

Sparse ReFT

Stanford CS224N Default Project

Jacob Householder

Department of Computer Science
Stanford University
jdhouse@stanford.edu

Abstract

We propose combining Hard-Concrete Binary gates and L_0 regularization [1] with ReFT [2] to induce sparsity over the applied interventions. This allows gradient descent to learn relevant intervention locations rather than treating them as hyperparameters. In our small scale experiments this results in 53.5% to 55% smaller interventions with minimal impact to task accuracy on two benchmarks. To demonstrate that our proposed method is viable and stable we evaluate relevant baselines, ablations, and a Pareto sweep over regularization strength.

1 Key Information to include

- Mentor: Tristan Thrush
- External Collaborators (if you have any): na
- Sharing project: No
- Late Days (if you are using any): 3

Team member	Late days remaining
Jacob Householder	3
Total pooled	0
Team size	1
Late days applied for the report	3

2 Introduction

ReFT is a framework for creating efficient task specific representations to steer LLM behavior. It exists at the intersection of efficient finetuning and interpretability. Transformer based NLP models embed semantics in their hidden states as they process inputs. Researchers have shown that careful edits of intermediate activations can cause desired downstream behavior. In this representation editing paradigm practitioners tune the representations rather than the generating model weights. ReFT is a family of these edits that enables light weight model customization, effectively, it enables a static large base model with small representation edits inserted at key points.

3 Related Work

Recently there have been many lightweight techniques to allow for efficient finetuning LLMs. They take advantage of the insight that the large models often have general latent representations and can be quickly adapted to a specific task with a small number of parameters. ReFT enables LLM steering towards a finetuning objective without updating lots of model weights. Current SOTA parameter efficient fine tuning methods (PEFTs, like Adapters [3, 4, 5, 6, 7], LoRA [8, 9], and prompt based methods [10]) involve either updating a large amount of model weights or adding a large amount of new structure compared to ReFT. The authors base their work on the idea that LLMs may already have

the ability to solve the desired downstream task, but need small targeted interventions to consistently perform the desired behavior [11, 12, 13].

Notable sparse LoRA methods like SparseAdapter [6] apply score based pruning. In contrast, we jointly learn the task and sparsity as we use an end-to-end differentiable mechanism.

4 Approach

ReFT provides a mechanism for small, targeted controllable changes to LLMs that are interpretable. We combine this with L_0 based sparsity in a natural manner to yield real parameter savings.

4.1 Introduction to interventions

We formalize an intervention I is a tuple of intervention function, set of token positions, and layer.

$$I = \langle \Phi, P, L \rangle$$

Interventions are applied to the output of the l -th layer during the forward pass as follows:

$$h^{(l)} = (\Phi(h_p^{(l)}) \text{ if } p \in P \text{ else } h_p^{(l)})_{p \in 1, \dots, n}$$

In the context of ReFT, the model is frozen during fine tuning and the intervention function Φ is parameterized by learned parameters ϕ . Position and layer locations for interventions are chosen as hyperparameters, concretely, they are fixed per training run. Training now only updates these interventions.

4.2 DiReFT

ReFT proposes using a small number of lightweight interchange interventions to finetune LLMs. The authors propose a framework where model behavior is modified by perturbing the hidden states rather than the weights. The authors propose two low rank variants of representation fine tuning: low rank linear subspace ReFT (LoReFT) and direct ReFT (DiReFT). LoReFT constrains the intervention Φ to

$$\Phi(h) = h + R^\top (Wh + b - Rh)$$

with learned parameters $\phi = \{R, W, b\}$. Note that $R \in \mathbb{R}^{r \times d}$ (st $r \leq d$) is a low rank matrix with orthonormal rows. DiReFT removes the orthogonality constraint and results in an intervention Φ as

$$\Phi(h) = h + W_2^\top (W_1 h + b)$$

The experiments suggest that these small targeted changes are often enough to perform well on a variety of downstream tasks.

We make use of DiReFT in our experiments for simplicity. Further, we use a unique intervention per layer and position within our prefix and suffix ranges. We believe that it would be worth considering evaluating LoReFT in our experiment setup.

4.3 Sparse gating via Hard-Concrete L_0 regularization

Our goal is to automatically find a minimal set of non-redundant interventions. This can naturally be posed as a sparsity problem where we penalize the model for every intervention that it uses. We formalize this idea with L_0 regularization.

The L_0 norm is the number of non-zero elements in a tensor. This norm is non-differentiable and has a combinatorial search space, thus directly computing it is intractable. There have been many works that have built up theory about how we can optimize this objective with differentiable surrogates.

We make use of a differentiable surrogate introduced by Louizos et al [1]. The high level goal of the technique is to introduce differentiable binary gates that lead to true sparsity rather than small weights like L_1 and L_2 regularization. The core idea is to make each parameter a product of a binary gate and the underlying parameter. The technique then replaces the non-differentiable L_0 norm with a differentiable expectation over stochastic binary gates.

$$J(\theta) = \frac{1}{N} \sum_i \mathcal{L}(h(x_i; \theta), y_i) + \lambda \|\theta\|_0, \quad \|\theta\|_0 = \sum_{j=1}^{|\theta|} \mathbb{I}[\theta_j \neq 0] \quad \theta^* = \arg \min_{\theta} \{J(\theta)\}$$

Now, consider the reparameterization of θ ,

$$\theta_j = z_j \tilde{\theta}_j, \quad z_j \in \{0, 1\}, \quad \tilde{\theta}_j \neq 0, \quad \|\theta\|_0 = \sum_{j=1}^{|\theta|} z_j$$

Now, we can treat each z_j as a sample from a Bernoulli distribution with parameter π_j , $z_j \sim \text{Bernoulli}(\pi_j)$, then the objective is the expectation over $q(z|\pi)$ the distribution of gates,

$$J(\tilde{\theta}, \pi) = \mathbb{E}_{q(z|\pi)} \left[\frac{1}{N} \sum \mathcal{L}(h(x_i; \tilde{\theta} \odot \mathbf{z}), y_i) \right] + \lambda \sum_{j=1}^{|\theta|} \pi_j \quad \tilde{\theta}^*, \pi^* = \arg \min_{\tilde{\theta}, \pi} \{J(\theta)\}$$

But, backpropagation does not work through Bernoulli variables (as they are discrete). So a continuous relaxation is employed.

Now, for a continuous random variable $s \sim q(s|\phi)$ with parameters ϕ , we can let the gates $\mathbf{z}$ be the hard sigmoid rectification of s .

$$\mathbf{z} = \min(1, \max(0, s))$$

Now, we can compute the probability that the gate $\mathbf{z}$ is non zero (active) as

$$q(\mathbf{z} \neq 0|\phi) = 1 - Q(s \leq 0|\phi)$$

given CDF $Q(\cdot)$ of s . This can now be used as a smooth surrogate as follows:

$$J(\theta, \phi) = \mathbb{E}_{q(s|\phi)} \left[\frac{1}{N} \sum_{i=1}^N \mathcal{L}(h(x_i; \tilde{\theta} \odot g(s)), y_i) \right] + \lambda \sum_{j=1}^{|\theta|} (1 - Q(s \leq 0|\phi))$$

Now, through the reparameterization trick [14] we can express the objective as an expectation over the noise distribution $p(\epsilon)$ and deterministic differentiable transform f of parameters ϕ and ϵ

$$J(\theta, \phi) = \mathbb{E}_{p(\epsilon)} \left[\frac{1}{N} \sum_{i=1}^N \mathcal{L}(h(x_i; \tilde{\theta} \odot g(f(\phi, \epsilon))), y_i) \right] + \lambda \sum_{j=1}^{|\theta|} (1 - Q(s_j \leq 0|\phi_j))$$

So, we can make the following Monte Carlo approximation

$$J(\theta, \phi) = \frac{1}{L} \sum_{l=1}^L \left(\frac{1}{N} \sum_{i=1}^N \mathcal{L}(h(x_i; \tilde{\theta} \odot \mathbf{z}), y_i) \right) + \lambda \sum_{j=1}^{|\theta|} (1 - Q(s_j \leq 0|\phi_j))$$

where

$$\mathbf{z} = g(f(\phi, \epsilon)) \quad \text{and} \quad \epsilon \sim p(\epsilon).$$

We will directly make use of this result. We will build our gates as PyTorch modules that do not take any input and are purely functions of their parameters ϕ and noise distribution $p(\epsilon)$.

4.4 Smoothing distribution: Binary Concrete

Given the previously derived objective, we make use of a binary concrete random variable s in the interval $(0, 1)$ with PDF $q(s|\phi)$ and CDF $Q(s|\phi)$. The parameters for the distribution are $\phi = (\log \alpha, \beta)$, where $\log \alpha$ is location and β is the temperature. The distribution is then stretched to the (γ, ζ) interval with $\gamma < 0$ and $\zeta > 1$. Finally, the hard sigmoid is applied on its random samples. Critically, this approximates the Bernoulli distribution and saturates to exactly 0 or 1 to enable true sparsity.

$$u \sim \mathcal{U}(0, 1), \quad s = \text{sigmoid}((\log u - \log(1 - u) + \log \alpha)/\beta), \quad \bar{s} = s(\zeta - \gamma) + \gamma$$

$$z = \min(1, \max(0, \bar{s}))$$

We directly implement the above equations in PyTorch for the training pass.

Now, the regularization penalty can be expressed as follows

$$\lambda \sum ||\theta||_0 = \lambda \sum_{j=1}^{|\theta|} (1 - Q(0|\phi)) = \lambda \sum_{j=1}^{|\theta|} \text{sigmoid}(\log \alpha_j - \beta \log \frac{-\gamma}{\zeta}).$$

At evaluation time the gates are evaluated deterministically:

$$\hat{z} = \min(1, \max(0, \text{sigmoid}(\log \alpha)(\zeta - \gamma) + \gamma)), \quad \theta^* = \tilde{\theta} \odot \hat{z}$$

We make use of these gates in a "group sparse" fashion where we turn off entire interventions by multiplying one gate by the entire output of an intervention at a given layer and position combination.

We manually implement ReFT intervention modules. Naturally PyTorch decomposes models into layers, so we create one module that adds interventions at positions to a layer. We modify GPT-2 to allow for these optional module per layer. We add a routine to initialize the interventions after the model is loaded, this must be run before passing parameters to the optimizer. The user can configure which layers and positions have interventions. For efficient batching we only support dense position ranges, but it is straightforward to extend to sparse positions per layer.

We create another intervention module that combines the binary gating mechanism with dense ReFT. These two methods combine rather nicely; we simply multiply each intervention (position, layer) by its gate and the gradients flow.

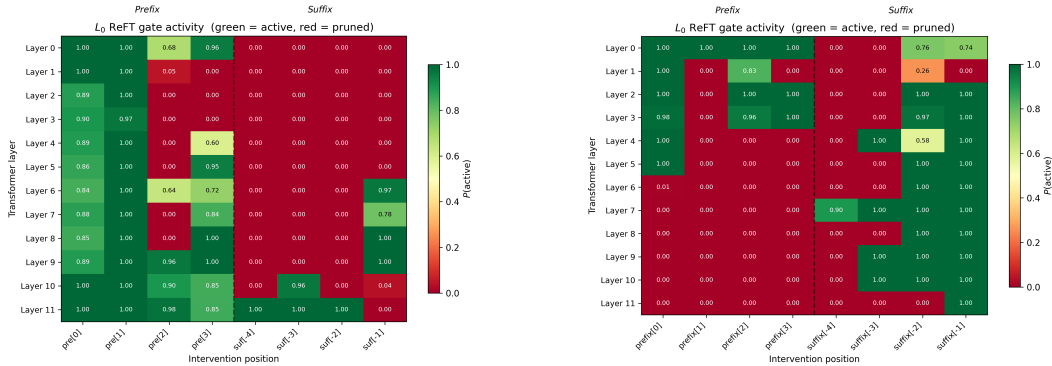
In our configuration we apply interventions to the complete residual stream, so after the FFN in the GPT-2 layer.

$$\begin{aligned} x &= x + \text{Dropout}(\text{Attn}(\text{LayerNorm}(x))) \\ h_{\text{out}} &= x + \text{Dropout}(\text{MLP}(\text{LayerNorm}(x))) \\ h_{\text{out}} &= h_{\text{out}} + \Phi_{l,P}(h_{\text{out}}) \end{aligned}$$

Sparse ReFT can thus be expressed as

$$h_{\text{out}} = h_{\text{out}} + \mathbf{z} \odot \Phi_{l,P}(h_{\text{out}})$$

where $\mathbf{z}$ is a learnable gate for positions in P and 1 otherwise. For clarity, we note that sparse gating only applies to positions where interventions are defined. This results in intervention activations like in Figure 1. We see that in the sonnet generation experiments that the entire first two prefix positions are active and the later layers in the suffix are active. In the paraphrase plot we see that mainly the suffix interventions are present.



(a) Sonnet generation.

(b) Paraphrase detection.

Figure 1: Active interventions after training.

5 Experiments

We evaluate our extension on the experiments as outlined in the default project:

- Cloze-Style Paraphrase Detection
- Sonnet Generation

We briefly describe the requirements for each task.

5.1 Paraphrase Detection

The paraphrase detection task makes use of the Quora dataset. It is a binary classification task, we attached a modified 2-class head (in $\mathbb{R}^{d \times 2}$) that either outputs the token for yes or no. Accuracy and f1 score are used to measure this dataset.

5.2 Sonnet Generation

The Sonnet generation dataset contains 154 sonnets from Shakespeare. The model is tasked with completing the sonnets and is measured based on the ChrF score of the completion [15].

5.3 Data

This is a default project. We used the datasets described in Table 1.

Dataset	Task	# train examples	# dev examples	# test examples
Quora	Paraphrase detection	14,506	20,215	40,431
Sonnet	Sonnet generation	143	0	12

Table 1: Datasets used.

5.4 Evaluation method

We perform an ablation to evaluate the effectiveness of our method. We have performed a series of constructive benchmarks that build up to DiReFT. We first benchmark the baselines model. Then we introduce full fine tuning. Then we introduce DiReFT and see that we can achieve similar performance to full fine tuning with many fewer parameters. Then we introduce L_0 regularization to sparsify the ReFT interventions and see that we further lose negligible task performance. Then evaluate DiReFT initialized to fewer prefix and suffix positions.

Critically, our L_0 ReFT experiments turn off interventions during evaluation. This means that sparsity translates to concrete parameter and computation reduction.

Finally, we perform a sweep over regularization parameter λ , this controls the strength of the regularization, we find that the sparsity accuracy tradeoff is stable.

5.5 Experimental details

We aimed to keep our experiments as standard as possible to minimize external factors in our results. We fix as many hyperparameters as possible and run all experiments with them, see Table 2.

The main difference between benchmarks and the L_0 training runs is that we warm up the λ parameter from 0 to $\lambda_{\max}$ over the course of training to allow the interventions to stabilize before the gates shut off. There is an interesting interplay between the two as the gates are generally randomly initialized and the W_2 matrix in the interventions is initialized to 0, this leads to stable training, but requires training to determine what the intervention should be. If the gates ramp too fast and close, it becomes the case that very few interventions are active as they have not had a chance to properly learn. This necessitates a warm up schedule for λ , we chose a simple linear fractional warm up from zero to λ over the entire duration of training. This allows the gates to form and ReFT to adequately intervene. To demonstrate that the choice of lambda is stable and predictable we provide Pareto frontiers of the task metric vs the sparsity for different λ choices.

Hyperparameter	Value
Optimizer	Adam
Epochs	10
Batch Size	8
Model Variant	GPT-2 Small
ReFT Rank	4
Seeds	11711, 11712, 11713
β	2/3
γ	-0.1
ζ	1.1

(a) Common hyperparameter configuration.

Task	Hyperparameter	Value
Sonnet	Temperature	1.2
Sonnet	Top-P	0.9
Sonnet	Lambda	1.5
Sonnet	Gate LR Mult	100
Paraphrase	Lambda	0.3
Paraphrase	Gate LR Mult	10

(b) Task specific parameters.

Table 2: Hyperparameters.

Due to small dataset sizes we introduce a gate learning rate multiplier. The intuition here is that we need to push the gates closed within a reasonable amount of steps. We cannot just increase lambda as this increases the cost of having open gates and generally pushes training towards closing nearly all of the gates. We found that the Sonnet (the smaller dataset) required a larger multiplier to achieve meaningful sparsity within 10 epochs.

We describe the ReFT experiment parameters in Table 3, all ReFT experiments intervene on all layers.

Experiment	Prefix positions	Suffix positions
ReFT	[0, 1, 2, 3]	[-4, -3, -2, -1]
Small ReFT	[0, 1]	[-2, -1]
Sparse ReFT	[0, 1, 2, 3]	[-4, -3, -2, -1]

Table 3: ReFT configurations. We see that Small ReFT has 50% of the parameters as ReFT and Sparse ReFT experiments.

5.6 Results

We report our results in Table 4. We see an appreciable gain in performance increase per parameter in our experiments.

The first thing to observe is that all methods outperform the frozen baselines. For sonnet generation the entire model is frozen as the experiment makes use of all pretrained layers and does not introduce any new random layers. For paraphrase we train the custom paraphrase detection head only as this is a newly initialized layer, therefore we will see training and accuracy plots for frozen in task.

Experiment	ChrF	Sparsity	# Learned	Experiment	Accuracy	Sparsity	# Learned
Frozen	39.50 $\pm$ 0.6	-	0	Frozen	0.666 $\pm$ 0.003	-	1536
Finetune	41.70 $\pm$ 0.3	-	124M	Finetune	0.898 $\pm$ 0.001	-	124M
Small ReFT	40.96 $\pm$ 0.2	-	150K	Small ReFT	0.844 $\pm$ 0.001	-	150K
ReFT	41.28 $\pm$ 0.2	-	295K	ReFT	0.856 $\pm$ 0.003	-	295K
Sparse ReFT	41.23 $\pm$ 0.570	53.5%	98K	Sparse ReFT	0.852 $\pm$ 0.003	55%	130K

(a) Sonnet.

(b) Paraphrase detection.

Table 4: Dev performance metrics.

6 Analysis

In our results we see that Sparse ReFT is able to reduce the number of active interventions by 40% to 60% with minimal loss in task accuracy. Further, SparseReFT outperforms the baseline Small ReFT by 0.57 ChrF and 0.008 accuracy, this leads us to believe that L_0 is able to keep the important interventions and discard redundant interventions. We note that Sparse ReFT increases the variance

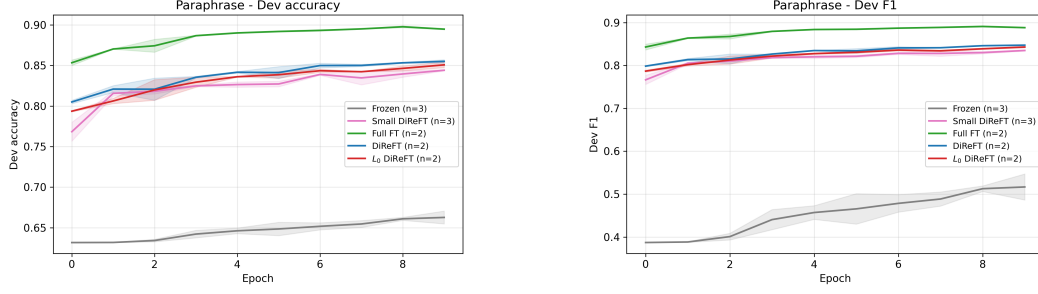


Figure 2: Paraphrase accuracy and F1 on Dev Set.

across seeds. This is potentially due to the discrete nature of the gates at inference time. Differences over seeds can lead to structural differences at inference time.

During parameter tuning we tried various ranks, and found that 4 is the sweet spot. With more than 4 there was little to no gain in task performance and with a reduction to 2 there was consistent loss in task performance.

6.1 Training stability

We generally see stable training loss across the experiments in Figure 3. We see an increase in total loss as λ is ramped in the L_0 experiments, the decomposition of task and train loss allows us to verify that the L_0 penalty does not destroy the signal from task loss. Further, when we decompose the L_0 training loss into the task loss and regularization penalty in Figure 4 we see that both smoothly decrease throughout training.

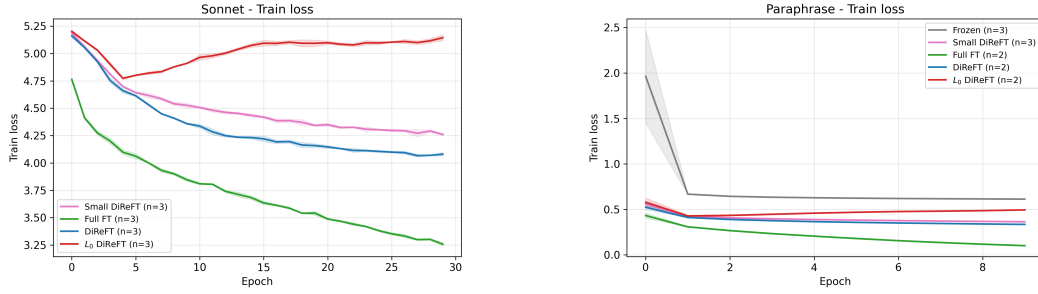


Figure 3: Training Loss.

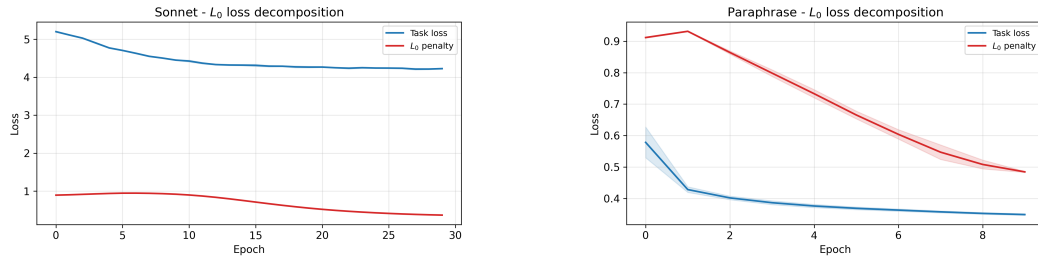


Figure 4: L_0 training loss decomposition. We see both task loss and L_0 loss decrease over time.

6.2 Sparsity analysis

To better understand the performance of the sparse binary gates we analyze a few different metrics. Here in Figure 5 we visualize task accuracy vs sparsity at different lambdas. We see that for our experiment configuration that increased sparsity does not lead to a dramatic decrease in task performance. In fact, on the sonnet generation task the sparse interventions perform nearly the same as the non-sparse ones.

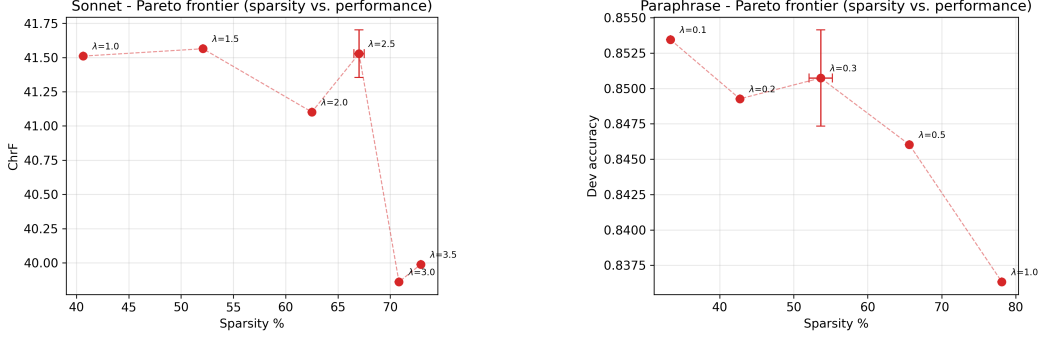


Figure 5: Sweep over sparsity strength (lambda).

In Figure 6 we see that sparsity smoothly increases during training after the warm up period. We see that as the mean gates value decreases sparsity increases as expected.

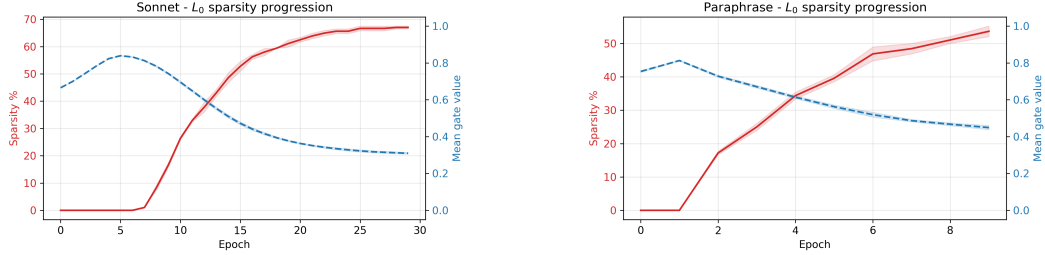


Figure 6: Sparsity and gate activations during training.

6.3 Small Extension of Results

After we had collected all of the results we explored applying ReFT to GPT-2 large. The results are: for Sparse ReFT with $\lambda = 1.5$ we observe ChrF 42.11.

6.4 Limitations

Given the context of this report we would like to informally call out some limitations in our experiments for rigor: Small scale, missing a random pruning baseline, missing a principled pruning baseline, missing hyperparameter sweeps, only used DiReFT and not LoReFT.

7 Conclusion

We show that ReFT can benefit from L_0 regularization to learn which interventions are important at small scale. We performed our experiments with a simplified version of ReFT that uses a unique intervention Φ for each layer and position. The direction of this work could be extended to LoReFT [2] and to a larger scale with more rigorous parameter searches over the baselines.

References

- [1] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization, 2018.
- [2] Zhengxuan Wu, Aryaman Arora, Zheng Wang, Atticus Geiger, Dan Jurafsky, Christopher D. Manning, and Christopher Potts. Reft: Representation finetuning for language models, 2024.
- [3] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.
- [4] Jonas Pfeiffer, Ivan Vulić, Iryna Gurevych, and Sebastian Ruder. MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online, November 2020. Association for Computational Linguistics.
- [5] Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.
- [6] Shwai He, Liang Ding, Daize Dong, Jeremy Zhang, and Dacheng Tao. SparseAdapter: An easy approach for improving the parameter-efficiency of adapters. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2184–2190, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.
- [7] Cheng Fu, Hanxian Huang, Xinyun Chen, Yuandong Tian, and Jishen Zhao. Learn-to-share: A hardware-friendly transfer learning framework exploiting computation and parameter sharing. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 3469–3479. PMLR, 18–24 Jul 2021.
- [8] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [9] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation, 2024.
- [10] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics.
- [11] Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah D. Goodman. Finding alignments between interpretable causal variables and distributed neural representations, 2024.
- [12] Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J. Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering, 2024.
- [13] Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems*, volume 36, pages 41451–41530. Curran Associates, Inc., 2023.

- [14] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.
- [15] Maja Popović. chrF: character n-gram F-score for automatic MT evaluation. In *Proceedings of the tenth workshop on statistical machine translation*, pages 392–395, 2015.