OPT2023: 15th Annual Workshop on Optimization for Machine Learning

New Horizons in Parameter Regularization: A Constraint Approach

Jörg K. H. Franke FRANKEJ@CS.UNI-FREIBURG.DE
University of Freiburg, Germany

Michael Hefenbrock
RevoAI, Germany

Gregor Koehler
German Cancer Research Center (DKFZ), Germany

Frank Hutter
University of Freiburg, Germany

Abstract

This work presents constrained parameter regularization (CPR), an alternative to traditional weight decay. Instead of applying a constant penalty uniformly to all parameters, we enforce an upper bound on a statistical measure (e.g., the L_2-norm) of individual parameter groups. This reformulates learning as a constrained optimization problem. To solve this, we utilize an adaptation of the augmented Lagrangian method. Our approach allows for varying regularization strengths across different parameter groups, removing the need for explicit penalty coefficients in the regularization terms. CPR only requires two hyperparameters and introduces no measurable runtime overhead. We offer empirical evidence of CPR's effectiveness through experiments in the "grokking" phenomenon, image classification, and language modeling. Our findings show that CPR can counteract the effects of grokking, and it consistently matches or surpasses the performance of traditional weight decay.

1. Introduction

While deep neural networks have exhibited unparalleled expressivity, they also possess millions, sometimes trillions, of parameters [5, 25]. This massive capacity makes them susceptible to overfitting, where models memorize nuances of the training data but underperform on unseen examples. To mitigate this, many different regularization techniques have been adopted, with weight decay and L_2 regularization being the most popular [2, 8, 15, 28]. L_2 regularization penalizes the squared magnitude of model parameters and (decoupled) weight decay (which is equivalent to L_2 regularization for non-adaptive gradient algorithms [18]) multiplies all weights with a constant at every step. This seemingly simple act offers numerous benefits by curbing the growth of individual weights, reducing the risk of relying on any particular feature excessively, and thus promoting model generalization.

However, not all parameters in a neural network have the same role or importance and different weights could benefit from different regularizations. Similarly, it is unclear if a single weight decay value is optimal for the entire duration of optimization, especially for large-scale training. Indeed, Ishii and Sato [10] showed that a small deep learning model could benefit from layer-wise weight decay values, and various works showed that scheduling weight decay could improve final performance [3, 16, 20, 27]. This indicates that a dynamic penalty for each individual parameter group could be beneficial for neural network training.

© J.K.H. Franke, M. Hefenbrock, G. Koehler & F. Hutter.

Since scheduled or parameter-wise weight decay comes with additional hyperparameters (which are often sensitive to the task), we propose a different approach to obtain customized, dynamic parameter regularization. Instead of uniformly penalizing weights, we propose to keep them in a certain range, thus ensuring stability without imposing regularization where it is unnecessary. Constraining parameters, especially based on statistical measures like the L_2 norm, provide a flexible and adaptive form of regularization that accounts for the heterogeneity of parameters.

In this paper, we propose *constrained parameter regularization (CPR)*, which enforces an upper bound on a statistical measure of individual parameter groups (e.g., a weight matrix in a linear layer). This allows us to reformulate regularization as a constrained optimization problem, which we address by an adaptation of the augmented Lagrangian method. The resulting regularization strength is individual to each parameter group and adaptive over time while being configurable by only two hyperparameters.

It should be mentioned that CPR is not the first to use Lagrangian methods in machine learning [21]. Its application in deep learning focuses mainly on variational methods and generative models. For example, Rezende and Viola [24] introduced the *Generalized ELBO with Constrained Optimization* algorithm to optimize VAEs using Lagrange multipliers, and Kohl et al. [13] and Franke et al. [6] adapted the method to train probabilistic U-nets and probabilistic Transformer models. While these works leverage the Lagrangian to handle several losses in joint optimization problems, our work leverages it to enable individual regularization strengths.

In the following, we introduce CPR for individualized and dynamic parameter regularization (Section 2). Specifically, to avoid the need for separate penalty scaling coefficients, we formulate regularization as a constraint problem and derive CPR as the solution to this problem (Section 2.1). We identify three different ways for initializing upper bounds on the statistical measure (Section 2.2). We provide empirical evidence of the benefit of CPR by showing improved performance over weight decay in experiments on the "grokking" phenomenon (Section 3.1), image classification with ResNets (Section 3.2) and a language modeling task with mid-scale GPT2 (Section 3.3). We provide an open-source implementation of CPR which can be easily adapted by replacing the optimizer class.

2. Constrained Parameter Regularization

In this section, we introduce Constrained Parameter Regularization (CPR), where we adapt the augmented Lagrangian method to enforce upper bounds on regularization terms. Please find a review of the augmented Lagrangian method in Appendix A. Compared to classical regularization, with a fixed regularization coefficient γ, the proposed approach will allow for variable regularization coefficients λ^j (Lagrange multipliers) for $j = 1, \cdots, J$ parameter groups $\boldsymbol{\theta}^j \subseteq \boldsymbol{\theta}$ that should be regularized. These regularization coefficients are updated alongside the network parameters $\boldsymbol{\theta}$.

2.1. Regularization through constraints

Classical weight decay, as introduced earlier, is used as a means to restrict the freedom of parameter adaptation. This restriction is applied with a scaling factor γ (hyperparameter) and applies uniformly to all parameters. However, we conjecture that applying an individual adaptation pressure instead may be beneficial. Unfortunately, this would require a separate coefficient for each parameter group where a separate weight decay should be applied. To avoid the need for separate scaling coefficients, we formulate regularization as a constrained problem. Here, the loss function $L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y})$, with

network parameters $\boldsymbol{\theta}$, takes the place of the objective. Consequently, the learning problem becomes

$$\underset{\boldsymbol{\theta}}{\text{minimize}}\ L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) \quad \text{s.t.} \quad c_j(\boldsymbol{\theta}^j) = R(\boldsymbol{\theta}^j) - \kappa^j \leq 0, \quad \text{for} \quad j = 1, \cdots, J, \tag{1}$$

where $R(\boldsymbol{\theta}^j)$ is a regularization function (e.g., the L_2-norm in case of weight decay) for a parameter group $\boldsymbol{\theta}^j \subseteq \boldsymbol{\theta}, j = 1, \cdots, J$, and $\kappa \in \mathbb{R}$ denotes a chosen bound.

To solve Equation 1, we follow the augmented Lagrangian method with slight modifications. First, instead of performing a full optimization of the loss before updating $\bar{\lambda}$, we perform updates in every step. This is motivated by the fact that full optimization is generally infeasible in a deep learning setting. Moreover, similar to the difference between weight decay and L_2-regularization, we treat the update between the loss-dependent and the constraint-dependent part separately. Hence, instead of introducing $\hat{L}(\boldsymbol{x}, \bar{\lambda}, \mu)$ analogously to Equation 3, and performing optimization on this objective, we independently apply updates for both steps. Consequently, the constraint violations do not accumulate in momentum terms. We also remove the influence of the learning rate on the regularization. From a practical perspective, our modification does not interfere with gradient-based optimization algorithms and can be readily combined with any such optimizer. The full algorithm is given by Algorithm 1 in Appendix B.

Conceptually, the method can be understood as the λ^j accumulating constraint function values (weighted with μ) over the iterations. These then increase (or decrease) the influence of the constraint (via its gradient) on the search direction. When points in the feasible domain are found for which $c_j(\boldsymbol{\theta}) \leq 0$, λ is decreased until it eventually reaches 0. If, on the other hand, the optimal solution lies on the boundary, where $c_j(\boldsymbol{\theta}) = 0$, λ should converge to a value $\lambda^\star$ where the update direction of the optimizer and the gradient of the constraints cancel each other. However, this situation is unlikely to occur in a deep learning setting due to the stochasticity of minibatches and potential adaptations to the learning rate.

2.2. Initialization of Bounds

The upper bound κ is the most crucial hyperparameter for CPR, and we identify three ways to initialize it. (1) Set κ uniform (`Kappa-K`): Set one value for all regularized parameter groups as an initial value for the upper bound, $\kappa \in \mathbb{R}^+$. (2) Set κ based on θ-initialization (`Kappa-kI`$_0$): Initialize the upper bound based on the initial parameter groups' regularization function, which could be affected by a parameter group's individual size and/or initialization scheme (e.g. a depth-dependent initialization): $\kappa^i = k \cdot R(\boldsymbol{\theta}^i_{t=0})$, with $k \in \mathbb{R}^+$ as the factor of the initial measure. (3) Set κ with warm start (`Kappa-I`$_s$): Instead of selecting a factor k of the initial regularization function, train our model parameters $\boldsymbol{\theta}$ for a specific number of update steps and then bind the regularization to the current regularization function value: $\kappa^i = R(\boldsymbol{\theta}^i_{t=s})$, with $s \in \mathbb{N}^+$ as a hyperparameter for the start of the regularization; please find an integration in CPR in Appendix B.2.

3. Experiments

We now describe three experiments to better understand CPR and the initialization of κ. Preliminary experiments showed that μ is not a sensitive hyperparameter, and we chose $\mu = 1.0$ for all our experiments. This leads to the upper bound κ as the sole hyperparameter of CPR. In our experiments, we consider a weight matrix in a neural network as an individual parameter group and regularize all parameters in a network except for biases and normalization weights.

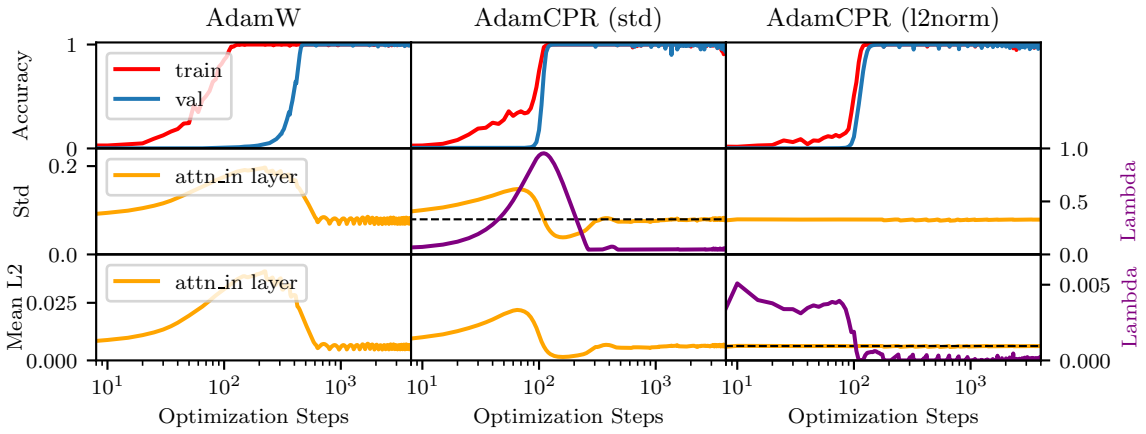

Figure 1: Experiments on the modular addition task: The training steps on the x-axis are displayed in log scale. We display the training and validation accuracy in red and blue (top), the standard deviation of the *attention-in* weight matrix (middle), and the mean L_2 norm (bottom). In the left column, we use AdamW for optimization, and next to it AdamCPR with standard deviation (middle) and L_2 norm (right) as regularization function. We see the λ value during training (purple) and the dotted black line represents the upper bound κ.

3.1. Modular Addition

In the realm of neural network regularization, the phenomenon of *grokking* has garnered significant attention. As discovered by Power et al. [22], grokking is characterized by a sudden generalization after prolonged training without discernible improvements in training or validation loss. We train a 1-layer Transformer on the modular addition task which is the primary benchmark for studying this phenomenon [17]. To explore CPR's possibilities, we compare it to AdamW with weight decay $\gamma = 1.0$ [12, 18]. We consider two regularization variations for CPR: one constraint based on the L_2 norm and one on the standard deviation. The standard deviation is interesting since it does not constrain the weight parameters to be centered around zero. We use `Kappa-kI`$_0$ for the initialization of κ with a factor of $k = 0.8$ for the L_2 norm and $k = 0.9$ for the standard deviation. We found these factors by a small grid search influenced by the rescaling described by Liu et al. [17]. A list of all hyperparameters can be found in Appendix C.

The results are presented in Figure 1. They reveal that AadmCPR nearly mitigates the effects of grokking and achieves faster convergence. Both constraint variations successfully bridge the performance gap between training and validation by dynamically regularizing parameters. Notably, the CPR based on standard deviation exhibited a more uniform behavior across the weight matrix. But at the end of the training, the variation with the standard deviation displays a less stable behavior. This may be caused by not encouraging a zero-centered parameter distribution. A unique feature of our approach is the individual adaptation of each parameter. For a granular analysis of each layer's behavior, we point to additional plots in Appendix C, where we see individual λ^j adaptions over the training progress. Due to its greater stability, we resort to the L_2 norm in the following experiments.

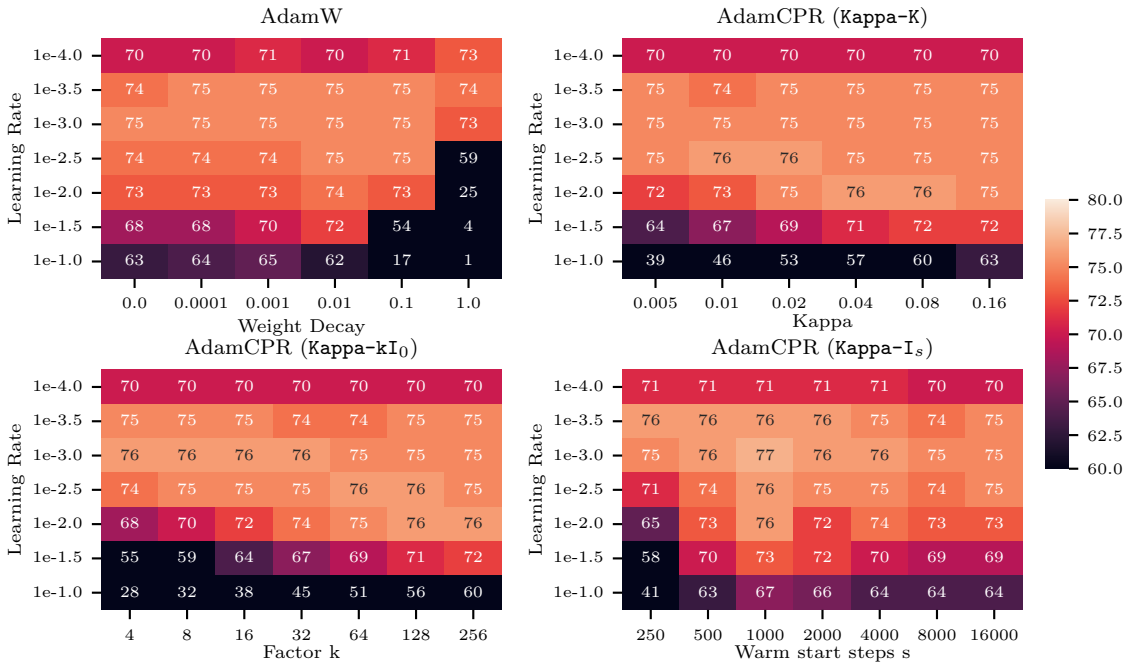

Figure 2: Mean percentage of correct labels across three seeds of a ResNet18 trained on CIFAR100 with use of AdamW (top left), AdamCPR with `Kappa-K` initialization (top right), `Kappa-kI`$_0$ (bottom left), and `Kappa-I`$_s$ (bottom right).

3.2. Image Classification

To evaluate CPR's effectiveness and design choices further, we tested CPR in image classification using ResNet18 on the CIFAR100 dataset [9, 14]. We compared AdamW to AdamCPR with the three initializations described in Section 2.2 and L_2-norm as a measure for the regularization constraint. For the κ initialization `Kappa-K`, we use a range of $\kappa = [0.005, \ldots, 0.16]$, for `Kappa-kI`$_0$ a range of $k = [4, \ldots, 256]$, and for `Kappa-I`$_s$ a range of $s = [250, \ldots, 4000]$ steps. Thus, the warmup steps we used for κ are in the same range as the learning rate warmup (500 steps). The ResNet18 was trained on a consumer GPU with no significant runtime difference between AdamW and AdamCPR. For a detailed list of training hyperparameters, we refer the reader to Table 3. Figure 2 shows the best mean validation performance for different learning rates and regularization hyperparameters. According to our experiments, the best configurations of AdamCPR outperform AdamW with different weight decay values. We found that initializing with `Kappa-I`$_s$ performs better than selecting a uniform κ in `Kappa-K` or a factor k in `Kappa-kI`$_0$. This could be because the warm started bounds may be considered "learned" since they reflect the actual magnitudes and distributions of the parameter groups in the training process and also depend on the training data. So the upper bound could be better adapted to the learning task. Finally, we want to note the seemingly linear dependence between the learning rate and well-performing initialization factors k when using `Kappa-kI`$_0$-

Table 1: The perplexity (PPL) of the GPT2 experiments. The values below the method denote the weight decay factor γ in case of AdamW and for AdamCPR the warm-start steps s of the $\texttt{Kappa-I}_s$ initialization.

Model	Runtime Size	AdamW 1e-3	AdamW 1e-2	AdamW 1e-1	AdamCPR 5k	AdamCPR **10k**	AdamCPR 20k
GPT2s	200k	17.98	17.84	18.58	17.96	**17.68**	17.80
GPT2s	400k	-	17.43	-	-	**17.32**	-
GPT2m	200k	-	14.23	-	-	**14.03**	-

3.3. Language Modelling

We also performed experiments training a mid-size GPT2 language model [23] on Openwebtext [7]. For an efficient implementation, we use flash attention [4] and rotary position embedding [26]. We compared AdamW on different weight decay values to AdamCPR. We use the L_2-norm as a measure for the regularization constraint and the warm-started κ initialization $\texttt{Kappa-I}_s$. We use a learning rate warmup for 5k steps, a cosine annealing, and train for 200k steps. We orientated the warmup steps of κ based on the warmup steps of the learning rate and evaluated initializing κ after 5k, 10k, and 20k steps. The complete hyperparameters can be found in Appendix E. The GPT2 model is trained on 8 A100 GPUs and the runtime for AdamW and AdamCPR did not differ. The mean results across three random seeds in Table 1 suggest that CPR outperforms the best weight decay configuration and that the selection for the warmup time for AdamCPR seems to not be very sensitive in this case. To investigate the scalability of our results above, we also performed experiments over a longer runtime (GPT2s 400k) and on a larger model (GPT2m). We used the same weight decay and $\texttt{Kappa-I}_s$ initialization as in the best GPT2s/200k experiment. We find again that AdamCPR outperforms AdamW, which could indicate that CPR is also capable of optimizing larger models or longer training. We show the learning curves of the best AdamW and best AdamCPR run in Figure E.1. We observe that CPR regularizes the model more strongly in the early stages of training which may lead to better final performance.

4. Conclusion & Future Work

In this work, we introduce constrained parameter regularization (CPR), a method for regularization of neural network training via constraints. By enforcing an upper bound on a regularization function, we achieve effective regularization of the neural network training. The constraints are handled by an adaptation of the augmented Lagrangian method without notable runtime overhead over standard weight decay. We provide empirical evidence for the capabilities of CPR weight decay on modular addition, image classification, and language modeling. CPR can be combined with any gradient-based optimizer and requires only a minor addition to the training loop. Nonetheless, CPR does require tuning of the hyperparameter κ to achieve superior performance. Future work could focus on a more efficient initialization of κ. For example, investigations could explore the relationship between the learning rate, parameter initialization, and the bound on the regularization function. Furthermore, one could adjust the upper bound during optimization to enhance regularization and increase tolerance against unfavorable initialization.

Acknowledgement

This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 417962828. We acknowledge funding by the European Union (via ERC Consolidator Grant DeepLearning 2.0, grant no. 101045765). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer JUWELS [11] at Jülich Supercomputing Centre (JSC).

References

[1] D. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, 1996. ISBN 1886529043.

[2] S. Bos and E. Chug. Using weight decay to optimize the generalization ability of a perceptron. In *Proceedings of International Conference on Neural Networks (ICNN'96)*, volume 1, pages 241–246 vol.1, 1996. doi: 10.1109/ICNN.1996.548898.

[3] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[4] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

[5] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

[6] Jörg Franke, Frederic Runge, and Frank Hutter. Probabilistic transformer: Modelling ambiguities and distributions for rna folding and molecule design. *Advances in Neural Information Processing Systems*, 35:26856–26873, 2022.

[7] Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[8] Stephen Hanson and Lorien Pratt. Comparing biases for minimal network construction with back-propagation. In *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the International Conference on Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016.

[10] Masato Ishii and Atsushi Sato. Layer-wise weight decay for deep neural networks. In *Image and Video Technology*, pages 276–289, Cham, 2018. Springer International Publishing.

[11] Jülich Supercomputing Centre. JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre. *Journal of large-scale research facilities*, 7(A138), 2021. doi: 10.17815/jlsrf-7-183. URL http://dx.doi.org/10.17815/jlsrf-7-183.

[12] D. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR'14)*. CBLS, 2014.

[13] Simon Kohl, Bernardino Romera-Paredes, Clemens Meyer, Jeffrey De Fauw, Joseph R Ledsam, Klaus Maier-Hein, SM Eslami, Danilo Jimenez Rezende, and Olaf Ronneberger. A probabilistic u-net for segmentation of ambiguous images. *Advances in neural information processing systems*, 31, 2018.

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[15] Anders Krogh and John Hertz. A simple weight decay can improve generalization. *Advances in Neural Information Processing Systems*, 4, 1991.

[16] Aitor Lewkowycz and Guy Gur-Ari. On the training dynamics of deep networks with l_2 regularization. In *Advances in Neural Information Processing Systems*, volume 33, pages 4790–4799, 2020.

[17] Ziming Liu, Eric J Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data. In *The Eleventh International Conference on Learning Representations*, 2023.

[18] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations (ICLR'19)*, 2019. Published online: iclr.cc.

[19] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[20] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[21] John Platt and Alan Barr. Constrained differential optimization. In *Neural Information Processing Systems*, volume 0, 1987.

[22] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. *1st Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021*, 2021.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[24] Danilo Jimenez Rezende and Fabio Viola. Taming vaes. *arXiv preprint arXiv:1810.00597*, 2018.

[25] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[26] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2021.

[27] Juseung Yun, Byungjoo Kim, and Junmo Kim. Weight decay scheduling and knowledge distillation for active learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVI 16*, pages 431–447. Springer, 2020.

[28] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2018.

Appendix

Appendix A. Background on the augmented Lagrangian

A.1. The augmented Lagrangian method

We briefly review the augmented Lagrangian method, see e.g. [1], which our method is based on. For the derivation, we follow the motivation of Nocedal and Wright [19, pp. 523-524].

Consider the following inequality-constrained optimization problem

$$\underset{\boldsymbol{x}}{\text{minimize}}\ f(\boldsymbol{x}) \quad \text{s.t.} \quad c(\boldsymbol{x}) \leq 0,$$

with $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$ and a constraint $c(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}$. One way to address the constraint is to find an equivalent, unconstrained problem with the same optimal solution. For example,

$$\underset{\boldsymbol{x}}{\text{minimize}}\ F(\boldsymbol{x}) \quad \text{with} \quad F(\boldsymbol{x}) = \max_{\lambda \geq 0}\ f(\boldsymbol{x}) + \lambda \cdot c(\boldsymbol{x}). \tag{2}$$

Evidently, for any infeasible point $\boldsymbol{x}$ with $c(\boldsymbol{x}) > 0$, $\lambda \cdot c(\boldsymbol{x})$ in the inner maximization can yield arbitrarily high values ($\to \infty$). Thus, any solution candidate must clearly be feasible. Unfortunately, $F(\boldsymbol{x})$ is not suitable for gradient-based optimization, as it provides no useful gradient information to restore feasibility. To alleviate this problem, we consider a smooth approximation of $F(\boldsymbol{x})$, namely

$$\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) = \max_{\lambda \geq 0}\ f(\boldsymbol{x}) + \lambda \cdot c(\boldsymbol{x}) - \frac{1}{2\mu}(\lambda - \bar{\lambda})^2, \tag{3}$$

where $\bar{\lambda} \in \mathbb{R}$ may be seen as a point we wish to remain proximal to and $\mu \in \mathbb{R}^+$ as a factor determining the strength with which this proximity is enforced. For $\mu \to \infty$, $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) \to F(\boldsymbol{x})$.

The maximization in $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ has a closed form solution with $\lambda^\star = (\bar{\lambda} + \mu \cdot c(\boldsymbol{x}))^+$, where $(\cdot)^+ = \max\{0, \cdot\}$, see Appendix A.2 for the derivation.

Consequently,

$$\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) = f(\boldsymbol{x}) + h(\boldsymbol{x}, \bar{\lambda}, \mu) \tag{4}$$

with

$$h(\boldsymbol{x}, \bar{\lambda}, \mu) = \begin{cases} c(\boldsymbol{x})(\bar{\lambda} + \frac{\mu}{2} c(\boldsymbol{x})), & \text{if} \quad \bar{\lambda} + \mu \cdot c(\boldsymbol{x}) \geq 0 \\ -\frac{1}{2\mu}\bar{\lambda}^2 & \text{else.} \end{cases} \tag{5}$$

The constraint thus only interferes with the minimization (gradient) of $f(\boldsymbol{x})$ if $(\bar{\lambda} + \mu \cdot c(\boldsymbol{x}))^+ \geq 0$.

We can now try to solve the unconstrained problem $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ with familiar methods, such as gradient descent, and obtain an approximate solution to the original problem. Specifically, the gradient of $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ with respect to $\boldsymbol{x}$ is given by

$$\nabla_{\boldsymbol{x}} \hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \lambda^\star \cdot \nabla_{\boldsymbol{x}} c(\boldsymbol{x}). \tag{6}$$

The quality of the approximation, and thus the solution, clearly depends on μ and $\bar{\lambda}$. However, after solving $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ for some value of $\bar{\lambda}$, we can perform an update step $\bar{\lambda} \leftarrow \lambda^\star$ and attempt to perform minimization again. Intuitively, if the previous minimization of $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ resulted in an infeasible solution with $c(\boldsymbol{x}) > 0$, $\bar{\lambda}$ is increased. Hence, the next minimization of $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ likely results in a solution with less constraint violation. On the other hand, if $c(\boldsymbol{x}) \leq 0$, $\bar{\lambda}$ is decreased. Subsequently, the influence of the constraint is decreased. This loop of alternating minimization of

$\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$ and update to $\bar{\lambda}$ can be repeated until a sufficiently good solution is found or the procedure converges if $\bar{\lambda}$ does not receive updates anymore.

For multiple constraints $c_j(\boldsymbol{x}),\ j = 1, \cdots, J$, the above can be readily extended with a multiplier λ^j for each constraint. Since the maximization in the smooth approximation is separable in the λ^j, the same update rule may be applied for each λ^j separately using on the respective constraint $c_j(\boldsymbol{x})$.

A.2. Derivation of the Lagrange multiplier update

For simplicity, we consider a single constraint. Note that multiple constraints can be addressed separately as the optimization problem would be separable in the respective λ^j. We need to solve

$$\underset{\lambda \geq 0}{\text{maximize}}\ f(\boldsymbol{x}) + \lambda \cdot c(\boldsymbol{x}) - \frac{1}{2\mu}(\lambda - \bar{\lambda})^2.$$

The optimal point of this problem is equivalent to the optimal point of

$$\underset{\lambda}{\text{minimize}}\ -f(\boldsymbol{x}) - \lambda \cdot c(\boldsymbol{x}) + \frac{1}{2\mu}(\lambda - \bar{\lambda})^2 \quad \text{s.t.} \quad -\lambda \leq 0.$$

To find candidates for optimal points, we need to solve the Karush–Kuhn–Tucker (KKT) system with the Lagrange function $\mathcal{L}(\lambda, \psi)$ and the Lagrange multiplier ψ

$$\mathcal{L}(\lambda, \psi) = -f(\boldsymbol{x}) - \lambda \cdot c(\boldsymbol{x}) + \frac{1}{2\mu}(\lambda - \bar{\lambda})^2 - \psi \cdot \lambda$$

Which leads to the KKT system

$$\begin{aligned} \nabla_\lambda \mathcal{L}(\lambda, \psi) = 0 &\iff 0 = -c(\boldsymbol{x}) + \frac{1}{\mu}(\lambda - \bar{\lambda}) - \psi \\ \nabla_\psi \mathcal{L}(\lambda, \psi) \leq 0 &\iff 0 \geq -\lambda \\ \lambda \cdot \psi = 0 & \end{aligned} \tag{7}$$

According to the complementary conditions Equation 7, the constraint is either active, hence $\lambda = 0$ and $\psi \geq 0$ or inactive, such that $\lambda > 0$, and consequently, $\psi = 0$.

Case: $\lambda = 0$ and $\psi \geq 0$

Here, $\lambda = 0$ (by assumption), and ψ is given by

$$\begin{aligned} \nabla_\lambda \mathcal{L}(\lambda, \psi) = 0 &\iff 0 = -c(\boldsymbol{x}) + \frac{1}{\mu}(0 - \bar{\lambda}) - \psi \\ &\ \psi = -c(\boldsymbol{x}) - \frac{\bar{\lambda}}{\mu} \end{aligned}$$

Since we require $\psi \geq 0$ for a KKT point, (note that $\mu > 0$)

$$\begin{aligned} & 0 \leq \psi = -c(\boldsymbol{x}) - \frac{\bar{\lambda}}{\mu} \\ \iff\ & 0 \leq -\mu \cdot c(\boldsymbol{x}) - \bar{\lambda} \\ \iff\ & 0 \geq \bar{\lambda} + \mu \cdot c(\boldsymbol{x}) \end{aligned}$$

Consequently, $\lambda = 0$ is a candidate for the optimal point only when $0 \geq \bar{\lambda} + \mu \cdot c(\boldsymbol{x})$.

Case: $\lambda > 0$ and $\psi = 0$ (inactive constraint)

For this case we get

$$\begin{aligned} \nabla_\lambda \mathcal{L}(\lambda, \psi) = 0 &= -c(\boldsymbol{x}) + \frac{1}{\mu}(\lambda - \bar{\lambda}) - 0 \\ 0 &= -\mu \cdot c(\boldsymbol{x}) + \lambda - \bar{\lambda} \\ \lambda &= \bar{\lambda} + \mu \cdot c(\boldsymbol{x}) \end{aligned}$$

Due to the geometry of the problem (quadratic with bound constraint), $\lambda = 0$ is the optimal solution if the constraint is active, i.e., if $\psi \geq 0$, which is the case if $0 \geq \bar{\lambda} + \mu \cdot c(\boldsymbol{x})$. Consequently, the optimal solution is given by

$$\lambda^\star = (\bar{\lambda} + \mu \cdot c(\boldsymbol{x}))^+. \tag{8}$$

Plugging this into $\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu)$, we get

$$\hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) = \begin{cases} f(\boldsymbol{x}) + c(\boldsymbol{x})(\bar{\lambda} + \frac{\mu}{2} c(\boldsymbol{x})), & \text{if} \quad \bar{\lambda} + \mu \cdot c(\boldsymbol{x}) \geq 0 \\ f(\boldsymbol{x}) - \frac{1}{2\mu}\bar{\lambda}^2, & \text{else} \end{cases}$$

And the gradient with respect to $\boldsymbol{x}$ is

$$\nabla_{\boldsymbol{x}} \hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) = \begin{cases} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} c(\boldsymbol{x})(\bar{\lambda} + \mu \cdot c(\boldsymbol{x})), & \text{if} \quad \bar{\lambda} + \mu \cdot c(\boldsymbol{x}) \geq 0 \\ \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) - 0 & \text{else} \end{cases}$$

Or more compactly by using Equation 8

$$\nabla_{\boldsymbol{x}} \hat{F}(\boldsymbol{x}, \bar{\lambda}, \mu) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} c(\boldsymbol{x}) \cdot \lambda^\star.$$

Appendix B. The CPR Algorithm

B.1. The CPR Algorithm with `Kappa-K` or `Kappa-kI`$_0$

Algorithm 1 Optimization with constrained parameter regularization (CPR).

Require: Loss Function $L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y})$ with parameters $\boldsymbol{\theta}$, and data $\mathcal{D} = \{(\boldsymbol{X}_n, \boldsymbol{y}_n)\}_{n=0}^{N}$
Require: Hyperparameters: Learning rate $\eta \in \mathbb{R}^+$, Lagrange multiplier update rate $\mu \in \mathbb{R}^+$
Require: Optimizer $\mathrm{Opt}(\cdot)$ for minimization, Regularization function $R(\boldsymbol{\theta})$ (e.g. L2-norm)

Initialization
$t \leftarrow 0$
$\boldsymbol{\theta}_t \leftarrow \mathrm{Initialize}(L(\cdot))$
$\lambda_t^j \leftarrow 0$ for $j = 1, \cdots, J$
$\kappa^j \leftarrow \mathrm{Initialize}(\boldsymbol{\theta}_0^j)$ for $j = 1, \cdots, J$ ▷ Initializing the upper bound, see Section 2.2
# Training
for $\boldsymbol{X}_t, \boldsymbol{y}_t \sim \mathcal{D}$ **do**
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \mathrm{Opt}(L(\boldsymbol{\theta}_t, \boldsymbol{X}_t, \boldsymbol{y}_t), \eta)$ ▷ Classic parameter update using, e.g., Adam.
    for each regularized parameter group $\boldsymbol{\theta}_t^j$ in $\boldsymbol{\theta}_t$ **do**
        $\lambda_{t+1}^j \leftarrow \left(\lambda_t^j + \mu \cdot (R(\boldsymbol{\theta}_t^j) - \kappa^j)\right)^+$
        $\boldsymbol{\theta}_{t+1}^j \leftarrow \boldsymbol{\theta}_{t+1}^j - \nabla_{\boldsymbol{\theta}^j} R(\boldsymbol{\theta}_t^j) \cdot \lambda_{t+1}^j$
    end for
    $t \leftarrow t + 1$
end for

B.2. The CPR Algorithm with `Kappa-I`$_s$

Algorithm 2 Optimization with constrained parameter regularization (CPR) and `Kappa-I`$_s$.

Require: Loss Function $L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y})$ with parameters $\boldsymbol{\theta}$, and data $\mathcal{D} = \{(\boldsymbol{X}_n, \boldsymbol{y}_n)\}_{n=0}^{N}$
Require: Hyperparameters: Learning rate $\eta \in \mathbb{R}^+$, Lagrange multiplier update rate $\mu \in \mathbb{R}^+$, starting step s for CBR.
Require: Optimizer $\mathrm{Opt}(\cdot)$ for minimization, Regularization function $R(\boldsymbol{\theta})$ (e.g. L2-norm)

Initialization
$t \leftarrow 0$
$\boldsymbol{\theta}_t \leftarrow \mathrm{Initialize}(L(\cdot))$
$\lambda_t^j \leftarrow 0$ for $j = 1, \cdots, J$
$\kappa^j \leftarrow \infty$ $j = 1, \cdots, J$
# Training
for $\boldsymbol{X}_t, \boldsymbol{y}_t \sim \mathcal{D}$ **do**
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \mathrm{Opt}(L(\boldsymbol{\theta}_t, \boldsymbol{X}_t, \boldsymbol{y}_t), \eta)$ ▷ Classic parameter update using, e.g., Adam.
    for each regularized parameter group $\boldsymbol{\theta}_t^j$ in $\boldsymbol{\theta}_t$ **do**
        $\lambda_{t+1}^j \leftarrow \left(\lambda_t^j + \mu \cdot (R(\boldsymbol{\theta}_t^j) - \kappa^j)\right)^+$
        $\boldsymbol{\theta}_{t+1}^j \leftarrow \boldsymbol{\theta}_{t+1}^j - \nabla_{\boldsymbol{\theta}^j} R(\boldsymbol{\theta}_t^j) \cdot \lambda_{t+1}^j$
        if $t = s$ **then** ▷ `Kappa-kI`$_s$ initialization, see Section 2.2.
            $\kappa^j \leftarrow R(\boldsymbol{\theta}_t^j)$
        end if
    end for
    $t \leftarrow t + 1$
end for

Appendix C. Experiments on Modular Addition Task

Table 2: Hyperparameters in the modular addition task.

Parameter	Value
Modular addition p-value	113
Train fraction	0.3
Batch size	512
Model dim	128
Number of layers	1
Number of heads	4
Activation	ReLU
Initialization type	sqrt_dim
Learning rate	0.001
Adam β_1	0.9
Adam β_2	0.98
Exclude from regularization	bias, norm

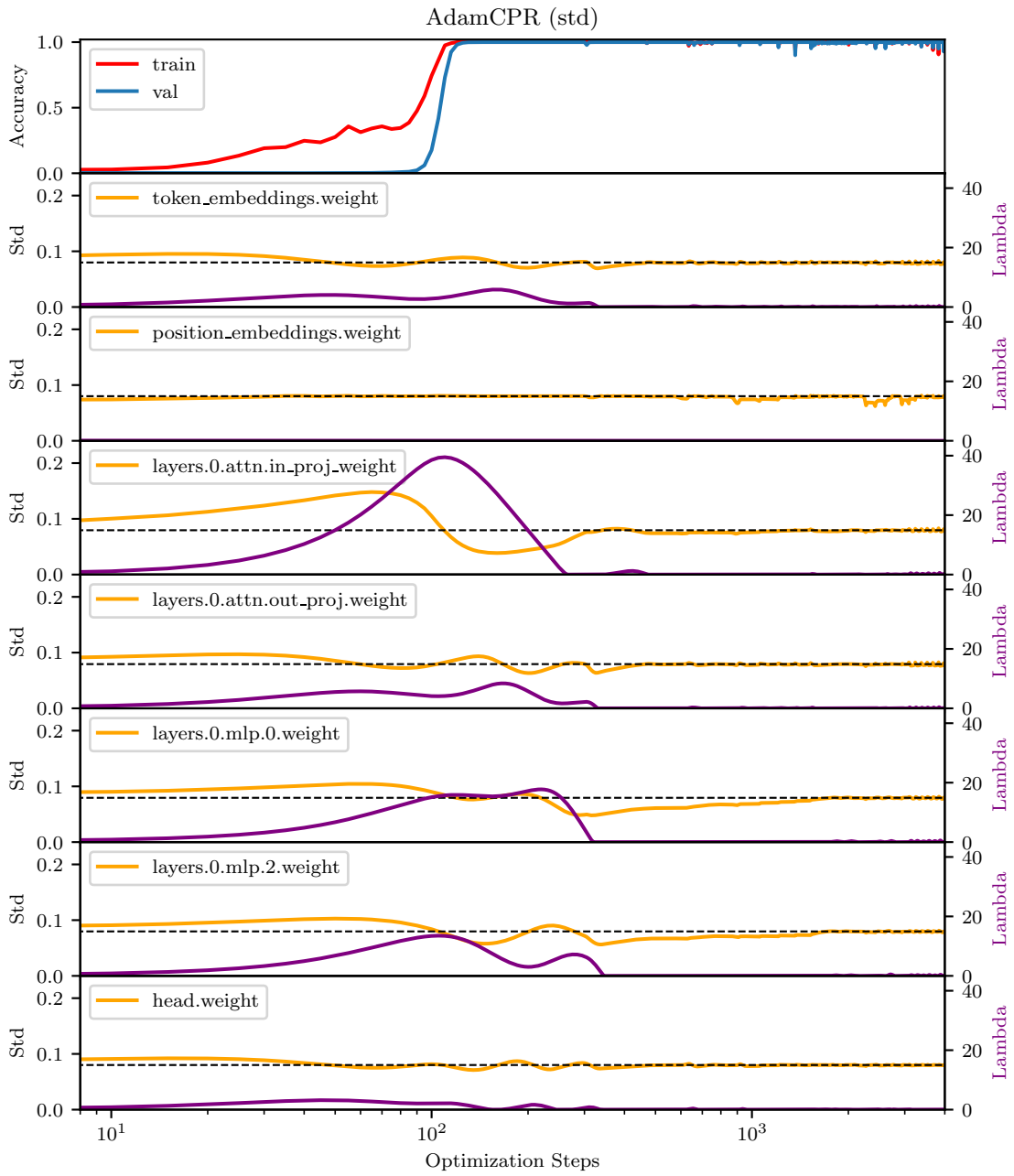

Figure C.1: Experiments on the modular addition task to illustrate the effect of CPR with regularization on the standard deviation on the different layers in the neural network. The x-axis displays the training steps in log scale. The top row shows the training and the validation accuracy in red and blue respectively. In the rows below, we see the standard deviation of the different layers during the training progress.

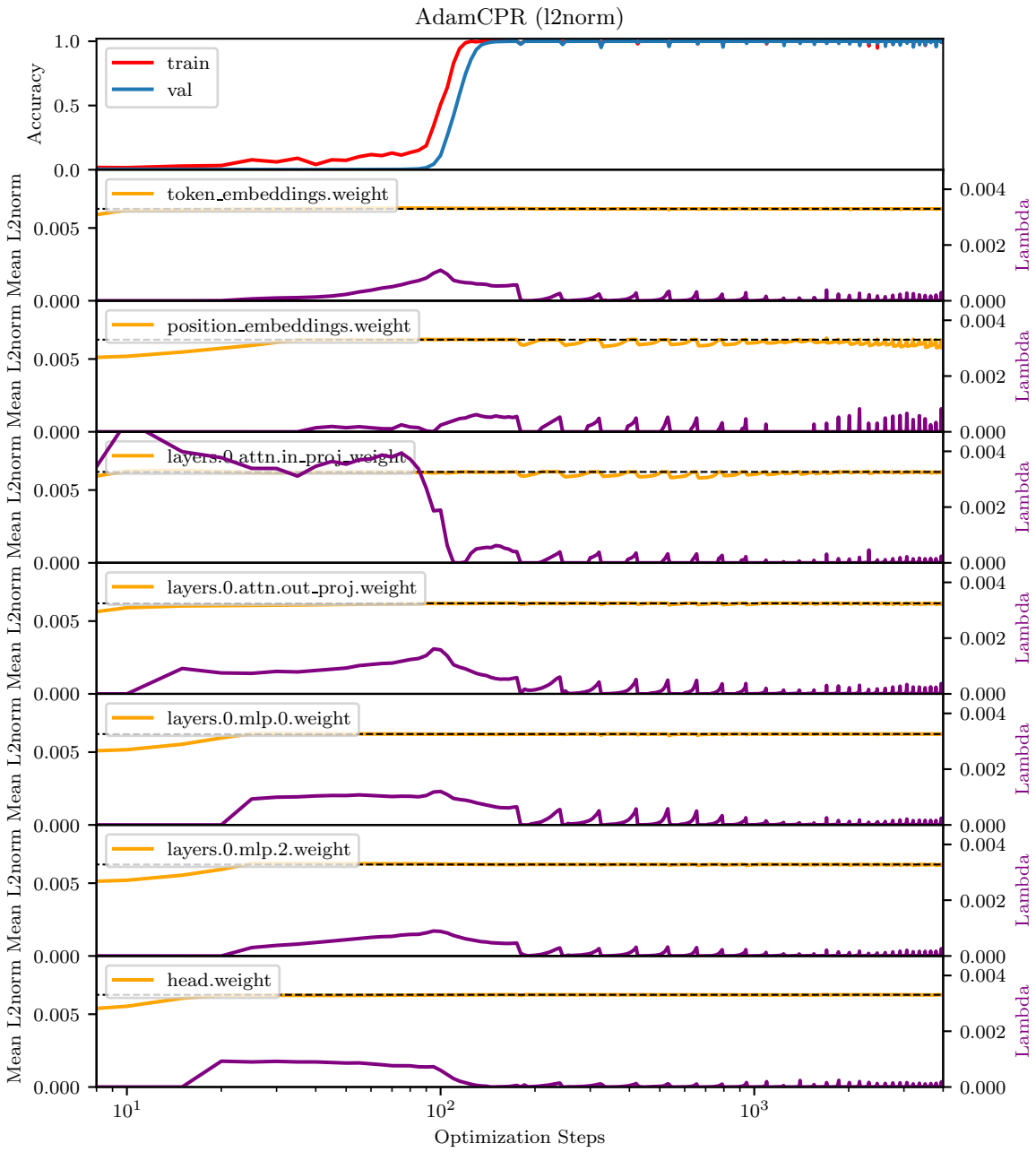

Figure C.2: Experiments on the modular addition task to illustrate the effect of CPR with regularization on the standard deviation on the different layers in the neural network. The x-axis displays the training steps in log scale. The top row shows the training and the validation accuracy in red and blue respectively. In the layer below, we see the mean L_2 norm of the different layers during the training process.

Appendix D. Experiments on Image Classification

Table 3: Hyperparameters of the ResNet18 on CIFAR100 experiment.

Parameter	**Value**
Seed	1,2,3
Dataset	CIFAR100
Batch size	128
Training Steps	20000
Model	ResNet18
Optimizer	AdamW / AdamCPR
Learning Rate	0.001
Beta1	0.9
Beta2	0.98
Weight Decay	0.1
Lr Schedule	Cosine with warmup
Lr Warmup Steps	500
Lr Decay Factor	0.1
Rescale Alpha	0, 0.8 . . . 16
CPR-μ	1.0
CPR-κ	0.8 . . . 16
CPR-k	4 . . . 256
CPR-κ warm-start steps	250 . . . 16000

Appendix E. Experiments on Language Modelling

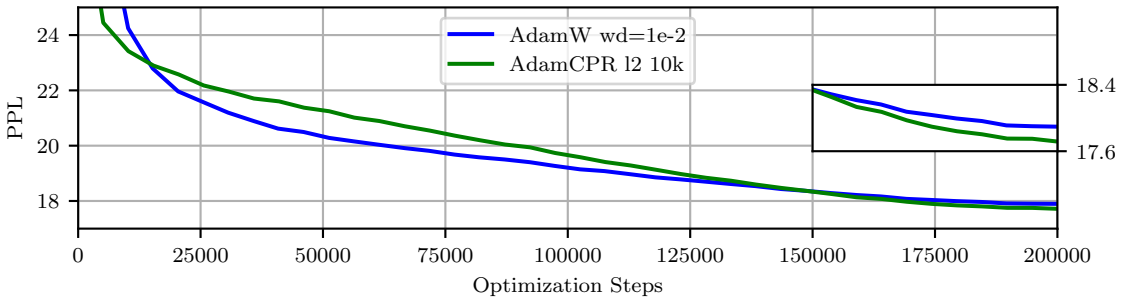

Figure E.1: Experiments on OpenWebText and a GPT2s model. The validation PPL of AdamW and the best CPR are displayed in blue and green respectively. We can notice that CPR regularized the training more at the beginning of the training.

Table 4: Hyperparameters of the language modeling task (GPT2 and Openwebtext).

Parameter	**GPT2s 200k**	**GPT2s 400k**	**GPT2m**
GPUs		8	
Gradient Clip Val		1.0	
Max Steps	200k	400k	200k
Precision		bf16-mixed	
Seed		1234	
Optimizer		AdamW or AdamCPR	
Learning Rate		0.002	
Weight Decay		0.1	
Beta1		0.9	
Beta2		0.99	
Eps		1.0×10^{-9}	
Stat Measure		L_2 norm	
Kappa		None, $0.005 \cdots 0.16$	
Kappa Factor		False, $4 \cdots 256$	
Lagmul Rate		1.0	
Kappa Adapt		True / False	
Kappa Init After Steps		False, $250 \cdots 16k$	
Bias Weight Decay		False	
Normalization Weight Decay		False	
Lr Num Warmup Steps		4000	
Lr Decay Factor		0.1	
Lr Schedule		Cosine	
Deepspeed Stage		2	
Model Dimension	768	768	1024
Number of Layers	12	768	24
Number of Heads	12	12	16
Fed Forward Dim	3072	3072	4048
Attn Dropout		0.1	
Resi Dropout		0.1	
Embed Dropout		0.1	
Rotary Pos Embed		True	
Rotary Emb Fraction		0.5	
Softmax Scale		True	
Key Dim Scaler		True	
Gating		False	
Use Glu		False	
Use Bias		True	
Flash Attn		True	
Initializer		Xavier Uniform	
Dataset Name		Openwebtext	
Max Sample Len		1024	
Batch Size	32	32	24
Val Ratio		0.0005	