

For our experiments we use one such dataset, OSS Self-Instruct (Wei et al., 2024). In particular, this dataset has question-answer prompts generated by StarCoder 2 which have been sourced, prompt-engineered, and validated by running in order to ensure high quality.

2.7 BigCodeBench

BigCodeBench is a code benchmark designed for solving practical and difficult programming questions. Built by BigCode, an open-sourced scientific collaboration dedicated to developing code-LLMs, it differs from other benchmarks like HumanEval by focusing on multi-step problems more akin to what you would see deployed in real life instead of short function completion. BigCodeBench-Hard is a subset of the benchmark with only the hardest problems; with 148 problems it is both more practical and resistant to saturation.

2.8 Contributions

We evaluate a variation of EA-KD for supervised fine-tuning on the OSS Self-Instruct dataset and StarCoder 2-3B. In particular we seek to answer these questions:

RQ1: *What is a good constant factor for a KL term (α hyperparameter) when fine-tuning small Code LLMs?* We determine that the KL term is highly sensitive to its scale, with all tested values but one ($\alpha = 0.2$) decreasing performance.

RQ2: *Can an entropy-weighted KL term like EA-KD further improve performance in Python code generation?* We find that our formulation of entropy-weighted KL has no impact on performance for our use case: neither positively nor negatively weighting based on entropy has a significant effect on performance for Python code generation.

3 Experimental setup

We train StarCoder 2 (3B) on the OSS Self-Instruct dataset filtered by executions with 50k question-answer pairs (BigCode, 2024). We use hyperparameters matching those used in the fine-tune by Cassano et al. (2024) and train for 2 epochs per model.

We adjust our loss StarCoder 2 loss to answer our research questions. We define our loss function $\mathcal{L}$ as a standard distillation loss (Hinton et al., 2015) with a CE and KL term, weighted by some value.

$$\mathcal{L}_{\text{ours}} = (1 - \alpha_i) \cdot \mathcal{L}_{\text{CE}} + \alpha_i \cdot \mathcal{L}_{\text{KD}} \tag{4}$$

For the weight α_i, we use a modification of EA-KD, parameterized by a constant term α and a sign term σ. The case of $\sigma = 0$ represents a standard constant-weighted KL term.

$$\alpha_i = \begin{cases} \alpha \cdot \left(1 - \frac{H_i}{H_{\max}}\right), & (\sigma \text{ positive}) \\ \alpha \cdot \frac{H_i}{H_{\max}}, & (\sigma \text{ negative}) \\ \alpha & (\sigma \text{ zero}) \end{cases} \tag{5}$$

We sample each model 64 times per problem on the BigCodeBench-Hard benchmark.

4 Results

Figure 1 shows the performance of each of our models on BigCodeBench-Hard. It is clear that there is no clear linear relationship for entropy, which has the opposite effect when $\alpha = 0.2$ as it does when $\alpha = 0.5$. It is clear that performance is sensitive to the choice of α; in our experiments $\alpha = 0.2$ performed better than any others.

α	σ	avg pass@1
0	zero	0.0351
.05	zero	0.0321
0.1	zero	0.0291
0.2	negative	0.0324
0.2	zero	**0.0439**
0.2	positive	0.0356
0.5	negative	0.0363
0.5	zero	*0.0264*
0.5	positive	0.0269

Figure 1: Performance for all our trained models on BigCodeBench-Hard. Best performance is **bold**, worst performance is *italic*.

4.1 A KL component can decrease forgetting, but also impair learning

Figure 1 shows that a fine-tuning loss with a constant-weight KL term is quite sensitive to the α hyperparameter. While the KL component may reduce forgetting, it can also impair learning during fine-tuning. At $\alpha \leq .1$, catastrophic forgetting seems to still a significant problem, while at $\alpha = .5$, learning may be significantly impaired.

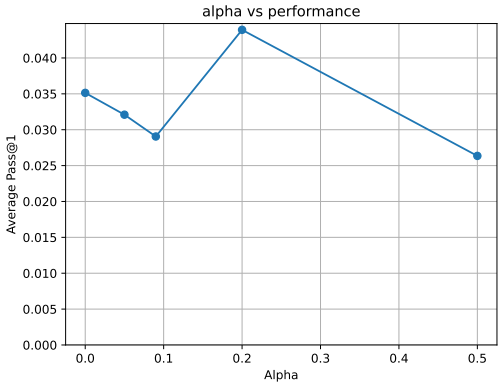

Figure 2: Increasing constant KL weight decreases performance except for $\alpha = .2$

4.2 Constant factor α better explains performance than entropy weight σ

We evaluated whether our observed model performances are better explained by the addition of the entropy-weighted term, the constant KL weight α, or neither. We fit our results to three least-squares regressions: one with entropy adjustment as a linear effect, one with α as a quadratic effect, and one combining both. Only α had a significant effect ($p = 0.007$). Entropy adjustment failed to explain performance with or without α included; its correlation was $R^2 = -0.001$. An ANOVA (Analysis of Variance) test comparing the entropy adjustment only model and the α model yielded $p = 2 \times 10^{-189}$, confirming that the constant factor of the KL component of the loss is the main driver of performance in our experiments.

We modeled entropy adjustment as a linear effect with value -1 for σ negative, 0 for no entropy adjustment, or 1 for σ positive, assuming that enforcing confidence and anti-enforcing confidence should have an opposite effect.

We modeled α as a quadratic effect because our results on the KL term indicated that too high of an α impairs learning, but too low encourages forgetting.

5 Conclusion

We have shown that in our specific setting, a constant KL term weighted by $\alpha = 0.2$ results in the best performance on our chosen benchmark, with a stastistically significant improvement over other choices of α. We've similarly shown that the use of entropy weighting in either direction has no significant impact on performance.

5.1 Threats to validity

Such a strong result for $\alpha = 0.2$ and the response seeming cubic in nature (Figure 2) calls for replication and a close check that the models were trained fairly. Additionally, our statistical analysis assumes a quadratic KL effect, yet it seems to be cubic in nature. As with most LLM research, it remains to be seen if these effects remain the same in larger models.

One important thing to note is that entropy-adjusting the KL weight adjusts the average weight of the KL component in general. Due to the large impact of said weight, any impacts from the entropy adjustment itself may be dwarfed by its impacts on general KL scale.

It would be valuable to measure the variance of the same model on the same benchmark with 64 samples. Though in theory it should produce a stable Pass@1, the variance in the results suggests it is worth investigation.

5.2 Code and Data

All of our code and data is available on GitHub at `https://github.com/zepdos/CS7150FinalProject-worth-the-weight`

References

BigCode. 2024. Self-oss-instruct starcoder2 dataset (execution-filtered, 50k examples). https://huggingface.co/datasets/bigcode/self-oss-instruct-sc2-exec-filter-50k. Accessed: 2025-04-21.

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. 2024. Knowledge transfer from high-resource to low-resource programming languages for code llms. *Preprint*, arXiv:2308.09895.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *Preprint*, arXiv:1503.02531.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, and 1 others. 2024. Starcoder2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Michael McCloskey and Neal J Cohen. 1989. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Chi-Ping Su, Ching-Hsun Tseng, Bin Pu, Lei Zhao, Zhuangzhuang Chen, and Shin-Jye Lee. 2023. Ea-kd: Entropy-based adaptive knowledge distillation. *arXiv preprint arXiv:2311.13621*.

Kartik Chandra Oded Padon Alex Aiken Percy Liang Sumith Kulal, Panupong Pasupat. 2019. Spoc: Search-based pseudocode to code. *CoRR*, abs/1906.04908.

Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Zachary Mueller, Harm de Vries, Leandro von Werra, Arjun Guha, and Lingming Zhang. 2024. Selfcodealign: Self-alignment for code generation. *Preprint*, arXiv:2410.24198.