Mind the Gap: Removing the Discretization Gap in Differentiable Logic Gate Networks

Shakir Yousefi [1] **Andreas Plesner** [1] **Till Aczel** [1] **Roger Wattenhofer** [1]

Abstract

Modern neural networks exhibit state-of-the-art performance on many benchmarks, but their high computational requirements and energy usage have researchers exploring more efficient solutions for real-world deployment. Logic gate networks (LGNs) learns a large network of logic gates for efficient image classification. However, learning a network that can solve a simple problem like CIFAR-10 can take days to weeks to train. Even then, almost half of the network remains unused, causing a *discretization gap*. This discretization gap hinders real-world deployment of LGNs, as the performance drop between training and inference negatively impacts accuracy. We inject Gumbel noise with a straight-through estimator during training to significantly speed up training, improve neuron utilization, and decrease the discretization gap. We theoretically show that this results from implicit Hessian regularization, which improves the convergence properties of LGNs. We train networks $4.5\times$ faster in wall-clock time, reduce the discretization gap by 98%, and reduce the number of unused gates by 100%.

1. Introduction

Deep neural networks achieve human-level performance on many tasks, but their high computational cost limits real-world deployment. This has sparked interest in models that retain accuracy while being more efficient. At their core, all digital computations reduce to Boolean operations (AND, OR, NOT, etc.). Motivating the question: *Can we express and execute machine learning models directly in the native language of hardware—namely, logic gates?* Logic Gate Networks (LGNs) offer one such approach by replacing arithmetic with compositions of discrete logic operations. While LGNs enable efficient inference, training them is difficult. Differentiable LGNs address this by introducing continuous relaxations that allow gradient-based training (Petersen et al., 2022; 2024).

We identify and propose solutions to two major challenges. **(1) Discretization gap**: The final model must be discretized after training, often leading to a significant accuracy drop (3%). **(2) Slow convergence**: Despite efficient inference, training is slow due to reliance on differentiable relaxations, making convergence slower than in standard neural networks. These challenges are interrelated.

The gap arises because the final parameters, after training, must be discretized. Small parameter perturbations can significantly change performance if the loss landscape is sharp. A sharp loss landscape can also cause poor gradient signals, which impact the convergence speed, causing training to take much longer, while a smooth loss landscape can reduce the discretization gap and speed up convergence (Foret et al., 2021; Chen & Hsieh, 2021). Our central hypothesis is that *smoother loss landscapes* make LGN models more robust to discretization and facilitate faster and more stable training. Since the loss landscape is smoother, the gradient signal is better, and the networks converge faster. Also, the improved gradient signal causes more neurons to collapse, thus reducing the impact of discretization.

We propose Gumbel Logic Gate Networks (Gumbel LGNs), which use the Gumbel-Softmax trick to inject noise into gate selection during training. This encourages exploration, smooths the loss landscape, and reduces the training-inference gap. Empirically, Gumbel LGNs converge faster and have smaller discretization gaps than standard Differentiable LGNs. To further reduce this gap, we adopt a discretization-aware training inspired by NAS: applying continuous relaxations only

[1]ETH Zurich, Zurich, Switzerland. Correspondence to: Andreas Plesner <aplesner@ethz.ch>, Till Aczel <taczel@ethz.ch>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

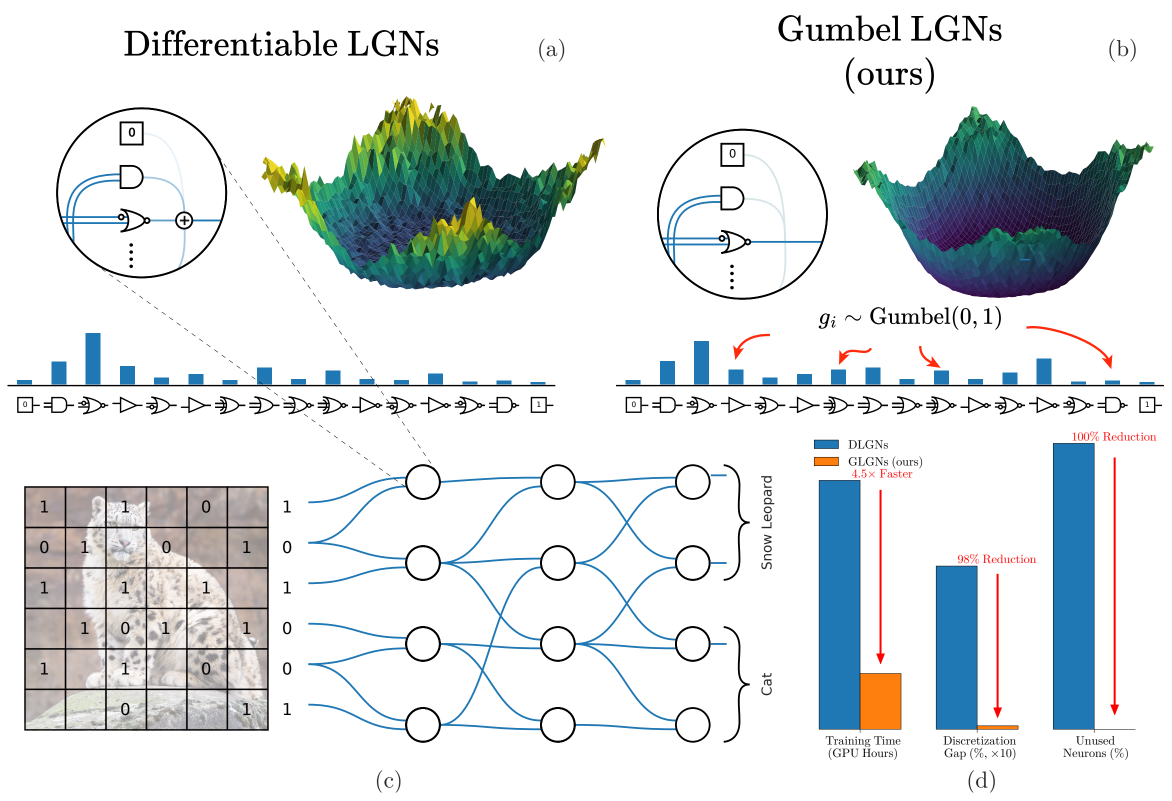

Figure 1. Overview figure. **(a)** Differentiable LGNs: Each node weighs and sums outputs of 16 logic gates, creating a brittle loss landscape that slows training and increases the discretization gap. **(b)** Gumbel LGNs: Injecting Gumbel noise and selecting the top gate smooths the loss landscape and aligns training with inference, improving convergence and reducing the gap. **(c)** Structure of Differentiable LGNs and Gumbel LGNs. Each neuron takes two inputs; final layer nodes are summed to produce class scores. **(d)** Gumbel LGNs yield up to $4.5\times$ faster convergence, 98% lower discretization gap, and elimination of unused neurons.

in the backward pass and using discrete gates in the forward pass. While the straight-through estimator may slightly slow convergence, it greatly reduces the gap and aligns training with inference without impacting inference speed.

To our knowledge, this is the first work to analyze the discretization gap in LGNs and connect it to loss landscape smoothness. Our approach scales to parameter spaces comparable to deep networks, exceeding $10^{3,600,000}$—vastly beyond NAS benchmarks, which typically reach up to 10^{18}.

Our contributions are as follows:
Empirical validation: We demonstrate that Gumbel LGNs train faster and improve neuron utilization.
Theoretical analysis: We prove that injecting Gumbel noise into Differentiable LGNs smooths their loss landscape by regularizing the Hessian's trace, thereby reducing the discretization gap and accelerating convergence.
Practical algorithmic insight: We show that using the straight-through estimator further reduces the discretization gap.

Extended background, related work, and Gumbel LGN details are in Appendices A to C.

2. Background

Logic Gate Networks Logic Gate Networks (LGNs) represent an entire network as a composition of discrete logic operations. In a learned network, each neuron in a hidden layer takes as input the value of two neurons (with output a and b) in the previous layer[1] and applies a fixed logic gate $h_i(a, b)$ to get its output. The final layer neurons are partitioned into k disjoint groups G_i. Following Petersen et al. (2022), we implement GroupSum, which computes class scores as $s_i = \frac{1}{\tau^{\text{GS}}} \sum_{j \in G_i} a_j$, where a_j is the binary activation of neuron j and τ^{GS} is a GroupSum temperature parameter. The class with the highest neuron activation count is predicted.

[1]At initialization, each neuron randomly picks which two neurons in the previous layer it uses for its inputs.

From Combinatorial Search to Differentiable Training Directly searching for the best discrete gate assignments is infeasible due to the size of the search space, so Differentiable Logic Gate Networks (Differentiable LGNs) (Petersen et al., 2022; 2024) introduce a continuous relaxation. Each of the 16 possible binary gates $h_i(a, b)$ is replaced by a continuous surrogate (e.g. $\mathrm{AND}(a, b) \mapsto a \cdot b$). In addition, each neuron maintains logits $\mathbf{z} \in \mathbb{R}^{16}$, which are initialized using a Gaussian, $\mathbf{z} \sim \mathcal{N}(0, 1)^{16}$. After a softmax, these logits define a probability distribution over gates, and the neuron's output is a weighted sum of the 16 gates:

$$f_{\mathbf{z}}^{\mathrm{soft}}(a, b) = \sum_{i=1}^{16} \frac{\exp z_i}{\sum_j \exp z_j} \cdot h_i(a, b). \tag{1}$$

Ensuring each logic gate maps from a continuous domain $f : [0, 1]^2 \to [0, 1]$. This "soft" network can be trained end-to-end with gradient descent.

Discretization After training, Differentiable LGNs are discretized to LGNs by selecting the logic gate with the highest logit value, i.e., it uses $h_i, i = \arg\max_i z_i$. We denote Differentiable LGNs evaluated in the differentiable setting (using Equation (1)) as *soft* and otherwise as *discrete*.

3. Related Work

Efficient Neural Architectures A significant body of research has focused on neural models that balance high performance with limited computational budgets, enabling edge deployment (Liu et al., 2021; Mishra & Gupta, 2024; Iqbal et al., 2024). Techniques include lookup tables (Chatterjee, 2018), binary and quantized networks (Frantar et al., 2022; Yuan & Agaian, 2023), and sparse networks (Hoefler et al., 2021; Frantar & Alistarh, 2023; Cheng et al., 2024).

Differentiable Logic Gate Networks (LGNs) recently achieved state-of-the-art results in image classification (Petersen et al., 2024). As the convolutional variant's code is unavailable, we focus on the original LGN (Petersen et al., 2022). Our improvements target convergence and are orthogonal to architectural innovations, thus likely transferable. We omit comparisons to other efficient models, already covered by Petersen et al. (2022; 2024).

Differentiable Neural Architecture Search Neural Architecture Search (NAS) automates selecting high-performing architectures (Zoph & Le, 2017), with efficiency improvements over early costly methods (Dong & Yang, 2019; Xie et al., 2020). A seminal contribution in this domain is Differentiable Architecture Search (DARTS) by Liu et al. (2019), which introduces a softmax-based relaxation over discrete architectural choices, allowing end-to-end optimization through gradient descent. More recently, Chen & Hsieh (2021) reduced the discretization gap of DARTS by introducing Smooth DARTS, which uses weight perturbations through uniform noise or adversarial optimization. These were shown to bias the optimization toward solutions with flatter minima and lower Hessian norm.

Sharpness-Aware Minimization A parallel line of research focuses on improving generalization by minimizing the sharpness of the loss landscape. Motivated by prior theoretical works on generalization and flat minima (Keskar et al., 2017; Dziugaite & Roy, 2017; Jiang* et al., 2019), Foret et al. (2021) introduced Sharpness-Aware Minimization (SAM). This technique explicitly seeks flat minima by optimizing the worst-case loss within a parameter perturbation neighborhood.

4. Empirical Evaluations

Our empirical evaluations focus on CIFAR-10, as the MNIST-like datasets (MNIST, FashionMNIST, etc.) have low discretization gaps (cf. Appendix O). CIFAR-10 and MNIST were also the datasets Petersen et al. (2022; 2024) mostly focused on. Due to constrained resources, we limit experiments by default to 48 GPU hours. We use the hyperparameters from (Petersen et al., 2022; 2024) whenever possible rather than tuning the parameters, such as learning rate, ourselves. Appendix N.1 contains all the default parameters.

Discretization Gap On Figure 2, the test accuracy as a function of training iteration for an LGN of depth 12 and width 256k on CIFAR-10; these are the default parameters unless stated otherwise. To quantify the discretization gap, we take the absolute difference between the discretized and soft network accuracy as shown on the right in Figure 2. Gumbel LGN converges much faster than the Differentiable LGN, with virtually no discretization gap. Combined with runtime results

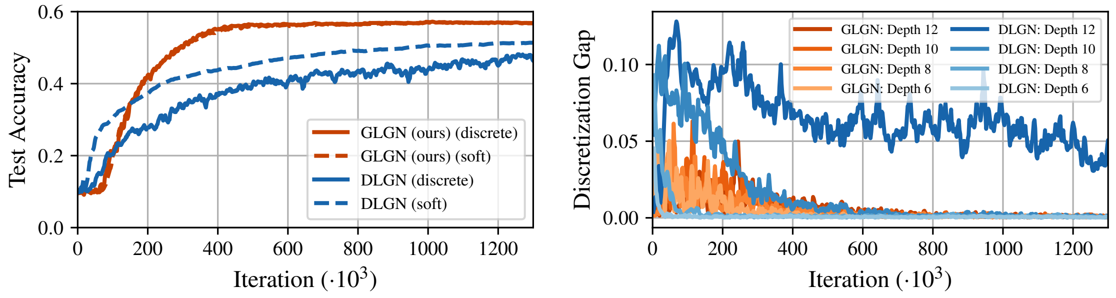

Figure 2. Performance of Gumbel LGNs and Differentiable LGNs on CIFAR-10. **Left:** Test accuracy for the default network with 12 layers and a width of 256k. **Right:** Discretization gap for various depths. Differentiable LGNs experience larger gaps and slower reduction as the depth increases. In contrast, Gumbel LGNs have consistently low gaps and fast reduction as the network depth increases.

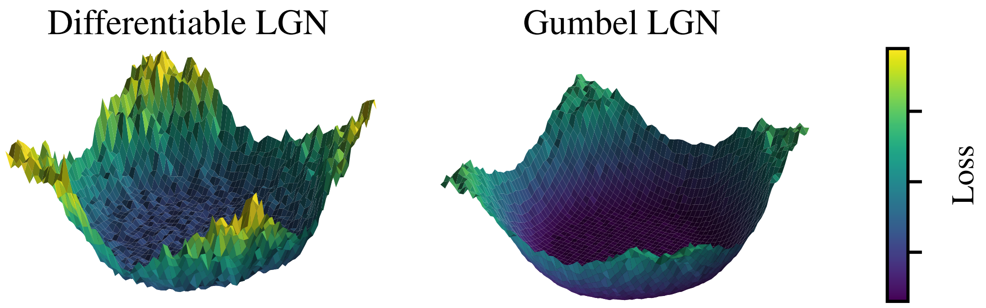

Figure 3. Visualization of loss landscapes. **Left:** Loss landscape of a Differentiable LGN. We see that the landscape is overall noisy. **Right:** Loss landscape of a Gumbel LGN with $\tau = 1.0$. We observe a much smoother loss landscape compared to the Differentiable LGNs.

from Table 4 in Appendix Q, Gumbel LGN converges[2] $4.75\times$ faster in iterations, making Gumbel LGNs $4.5\times$ faster in wall-clock time to train. Note that the Differentiable LGN still improves after 48 hours.

Gap Scaling with Depth On the right of Figure 2, we see the discretization gap for models of various depths for Differentiable LGNs and Gumbel LGNs. As the model depth increases, the expressive power of the networks theoretically increases. The Differentiable LGNs experience bigger discretization gaps as the depth increases, while our Gumbel LGNs are stable across depths.

Curvature Visualization We project the high-dimensional parameter space onto two-dimensional subspaces to assess the loss of landscape curvature. Following Li et al. (2018), we select random directions and interpolate the loss surface along these axes, providing insight into the optimization landscape's geometry around learned solutions. Visualization details are in Appendix M. Figure 3 shows that Gumbel LGN has a visually smoother loss surface.

5. Conclusion

We introduced Gumbel logic gate networks (Gumbel LGNs), addressing two critical limitations of Differentiable LGNs: slow convergence during training and a large discretization gap between training and inference. Our theoretical analysis shows that Gumbel noise during gate selection promotes flatter minima by implicitly minimizing the Hessian trace, reducing sensitivity to parameter discretization. Experiments on CIFAR-10 demonstrate that Gumbel LGNs converge $4.5\times$ faster in wall-clock time than Differentiable LGNs while reducing the discretization gap by 98% and achieving 100.0% improvement in neuron utilization. These advantages become more pronounced with depth, indicating favorable scaling properties. Our improvements are dataset and architecture-independent, and several promising directions remain for future exploration: extending Differentiable LGNs to convolutional architectures, validating on more complex datasets like CIFAR-100 and ImageNet32, exploring hardware-specific optimizations, and investigating adaptive temperature scheduling.

[2]For this, we match Gumbel LGNs' discrete accuracy with Differentiable LGNs' maximum discrete accuracy.

References

Al-Quraan, M., Mohjazi, L., Bariah, L., Centeno, A., Zoha, A., Arshad, K., Assaleh, K., Muhaidat, S., Debbah, M., and Imran, M. A. Edge-Native Intelligence for 6G Communications Driven by Federated Learning: A Survey of Trends and Challenges. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 7(3):957–979, June 2023. ISSN 2471-285X. doi: 10.1109/TETCI.2023.3251404.

Andriushchenko, M. and Flammarion, N. Towards Understanding Sharpness-Aware Minimization, June 2022.

Avron, H. and Toledo, S. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):1–34, 2011.

Bottou, L., Cortes, C., Denker, J., Drucker, H., Guyon, I., Jackel, L., LeCun, Y., Muller, U., Sackinger, E., Simard, P., and Vapnik, V. Comparison of classifier methods: A case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3 - Conference C: Signal Processing (Cat. No.94CH3440-5)*, volume 2, pp. 77–82 vol.2, October 1994. doi: 10.1109/ICPR.1994.576879.

Chang, J., zhang, x., Guo, Y., MENG, GAOFENG., XIANG, SHIMING., and Pan, C. DATA: Differentiable ArchiTecture approximation. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Chatterjee, S. Learning and Memorization. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 755–763. PMLR, July 2018.

Chen, X. and Hsieh, C.-J. Stabilizing Differentiable Architecture Search via Perturbation-based Regularization, January 2021.

Cheng, H., Zhang, M., and Shi, J. Q. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Chitty-Venkata, K. T., Emani, M., Vishwanath, V., and Somani, A. K. Neural Architecture Search Benchmarks: Insights and Survey. *IEEE Access*, 11:25217–25236, 2023. ISSN 2169-3536. doi: 10.1109/ACCESS.2023.3253818.

Chrabaszcz, P., Loshchilov, I., and Hutter, F. A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets. *CoRR*, January 2017.

Chu, X., Zhou, T., Zhang, B., and Li, J. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search, July 2020.

Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., and Ha, D. Deep Learning for Classical Japanese Literature, November 2018.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. EMNIST: An extension of MNIST to handwritten letters, March 2017.

Dong, X. and Yang, Y. Searching for a Robust Neural Architecture in Four GPU Hours. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1761–1770, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-7281-3293-8. doi: 10.1109/CVPR.2019.00186.

Du, J., Yan, H., Feng, J., Zhou, J. T., Zhen, L., Goh, R. S. M., and Tan, V. Y. F. Efficient Sharpness-aware Minimization for Improved Training of Neural Networks, May 2022a.

Du, J., Zhou, D., Feng, J., Tan, V., and Zhou, J. T. Sharpness-aware training for free. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 23439–23451. Curran Associates, Inc., 2022b.

Dziugaite, G. K. and Roy, D. M. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data, October 2017.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20 (55):1–21, 2019.

Foret, P., Kleiner, A., Mobahi, H., and Neyshabur, B. Sharpness-Aware Minimization for Efficiently Improving Generalization, April 2021.

Frantar, E. and Alistarh, D. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Gumbel, E. J. Statistical Theory of Extreme Values and Some Practical Applications. *Journal of the Royal Statistical Society. Series A (General)*, 118(1):106, 1955. ISSN 00359238. doi: 10.2307/2342529.

He, Y. and Xiao, L. Structured pruning for deep convolutional neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 46(5):2900–2919, 2023.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021.

Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Ipsen, I. C. Computing an eigenvector with inverse iteration. *SIAM review*, 39(2):254–291, 1997.

Iqbal, S., Khan, T. M., Naqvi, S. S., Naveed, A., Usman, M., Khan, H. A., and Razzak, I. LDMRes-Net: A Lightweight Neural Network for Efficient Medical Image Segmentation on IoT and Edge Devices. *IEEE Journal of Biomedical and Health Informatics*, 28(7):3860–3871, July 2024. ISSN 2168-2208. doi: 10.1109/JBHI.2023.3331278.

Jang, E., Gu, S., and Poole, B. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, February 2017.

Jiang*, Y., Neyshabur*, B., Mobahi, H., Krishnan, D., and Bengio, S. Fantastic Generalization Measures and Where to Find Them. In *International Conference on Learning Representations*, September 2019.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations*, February 2017.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images, 2009.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998. ISSN 1558-2256. doi: 10.1109/5.726791.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Li, T., Zhou, P., He, Z., Cheng, X., and Huang, X. Friendly Sharpness-Aware Minimization. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5631–5640, Seattle, WA, USA, June 2024. IEEE. ISBN 979-8-3503-5300-6. doi: 10.1109/CVPR52733.2024.00538.

Li, Y., Pintea, S.-L., and Gemert, J. C. V. Equal Bits: Enforcing Equally Distributed Binary Network Weights. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(2):1491–1499, June 2022. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v36i2.20039.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable Architecture Search, April 2019.

Liu, S., Ha, D. S., Shen, F., and Yi, Y. Efficient neural networks for edge devices. *Computers & Electrical Engineering*, 92: 107121, June 2021. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2021.107121.

Liu, Y., Mai, S., Chen, X., Hsieh, C.-J., and You, Y. Towards Efficient and Scalable Sharpness-Aware Minimization. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12350–12360, New Orleans, LA, USA, June 2022a. IEEE. ISBN 978-1-6654-6946-3. doi: 10.1109/CVPR52688.2022.01204.

Liu, Y., Mai, S., Cheng, M., Chen, X., Hsieh, C.-J., and You, Y. Random sharpness-aware minimization. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24543–24556. Curran Associates, Inc., 2022b.

Maddison, C. J., Tarlow, D., and Minka, T. A* Sampling. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

McAllester, D. A. PAC-Bayesian model averaging. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pp. 164–170, Santa Cruz California USA, July 1999. ACM. ISBN 978-1-58113-167-3. doi: 10.1145/307400.307435.

Mi, P., Shen, L., Ren, T., Zhou, Y., Sun, X., Ji, R., and Tao, D. Make sharpness-aware minimization stronger: A sparsified perturbation approach. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 30950–30962. Curran Associates, Inc., 2022.

Miotti, P., Niklasson, E., Randazzo, E., and Mordvintsev, A. Differentiable Logic CA: From Game of Life to Pattern Generation. https://google-research.github.io/self-organising-systems/difflogic-ca/, March 2025.

Mises, R. V. and Pollaczek-Geiringer, H. Praktische Verfahren der Gleichungsauflösung Section 1-4. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 9(1):58–77, 1929. ISSN 1521-4001. doi: 10.1002/zamm.19290090105.

Mishra, R. and Gupta, H. P. Designing and Training of Lightweight Neural Networks on Edge Devices Using Early Halting in Knowledge Distillation. *IEEE Transactions on Mobile Computing*, 23(5):4665–4677, May 2024. ISSN 1558-0660. doi: 10.1109/TMC.2023.3297026.

Mueller, M., Vlaar, T., Rolnick, D., and Hein, M. Normalization Layers Are All That Sharpness-Aware Minimization Needs, November 2023.

Park, D., Kim, S., An, Y., and Jung, J.-Y. LiReD: A Light-Weight Real-Time Fault Detection System for Edge Computing Using LSTM Recurrent Neural Networks. *Sensors*, 18(7):2110, June 2018. ISSN 1424-8220. doi: 10.3390/s18072110.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural Computation*, 6(1):147–160, 1994.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Deep Differentiable Logic Gate Networks. In *Advances in Neural Information Processing Systems*, October 2022.

Petersen, F., Kuehne, H., Borgelt, C., Welzel, J., and Ermon, S. Convolutional differentiable logic gate networks. In Globerson, A., Mackey, L., Belgrave, D., Fan, A., Paquet, U., Tomczak, J., and Zhang, C. (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 121185–121203. Curran Associates, Inc., 2024.

Ren, P., Xiao, Y., Chang, X., Huang, P.-y., Li, Z., Chen, X., and Wang, X. A Comprehensive Survey of Neural Architecture Search: Challenges and Solutions. *ACM Comput. Surv.*, 54(4):76:1–76:34, May 2021. ISSN 0360-0300. doi: 10.1145/3447582.

Sagun, L., Bottou, L., and LeCun, Y. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Sun, M., Liu, Z., Bair, A., and Kolter, J. Z. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Tian, Y., Liu, C., Xie, L., jiao, J., and Ye, Q. Discretization-aware architecture search. *Pattern Recognition*, 120:108186, December 2021. ISSN 0031-3203. doi: 10.1016/j.patcog.2021.108186.

Trefethen, L. N. and Bau, D. *Numerical Linear Algebra*. SIAM, 2022.

Tu, R., Roberts, N., Khodak, M., Shen, J., Sala, F., and Talwalkar, A. NAS-Bench-360: Benchmarking Neural Architecture Search on Diverse Tasks. *Advances in Neural Information Processing Systems*, 35:12380–12394, December 2022.

Wang, P., Zhang, Z., Lei, Z., and Zhang, L. Sharpness-Aware Gradient Matching for Domain Generalization. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3769–3778, Vancouver, BC, Canada, June 2023. IEEE. ISBN 979-8-3503-0129-8. doi: 10.1109/CVPR52729.2023.00367.

Wei, M. and Schwab, D. J. How noise affects the hessian spectrum in overparameterized neural networks. *arXiv preprint arXiv:1910.00195*, 2019.

Wen, K., Ma, T., and Li, Z. How Does Sharpness-Aware Minimization Minimize Sharpness?, January 2023.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms, September 2017.

Xie, S., Zheng, H., Liu, C., and Lin, L. SNAS: Stochastic Neural Architecture Search, April 2020.

Yadav, C. and Bottou, L. Cold Case: The Lost MNIST Digits, November 2019.

Yuan, C. and Agaian, S. S. A comprehensive review of Binary Neural Network. *Artificial Intelligence Review*, 56(11): 12949–13013, November 2023. ISSN 0269-2821, 1573-7462. doi: 10.1007/s10462-023-10464-w.

Zela, A., Siems, J., and Hutter, F. NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search. In *International Conference on Learning Representations*, September 2019.

Zhang, Y., Pan, J., Liu, X., Chen, H., Chen, D., and Zhang, Z. FracBNN: Accurate and FPGA-Efficient Binary Neural Networks with Fractional Activations. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '21, pp. 171–182, New York, NY, USA, February 2021. Association for Computing Machinery. ISBN 978-1-4503-8218-2. doi: 10.1145/3431920.3439296.

Zhang, Y., Qin, Y., Zhang, Y., Zhou, X., Jian, S., Tan, Y., and Li, K. OnceNAS: Discovering efficient on-device inference neural networks for edge devices. *Information Sciences*, 669:120567, May 2024. ISSN 00200255. doi: 10.1016/j.ins.2024.120567.

Zoph, B. and Le, Q. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*, February 2017.

A. Background

Logic Gate Networks Logic Gate Networks (LGNs) represent an entire network as a composition of discrete logic operations. In a learned network, each neuron in a hidden layer takes as input the value of two neurons (with output a and b) in the previous layer[3] and applies a fixed logic gate $h_i(a, b)$ to get its output. The final layer neurons are partitioned into k disjoint groups G_i. Following Petersen et al. (2022), we implement GroupSum, which computes class scores as $s_i = \frac{1}{\tau^{\text{GS}}} \sum_{j \in G_i} a_j$, where a_j is the binary activation of neuron j and τ^{GS} is a GroupSum temperature parameter. The class with the highest neuron activation count is predicted.

From Combinatorial Search to Differentiable Training Directly searching for the best discrete gate assignments is infeasible due to the size of the search space, so Differentiable Logic Gate Networks (Differentiable LGNs) (Petersen et al., 2022; 2024) introduce a continuous relaxation. Each of the 16 possible binary gates $h_i(a, b)$ is replaced by a continuous surrogate (e.g. AND$(a, b) \mapsto a \cdot b$). In addition, each neuron maintains logits $\mathbf{z} \in \mathbb{R}^{16}$, which are initialized using a Gaussian, $\mathbf{z} \sim \mathcal{N}(0, 1)^{16}$. After a softmax, these logits define a probability distribution over gates, and the neuron's output is a weighted sum of the 16 gates:

$$f_{\mathbf{z}}^{\text{soft}}(a, b) = \sum_{i=1}^{16} \frac{\exp z_i}{\sum_j \exp z_j} \cdot h_i(a, b). \tag{2}$$

Ensuring each logic gate maps from a continuous domain $f : [0, 1]^2 \to [0, 1]$. This "soft" network can be trained end-to-end with gradient descent.

Discretization After training, Differentiable LGNs are discretized to LGNs by selecting the logic gate with the highest logit value, i.e., it uses $h_i, i = \arg\max_i z_i$. We denote Differentiable LGNs evaluated in the differentiable setting (using Equation (2)) as *soft* and otherwise as *discrete*.

Gumbel-Softmax The Gumbel-Softmax trick offers an efficient and effective way to draw samples from a categorical distribution with class probabilities $\pi \in \Delta^k$ (Gumbel, 1955; Maddison et al., 2014; Jang et al., 2017; Maddison et al., 2017). Let $g \sim \text{Gumbel}(0, 1)$ distribution if $u \sim U(0, 1)$ and $g = -\log(-\log u)$. We can then draw a sample z from π as the value for index i given by Equation (3).

$$i = \arg\max_j (g_j + \log \pi_j), \quad g_j \sim \text{Gumbel}(0, 1). \tag{3}$$

We can make the argmax operation continuous and differentiable with respect to the class probabilities π_i, and generate k-dimensional sample vectors $y \in \mathbb{R}^k$ using a softmax with temperature τ as below:

$$\pi_i^{\text{Gumbel}} = \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_j \exp((\log \pi_j + g_j)/\tau)}, \quad \pi_i = \sum_{i=1}^{k} \frac{\exp z_i}{\sum_j \exp z_j}, \quad z_i \in \mathbb{R}. \tag{4}$$

B. Related Work

Efficient Neural Architectures A significant body of research has focused on designing neural models that maintain high performance while operating within limited computational budgets, e.g., for deployment on edge devices (Park et al., 2018; Liu et al., 2021; Al-Quraan et al., 2023; Mishra & Gupta, 2024; Zhang et al., 2024; Iqbal et al., 2024). These light models use various methods such as lookup tables (Chatterjee, 2018), binary and quantized neural networks (Zhang et al., 2021; Li et al., 2022; Frantar et al., 2022; Yuan & Agaian, 2023), and sparse neural networks (Hoefler et al., 2021; Sun et al., 2023; He & Xiao, 2023; Frantar & Alistarh, 2023; Cheng et al., 2024).

Of particular interest in this context are Differentiable Logic Gate Networks (LGNs), which have recently demonstrated state-of-the-art performance in image classification tasks (Petersen et al., 2022; 2024), as well as in rule extraction from observed cellular automata dynamics (Miotti et al., 2025). The convolutional variant's code is not public; we therefore focus on the original LGN Petersen et al. (2022). Our proposed improvements target convergence behavior and are orthogonal to the architectural innovations of the convolutional LGN variant; hence, we expect them to be transferable without loss of generality. We refrain from comparisons to other efficient neural models, as such benchmarks were already comprehensively addressed in the works mentioned above by Petersen et al. (2022; 2024).

[3] At initialization, each neuron randomly picks which two neurons in the previous layer it uses for its inputs.

Differentiable Neural Architecture Search Neural Architecture Search (NAS) aims to automate the selection of high-performing model architectures from a large design space (Zoph & Le, 2017; Elsken et al., 2019; Ren et al., 2021). While early approaches were computationally expensive, subsequent efforts have focused on improving efficiency (Dong & Yang, 2019; Xie et al., 2020). Several works have addressed the issue of train–test performance discrepancies by proposing sampling-based training (Chang et al., 2019) or regularization techniques that bias architecture selection toward configurations with better generalization (Chu et al., 2020).

A seminal contribution in this domain is Differentiable Architecture Search (DARTS) by Liu et al. (2019), which introduces a softmax-based relaxation over discrete architectural choices, allowing end-to-end optimization through gradient descent. This principle strongly resonates with the soft gate selection mechanism employed in LGNs.

More recently, Chen & Hsieh (2021) reduced the discretization gap of DARTS by introducing Smooth DARTS, which uses weight perturbations through uniform noise or adversarial optimization. These were shown to bias the optimization toward solutions with flatter minima and lower Hessian norm. This technique—often referred to as *curvature regularization*—reduces sensitivity to sharp local optima and enhances generalization.

Differentiable LGNs as DARTS The works on LGNs by Petersen et al. (2022; 2024) do not explicitly draw connections to NAS, but the conceptual similarity is high. Both LGNs and DARTS use softmax-based weighting to choose between multiple candidate functions in a differentiable manner. A key distinction lies in the scale of the search space. Conventional NAS approaches typically explore search space sizes up to 10^{18} (Zela et al., 2019; Tu et al., 2022; Chitty-Venkata et al., 2023),while LGNs operate over exponentially larger spaces—$16^{6\cdot 64,000} \approx 10^{462,382}$ for MNIST and $\approx 10^{3,699,056}$ for CIFAR-10. This scale is enabled by the simplicity of logic operations, which have no learnable parameters. Thus, LGNs demonstrate the viability of differentiable NAS at previously unexplored scales.

Conventional NAS frameworks often permit a retraining phase after discretizing the architecture, thereby reducing the discretization gap. LGNs, in contrast, lack such flexibility, as their neurons contain no parameterized operations, and thus the gap persists.

Moreover, in DARTS and related approaches, this training approach favors operations, such as residual connections (Tian et al., 2021). Typically, we aim to avoid these residual connections, as they do not increase the models' expressive power (Chu et al., 2020). In relation, Petersen et al. (2024) finds that their convolutional method with residual initialization mainly converges to residual connections.

Sharpness-Aware Minimization A parallel line of research focuses on improving generalization by minimizing the sharpness of the loss landscape. Motivated by prior theoretical works on generalization and flat minima (Keskar et al., 2017; Dziugaite & Roy, 2017; Jiang* et al., 2019), Foret et al. (2021) introduced Sharpness-Aware Minimization (SAM) in Equation (5). This technique explicitly seeks flat minima by optimizing the worst-case loss within a perturbation neighborhood.

$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{\|\boldsymbol{\epsilon}|_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}), \tag{5}$$

where $L_{\mathcal{S}}(\boldsymbol{w})$ is a loss function over a training set $\mathcal{S}$ of training samples evaluated for model parameters $\boldsymbol{w}$. $p \in [1, \infty[$ is the p-norm used (usually $p = 2$) and $\rho > 0$ is a hyperparameter (Foret et al., 2021). Since its introduction, SAM has inspired numerous follow-up studies focused on improving computational efficiency (Liu et al., 2022a; Du et al., 2022a;b; Mi et al., 2022; Liu et al., 2022b; Mueller et al., 2023) as well as providing theoretical insights into its efficacy (Andriushchenko & Flammarion, 2022; Wang et al., 2023; Wen et al., 2023; Li et al., 2024). We refer to Appendix J for a detailed description of SAM.

C. Gumbel Logic Gate Networks

We introduce *Gumbel Logic Gate Networks* (Gumbel LGNs), which employ discrete sampling of logic gates via the Gumbel-Softmax trick with a straight-through (ST) estimator (Jang et al., 2017; Maddison et al., 2017). While conventional Differentiable LGNs maintain a convex combination of gates throughout training and prune to hard selections at inference time, Gumbel LGNs resemble inference-time behavior directly during training by stochastically selecting individual gates per forward pass. We perturb the gate logits with Gumbel noise and select the most probable gate, following the argmax operation (cf. Equation (3)). During backpropagation, the non-differentiable argmax is approximated using the Gumbel-Softmax (Equation (4)), enabling end-to-end training.

This approach is motivated by two key observations: (1) Implicit smoothening via noise: Injecting Gumbel noise during the forward pass introduces a form of stochastic smoothing, effectively averaging over local perturbations of the loss surface. As we show, this process approximates a curvature-penalizing loss that favors flatter minima and smaller Hessian norm. In addition, this is known to correlate with improved generalization (Foret et al., 2021; Chen & Hsieh, 2021). (2) Inference-time alignment: In Differentiable LGNs, the training objective is misaligned with inference behavior, as training relies on weighted combinations of gates that are ultimately discarded. This discrepancy harms generalization. In contrast, Gumbel LGNs train under the same discrete selection mechanism, which is used at inference.

Training Gumbel LGNs As with Differentiable LGNs, we model each neuron as a distribution over binary, relaxed logic gates $\mathcal{S} = \{h_1, h_2, \dots, h_{16}\}$, where each gate $h_i : [0,1]^2 \to [0,1]$ operates on relaxed Boolean inputs. We associate each gate i with logit z_i, and the gate has weight π_i^{Gumbel} from Equation (4). The output of the neuron with inputs (a, b) is then:

$$f_{\mathbf{z}}^{\text{soft}}(a,b) = \sum_{i=1}^{16} \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_j \exp((\log \pi_j + g_j)/\tau)} \cdot h_i(a,b) = \sum_{i=1}^{16} \pi_i^{\text{Gumbel}} \cdot h_i(a,b), \tag{6}$$

$\tau > 0$ is a temperature parameter controlling the sharpness of the distribution. As $\tau \to 0$, the distribution increasingly peaks around the maximum-logit index (Jang et al., 2017). This relaxation enables end-to-end differentiability while encouraging the network to commit to discrete logic gates during training.

We employ a straight-through (ST) estimator to bridge the discretization gap between the continuous relaxation used during training and the hard decisions required during inference. In this formulation, each neuron selects a single logic gate in the forward pass via a hard (non-differentiable) choice, while gradients are estimated through a soft relaxation in the backward pass. See Appendix K for pseudo-code implementation of the training process.

Concretely, during the forward pass, we sample Gumbel noise $\mathbf{g} \sim \text{Gumbel}(0,1)^{16}$ and compute:

$$f_{\mathbf{z}}^{\text{discrete}}(a,b) = h_k(a,b) \tag{7}$$

using the gate h_k with maximum perturbed logit. During the backward pass, we use the soft Gumbel-Softmax relaxation (Equation (6)) to compute gradients, effectively treating the hard output as if it were differentiable:

$$\frac{\partial f_{\mathbf{z}}^{\text{discrete}}}{\partial z_i} := \frac{\partial f_{\mathbf{z}}^{\text{soft}}}{\partial z_i}.$$

This ST estimator mechanism encourages the network to make discrete decisions and allows end-to-end optimization via backpropagation. See Figure 1 or Figure 8 in Appendix H for visualizations.

Implicit Gap Reduction via Gumbel Smoothing. We present a theoretical result that supports the use of Gumbel perturbations during training. Consider a loss function $\mathcal{L}$, logits $\mathbf{z} \in \mathbb{R}^{16}$, and $\mathbf{g}$ with i.i.d entries $g_i \sim \text{Gumbel}(0,1)$. Adding Gumbel noise with $\tau \in \mathbb{R}$ to the logits can be seen as Monte-Carlo sample of the objective $J(\mathbf{z})$;

$$J(\mathbf{z}) = \mathbb{E}\left[\mathcal{L}(\text{softmax}((\mathbf{z}+\mathbf{g})/\tau)\right].$$

This can be interpreted as a form of stochastic smoothing. This gives us the following lemma:

Lemma C.1 (Gumbel-Smoothing)**.** *Let $\mathcal{L} : \mathbb{R}^{16} \to \mathbb{R}$ be twice continuously differentiable (with Lipschitz Hessian), and let $\mathbf{z} \in \mathbb{R}^{16}, \mathbf{g} \sim \text{Gumbel}(0,1)^{16}$. Consider $J(\mathbf{z})$*

$$J(\mathbf{z}) = \mathbb{E}\left[\mathcal{L}(\text{softmax}((\mathbf{z}+\mathbf{g})/\tau)\right]$$

and set $\mathbf{a} = \mathbf{z}/\tau$ and $f(\mathbf{a}) = \mathcal{L}(\text{softmax}(\mathbf{a}))$, then

$$J(\mathbf{z}) = \mathcal{L}(\text{softmax}(\mathbf{z}/\tau)) + \frac{\pi^2}{12\tau^2}\text{tr}(H_f(\mathbf{z}/\tau)) + O(\tau^{-3}).$$

Proof. See Appendix D. □

Intuitively, by injecting Gumbel noise during training, we encourage the optimizer to find parameters that are robust to small perturbations. This results in flatter loss landscapes and reduces the sensitivity to parameter discretization when switching to inference mode. The expected loss scales as

$$\frac{\pi^2}{12\tau^2}\operatorname{tr}(H_f(\mathbf{z}/\tau)).$$

So as the temperature τ **increases**, the coefficient $1/\tau^2$ **decreases**, reducing the degree of implicit smoothing. We illustrate this with two representative choices:

- **Small** τ (e.g. 0.1) $\implies$ $1/\tau^2$ is large $\implies$ large smoothing, flat minima.
- **Large** τ (e.g. 2.0) $\implies$ $1/\tau^2$ is small $\implies$ almost no smoothing.

As a result, adjusting the temperature τ offers a mechanism to control the strength of this curvature-aware regularization, the convergence of the model, and to reduce the discretization gap *implicitly*.

D. Gumbel Smoothing

The proof of Lemma C.1 depends on the translation invariance of the softmax.

Lemma D.1 (Translation-Invariance of Softmax). *Consider logits $\mathbf{z} \in \mathbb{R}^d$, then adding any constant $c \in \mathbb{R}$ to $\mathbf{z}$, $\mathbf{z} + c$, does not change the output of the softmax. Concretely, for any logit z_i we have*

$$\operatorname{softmax}(z_i + c) = \operatorname{softmax}(z_i)$$

Proof. Denote $z_i' := z_i + c$, then writing the output of our softmax yields

$$\begin{aligned}\operatorname{softmax}(z_i') &= \frac{e^{z_i'}}{\sum_j e^{z_i'}} \\ &= \frac{e^{z_i+c}}{\sum_j e^{z_i+c}} \\ &= \frac{e^{z_i'}}{\sum_j e^{z_i'}} \\ &= \frac{e^c \cdot e^{z_i}}{\sum_j e^c \cdot e^{z_i}} \\ &= \frac{e^c \cdot e^{z_i}}{e^c \cdot \sum_j \cdot e^{z_i}} \\ &= \frac{e^{z_i}}{\sum_j e^{z_i}} = \operatorname{softmax}(z_i)\end{aligned}$$

□

Restating lemma Lemma C.1, we write:

Lemma D.2 (Gumbel-Smoothing). *Let $\mathcal{L} : \mathbb{R}^{16} \to \mathbb{R}$ be twice continuously differentiable (with Lipschitz Hessian), and let $\mathbf{z} \in \mathbb{R}^{16}, \mathbf{g} \sim \operatorname{Gumbel}(0,1)^{16}$. Consider $J(\mathbf{z})$*

$$J(\mathbf{z}) = \mathbb{E}\left[\mathcal{L}(\operatorname{softmax}((\mathbf{z} + \mathbf{g})/\tau)\right]$$

and set $\mathbf{a} = \mathbf{z}/\tau$ and $f(\mathbf{a}) = \mathcal{L}(\operatorname{softmax}(\mathbf{a}))$, then

$$J(\mathbf{z}) = \mathcal{L}(\operatorname{softmax}(\mathbf{z}/\tau)) + \frac{\pi^2}{12\tau^2}\operatorname{tr}(H_f(a)) + O(\tau^{-3}).$$

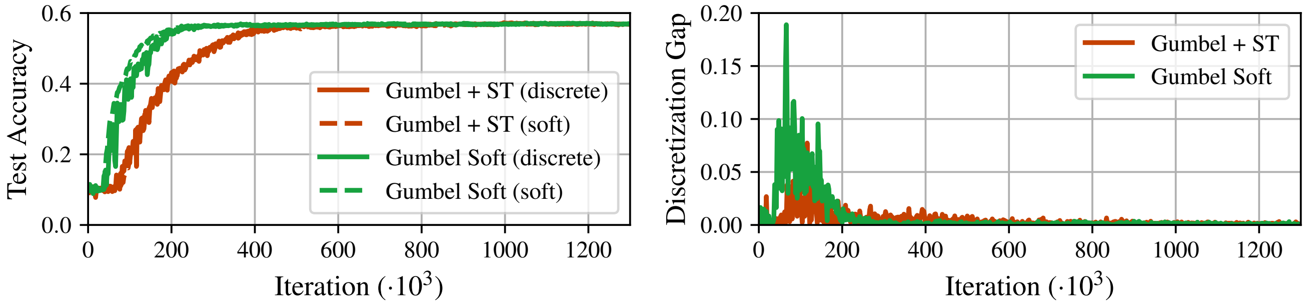

Figure 4. Straight-through (ST) estimator ablation. The Gumbel LGNs uses hard gate choices in the forward pass as shown in (3) called the ST estimator. On the left, we show the test accuracy over training iterations; on the right, we show the discretization gap. Gumbel LGNs with ST estimator converge slightly slower in test accuracy, but the discretization gap is smaller.

Proof. Rewriting $J(\mathbf{z})$ in terms of f gives us

$$J(\mathbf{z}) = \mathbb{E}\left[f\left(\mathbf{a} + \frac{\mathbf{g}}{\tau}\right)\right]$$

Consider a second-order Taylor expansion of f around $\mathbf{a}$

$$f\left(\mathbf{a} + \frac{\mathbf{g}}{\tau}\right) = f(\mathbf{a}) + \nabla f(\mathbf{a})^\top \left(\frac{\mathbf{g}}{\tau}\right) + \frac{1}{2}\left(\frac{\mathbf{g}}{\tau}\right)^\top H_f(\mathbf{a})\left(\frac{\mathbf{g}}{\tau}\right) + O(\|\mathbf{g}\|^3/\tau^3)$$

Taking expectations, and recalling that $\mathbb{E}[g_i] = \gamma,\ \mathrm{Var}(g_i) = \pi^2/6$, where $\gamma \approx 0.57721$ is the Euler-Mascheroni constant. we get

$$J(\mathbf{z}) = f(\mathbf{a}) + \left(\frac{\gamma}{\tau}\right)\nabla f(\mathbf{a})^\top \mathbf{1} + \frac{1}{2\tau^2}\left[\gamma^2 \mathbf{1}^\top H_f(\mathbf{a})\mathbf{1} + \frac{\pi^2}{6}\mathrm{tr}(H_f(\mathbf{a}))\right] + O(\tau^{-3})$$

which follows from $\mathbb{E}[g_i^2] = \mathrm{Var}(g_i) + \mathbb{E}[g_i]^2 = \pi^2/6 + \gamma^2$, $\mathbb{E}[g_i g_j] = \gamma^2$ for $i \neq j$.

$$\mathbb{E}[\mathbf{g}\mathbf{g}^\top] = \gamma^2 \mathbf{1}\mathbf{1}^\top + \frac{\pi^2}{6}I$$

and the following trace-lemma

$$\mathbb{E}\left[\mathbf{g}^\top H_f(\mathbf{a})\mathbf{g}\right] = \mathrm{tr}(H_f(\mathbf{a})\mathbb{E}[\mathbf{g}\mathbf{g}^\top]) = \gamma^2 \mathbf{1}^\top H_f(\mathbf{a})\mathbf{1} + \frac{\pi^2}{6}\mathrm{tr}(H_f(\mathbf{a}))$$

Since the softmax is translation-invariant in its input $\mathbf{a}$, we have $\nabla f(\mathbf{a})^\top \mathbf{1} = 0$ and $H_f(\mathbf{a})\mathbf{1} = 0$, so all terms depending on γ drop, finally giving us

$$J(\mathbf{z}) = \mathcal{L}(\mathrm{softmax}(\mathbf{z}/\tau)) + \frac{\pi^2}{12\tau^2}\mathrm{tr}(H_f(\mathbf{z}/\tau)) + O(\tau^{-3}).$$

□

Hence, minimizing our stochastic loss implicitly smoothens the curvature by minimizing the trace of the Hessian.

E. Ablations

E.1. Straight Through Estimator

To better understand the source of gap reduction achieved by the Gumbel-Softmax trick, we perform an ablation to isolate the contribution of the ST estimator. As discussed previously, Gumbel-Softmax combines (i) ST estimation as seen in

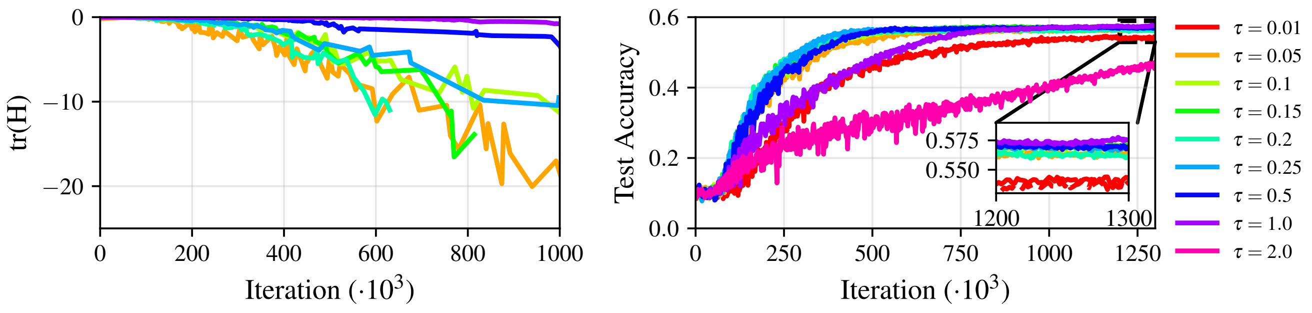

Figure 5. Ablation over the temperature τ for Gumbel LGNs. **Left:** Estimated Hessian trace using Hutchinson's method. The trace shrinks as τ decreases, indicating fewer large positive eigenvalues, thus suggesting a flatter loss surface, which may reduce the risk of loss increases when discretizing parameters. **Right:** Test accuracy for the τ-values. We see a goldilocks zone for the temperature, as if τ is large (> 1) or small (< 0.1), then the network converges much slower. In the zoomed-in view, we plot the non-discretized view as dashed lines and see that these are similar to the discretized values, i.e., the discretization gap is low for all. Except for $\tau = 0.01$ and $\tau = 2$, there is only a small variation in the final accuracy, as seen in Table 1.

Table 1. Maximum and final test accuracy for the tested τ values. The iterations column indicates the number of training iterations ($\cdot 10^3$) required to be within 1% of the maximum accuracy. We see that with a medium value of $\tau \approx 0.25$ the network converges much faster than for high values > 1.

τ	0.01	0.05	0.10	0.15	0.20	0.25	0.50	1.00	2.00
Max accuracy	0.547	0.566	0.574	0.574	0.566	0.573	0.573	0.578	0.490
Final accuracy	0.546	0.564	0.570	0.571	0.563	0.568	0.572	0.575	0.480
Iterations ($\cdot 10^3$)	972	602	632	518	472	440	530	918	1342

(3) and (ii) implicit smoothing via Gumbel noise. By disabling the ST path, we aim to identify whether gap reduction primarily stems from the discrete gradient approximation or the added stochasticity. The setup without ST corresponds to Differentiable LGNs with noisy logits and is denoted as Soft Gumbel.

In Figure 4, we see that imputing Gumbel noise alone impacts both convergence and discretization compared to Differentiable LGNs. However, we observe that including the ST estimator delays convergence for a fixed τ, but further reduces the discretization gap.

E.2. τ-parameter

We evaluate how varying the temperature τ affects optimization dynamics. Recall that higher τ reduces the degree of implicit smoothing, potentially leading to sharper minima and slower convergence. In our experiment, we test $\tau \in [0.01, 2.0]$. We show the results in Table 1 and Figure 5 where we observe a goldilocks zone for the temperature; if τ is large (> 1) or small (< 0.1), then the network converges much slower. However, higher temperatures such as 1 seem to converge to slightly better solutions. Although the difference is minor, $< 0.5\%$ when $\tau = 0.25$ goes to $\tau = 1$.

F. Curvature and entropy results

F.1. Hessian Trace Approximations

The Hessian scales quadratically with model size, so direct computations are infeasible for our networks with millions of parameters. Still, we can use iterative methods to approximate the trace, etc. (Mises & Pollaczek-Geiringer, 1929; Ipsen, 1997; Hutchinson, 1989; Trefethen & Bau, 2022). We approximate the trace using Hutchinson's method with 200 Rademacher random vectors, where each coordinate is independently sampled from $\{-1, +1\}$ with equal probability (Hutchinson, 1989). Full experimental details are provided in Appendix L.

As shown in Figure 5, decreasing the τ-parameter reduces the estimated Hessian trace. This aligns with the theoretical insights from Appendix C; lower τ values place greater weight on trace reduction. The trace is negative, this is expected:

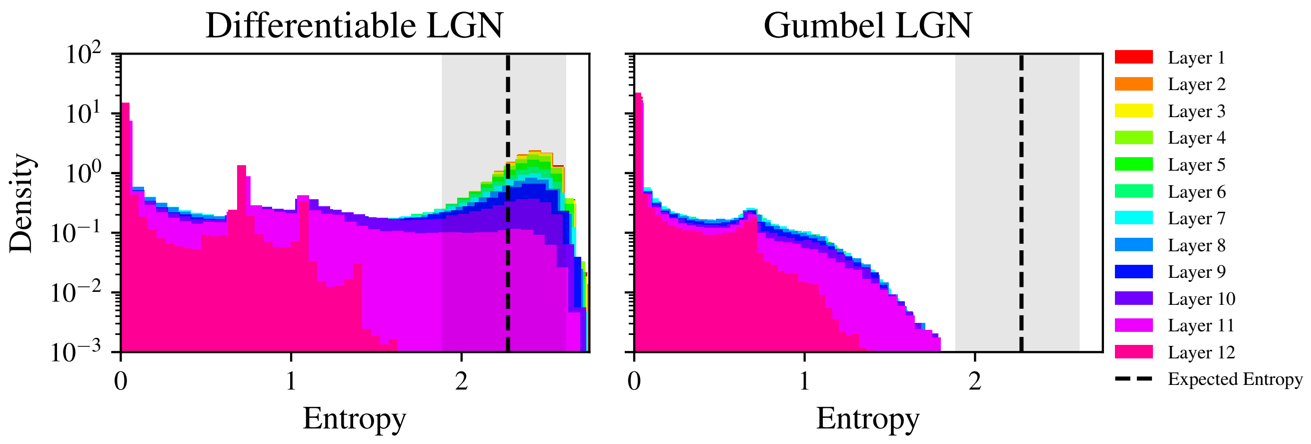

Figure 6. Entropy distribution for neurons in each layer for trained Differentiable LGNs and Gumbel LGNs models. The dashed black line indicates the expected entropy (cf. Appendix P) of neurons before training, and the shaded region is the 95% interval computed by sampling. We see that almost all the neurons in Gumbel LGNs have converged, while neurons in early layers of Differentiable LGNs still have high entropy.

stochastic gradient noise in overparameterized networks tends to systematically lower the expected Hessian trace, biasing solutions toward flatter regions of the loss landscape that may have negative trace values (Sagun et al., 2016; 2017; Wei & Schwab, 2019). Large negative eigenvalues often vanish, the trace remains influenced by many small eigenvalues, resulting in a negative overall trace.

F.2. Entropy over Logic Gates

Petersen et al. (2022) noted that neurons collapse to single gates, but they did not investigate the extent of the collapse. We examine this through neuron entropy by sampling 100k newly initialized neurons and computing the 95% interval. We also estimate expected entropy theoretically (see Appendix P), giving us a baseline distribution for neurons that have not learned. As neurons collapse, their entropy converges to 0. Figure 6 shows that many early Differentiable LGN layers do not collapse, while Gumbel LGN neurons converge with entropies near 0. Defining *unused gates* as those with entropy above the 2.5%-percentile threshold, Gumbel LGNs have 0.00% and Differentiable LGNs have 49.81% unused gates, representing a 100.00% reduction by Gumbel.

G. Limitations

While Gumbel LGNs demonstrate significant improvements for deeper networks, limitations do remain. Our evaluation focuses primarily on CIFAR-10, with limited exploration of more complex datasets or problems with many output classes. We plan to thoroughly evaluate the methods on datasets like CIFAR-100 and ImageNet32 (Krizhevsky & Hinton, 2009; Chrabaszcz et al., 2017).

The temperature parameter τ requires tuning to balance convergence speed and accuracy. Finally, our theoretical analysis connects Gumbel noise to Hessian trace minimization under simplifications, but a comprehensive theoretical treatment of the discretization gap remains an open challenge.

H. Logic Gate Network Visualizations

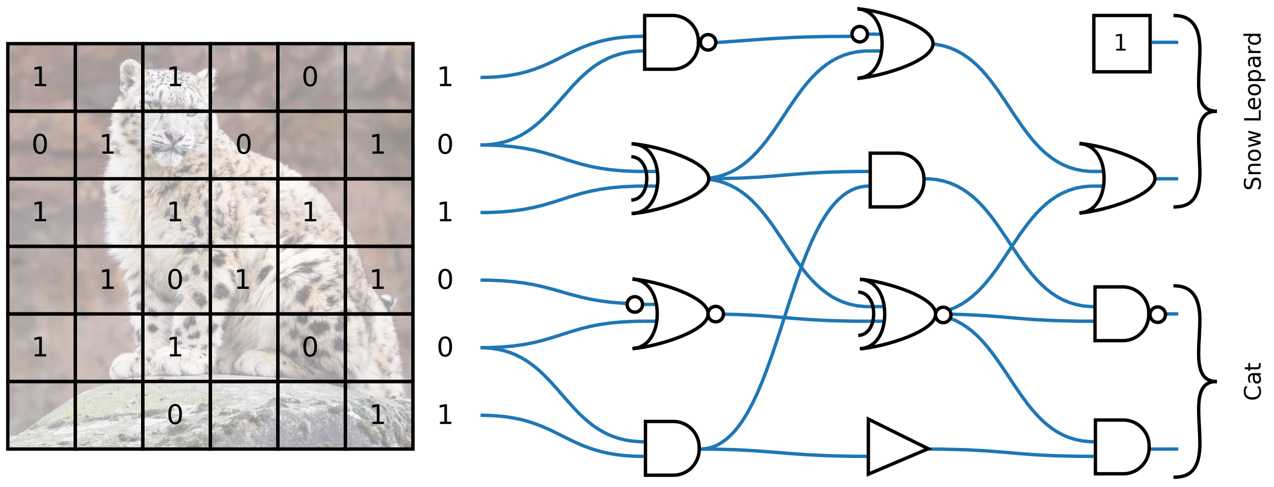

Figure 7. A diagram showing a standard logic gate network (LGN). Each logic gate receives two inputs from the previous layer. The image is first binarized before being passed into the network, and the output neurons are grouped, and each neuron in a group votes whether the image belongs to the group/class.

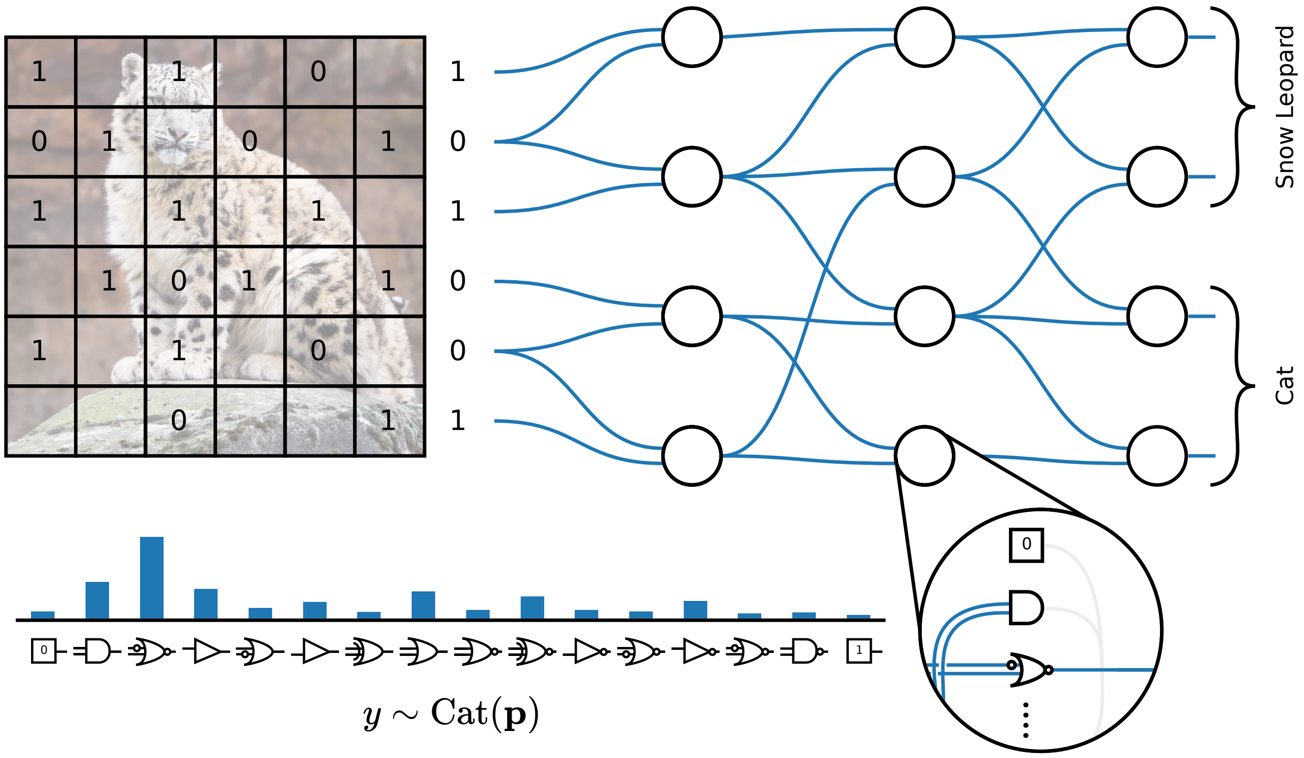

Figure 8. Forward pass through a Gumbel LGN. The top panel shows neurons producing class scores. **Bottom-left:** categorical distribution $\mathrm{Cat}(\mathbf{p})$ over relaxed logic gates, parameterized by learnable weights $\mathbf{z} \in \mathbb{R}^{16}$. **Bottom-right (zoom-in):** internal view of one neuron. The signal passes through a single selected relaxed logic gate (colors indicate which gate is chosen).

I. Distribution over Logic Gates

Gate Distribution by Layer We look in Figure 9 at the distribution of the logic gates in the final network. Our first observation is that the distributions are far more uniform for Gumbel LGNs in layers 1 to 11 than for Differentiable LGNs. At the same time, we see a sharp transition for Differentiable LGNs after layer 8. This could match the results in Figure 6, indicating that neurons in Differentiable LGNs struggle to converge in all but the final layers.

The "1" gate can be seen as a bias towards specific classes. Notably, Gumbel LGN primarily uses this gate type in the last layer, so we analyze this further.

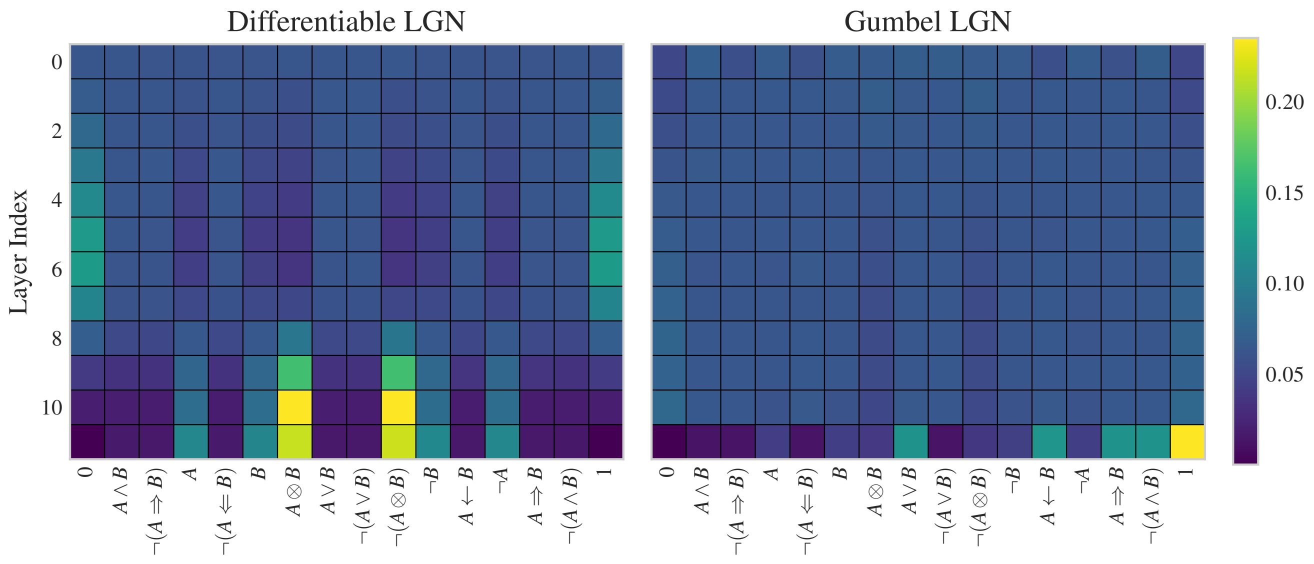

Figure 9. Gate distribution for the gates split by layer for a Differentiable LGN and a Gumbel LGN. Interestingly, the distribution is far more uniform for the Gumbel LGN in layers 1 to 11 than for the Differentiable LGN. In addition, the Gumbel LGN primarily has "1" gates in the final layer, which can be seen as a constant bias towards certain classes. We analyze this further in Appendix I.

Gate Distribution by Class We see in Figure 10 the gate distribution for each of the 10 classes in the last layer. Interestingly, the distributions between Softmax and Gumbel are quite different, but the classes are nearly identical. Since almost all classes have several "1" gates, pruning these would be possible as softmax is translation invariant.

J. Extended Sharpness-Aware Minimization

A parallel line of research focuses on improving generalization by minimizing the sharpness of the loss landscape. Motivated by prior theoretical works on generalization and flat minima (Keskar et al., 2017; Dziugaite & Roy, 2017; Jiang* et al., 2019), Foret et al. (2021) introduced Sharpness-Aware Minimization (SAM). This technique explicitly seeks flat minima by optimizing the worst-case loss within a perturbation neighborhood. Let $L_{\mathcal{S}}(\boldsymbol{w})$ be a loss function over a training set $\mathcal{S}$ of samples from a distribution $\mathscr{D}$ evaluated for model parameters $\boldsymbol{w}$.

$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{\|\boldsymbol{\epsilon}|_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}), \tag{8}$$

where $p \in [1, \infty[$ is the p-norm used (usually $p = 2$) and $\rho > 0$ is a hyperparameter (Foret et al., 2021). This optimization objective arises from the PAC-Bayesian generalization bound shown in Equation (9) (McAllester, 1999), and Foret et al. (2021) use it to show Equation (10). Below, $n = |\mathcal{S}|$, k is the number of model parameters, $\mathscr{P}$ and $\mathscr{Q}$ are the prior and posterior distributions of the model parameters, respectively. Lastly, the equations hold with probability $1-\delta$.

$$\mathbb{E}_{\boldsymbol{w}\sim\mathscr{Q}}[L_{\mathscr{D}}(\boldsymbol{w})] \leq \mathbb{E}_{\boldsymbol{w}\sim\mathscr{Q}}[L_{\mathcal{S}}(\boldsymbol{w})] + \sqrt{\frac{KL(\mathscr{Q}||\mathscr{P}) + \log\frac{n}{\delta}}{2(n-1)}}, \tag{9}$$

$$L_{\mathscr{D}}(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2\leq\rho} L_{\mathcal{S}}(\boldsymbol{w}+\boldsymbol{\epsilon}) + \sqrt{\frac{k\log\left(1+\frac{\|\boldsymbol{w}\|_2^2}{\rho^2}\left(1+\sqrt{\frac{\log(n)}{k}}\right)^2\right) + 4\log\frac{n}{\delta} + \tilde{O}(1)}{n-1}}. \tag{10}$$

Since its introduction, SAM has inspired numerous follow-up studies focused on improving computational efficiency (Liu et al., 2022a; Du et al., 2022a;b; Mi et al., 2022; Liu et al., 2022b; Mueller et al., 2023) as well as providing theoretical insights into its efficacy (Andriushchenko & Flammarion, 2022; Wang et al., 2023; Wen et al., 2023; Li et al., 2024).

K. Training GLGNs

Algorithm 1 Training STE-GLGNs

```
while not converged do
    Sample Gumbel noise $g \sim \mathrm{Gumbel}(0,1)$
    Compute soft sample $a' = \mathrm{softmax}((\log\alpha + g)/\tau)$
    Compute hard sample $\hat{a} = \mathrm{one\_hot}(\arg\max a')$
    Forward pass through logic gate network using $\hat{a}$ (with $\mathrm{stop\_grad}(\hat{a} - a') + a'$)
    Compute loss $\mathcal{L}$
    Backpropagate and update $\alpha$ and other parameters
end while
```

L. Estimating Hessian of Loss

Computing the full Hessian of the loss function is not computationally feasible due to the quadratic scaling with the number of parameters. Instead, we use scalable stochastic methods to estimate the trace of the Hessian, which provides useful curvature information. Specifically, we focus on the trace as our Lemma C.1 directly minimizes the trace of the Hessian.

Hessian-Vector Products In order to estimate the trace, we will be using Hessian-vector products on the form Hv, where $H = \nabla^2\mathcal{L}(\theta)$ is the Hessian of the loss $\mathcal{L}$ with respect to model parameters θ, and $v \in \mathbb{R}^d$ is an arbitrary vector. While explicitly forming H would require $O(d^2)$ memory and computation, such products can be computed efficiently using reverse-mode automatic differentiation (also known as Pearlmutter's trick) in $O(d)$ time and memory (Pearlmutter, 1994).

Given a scalar-valued function $\mathcal{L}(\theta)$, the Hessian-vector product Hv is defined as:

$$Hv = \nabla^2\mathcal{L}(\theta)\, v = \frac{d}{d\epsilon}\nabla\mathcal{L}(\theta+\epsilon v)\Big|_{\epsilon=0}.$$

This formulation allows for efficient computation using automatic differentiation frameworks, without explicitly constructing the Hessian matrix.

Trace Estimation via Hutchinson's Method To estimate the trace of the Hessian, we employ Hutchinson's stochastic trace estimator (Hutchinson, 1989), which approximates the trace of a matrix H as

$$\mathrm{tr}(H) \approx \frac{1}{m}\sum_{i=1}^{m} z_i^\top H z_i,$$

where each $z_i \in \mathbb{R}^d$ is a random vector with zero-mean, unit-variance, i.i.d. entries. The choice of distribution for z_i's affects the variance of our estimator. While both Gaussian and Rademacher distributions satisfy this, Rademacher vectors (each entry sampled from $\{-1,+1\}$ with equal probability) lead to a lower-variance estimator. This is formally shown in (Avron & Toledo, 2011). This makes it especially well-suited for estimating curvature efficiently in high-dimensional models.

We use $m = 200$ Rademacher vectors to produce a stable estimate. Each Hessian-vector product Hz_i is computed efficiently using reverse-mode automatic differentiation without explicitly forming the Hessian matrix.

Choice of Evaluation Points Since both trace and eigenvalue estimators are noisy and computationally expensive, we evaluate them only at selected points during training. Specifically, we choose points that correspond to monotonically increasing accuracy on the test set.

M. Loss Surface Visualization

Visualizing the geometry of the loss landscape provides insights into the optimization dynamics during training and the discretization gap. We follow the same procedure as in Li. et al 2018 (Li et al., 2018), which constructs a two-dimensional slice of the high-dimensional loss surface by perturbing model weights in random directions.

Methodology Let $\theta \in \mathbb{R}^d$ be a parameter vector of a trained model. The goal is to evaluate the loss $\mathcal{L}(\theta')$ over a grid of points

$$\theta'(\alpha, \beta) = \theta + \alpha d_1 + \beta d_2$$

where d_1 and d_1 are orthogonal directions with $\alpha, \beta \in \mathbb{R}$. We choose α, β such that we probe the model in a unit circle, i.e. $(-1, 1)$.

Direction Sampling The direction vectors d_1 and d_2 are generated as follows: Each direction is drawn using a Gaussian distribution $d_i \sim \mathcal{N}(0, 1)^d$. We then normalize $d_i = d_i / \|d_i\|$, such that each perturbation has the same overall scale.

Orthogonalization To ensure that d_1, d_2 span a meaningful plane, we apply Gram-Schmidt orthogonalization:

$$d_1 \leftarrow d_2 - \frac{\langle d_1, d_2 \rangle}{\langle d_1, d_1 \rangle} d_1$$

This ensures that we span independent, meaningful axes for visualization. This allwos us to visualize the curvature in three-dimensions.

N. Experimental Configuration

N.1. Hyperparameters

Gap Scaling with Depth we use the same hyperparameters for this experiment for Differentiable LGNs and Gumbel LGNs. Specifically, we fix the width of the network to $256K$ neurons and train the network over the depths $\{6, 8, 10, 12\}$. We optimize the network using Adam with a learning rate of 0.01. Furthermore, the batch size is set to 128 and the final parameter in the GroupSum is set to $1/0.01$. This mirrors the original experimental setup of Petersen et al. (2022) for CIFAR-10. Finally, we fix the $\tau = 1$ parameter for the Gumbel noise in the Gumbel LGNs.

A. Default CIFAR-10 Training	
Optimizer	Adam
Learning rate	0.01
Batch size	128
Depth	12 (unless varied)
Width	256 k neurons
GroupSum scale	1/0.01
B. Gap-Scaling (Depth Ablation)	
Depths tested	{6, 8, 10, 12}
Width	256k
Other settings	same as (A)
C. Ablation Studies	
Straight-Through vs Soft	depth=12, width=256k, tau=1.0
Temperature sweep	tau in {0.01,0.05,0.1,0.15,0.2,0.25,0.5,1,2}
D. Hessian Estimation	
Trace estimator	Hutchinson, m=200 Rademacher vectors
Top-eigenvalue estimator	Power iteration, 200 iterations
Evaluation points	checkpoints at monotonic test accuracy

Table 2. All hyperparameter settings, grouped by experiment.

Ablation Studies For both the straight-through and τ-parameter ablations, we use the deepest model from the gap scaling experiment (depth 12, width $256K$) to evaluate each effect in a stress-tested regime. This allows us to isolate the effect of either variant.

N.2. Implementation

Our code extends the official PyTorch `Difflogic` library by Felix Petersen, i.e., the reference implementation provided alongside the Differentiable LGN paper (Petersen et al., 2022). During the forward pass we replace the standard `torch.nn.functional.softmax` with a hard Gumbel-Softmax, `torch.nn.functional.gumbel_softmax` , thereby enabling discrete sampling while maintaining end-to-end differentiability.

O. Softmax DiffLogic Performance on MNIST-like Baselines

For all datasets, we use the same model and experiment configs.[4] The models have six layers and a width of 64k. See Table 3 for the results.

Besides the classic MNIST (Bottou et al., 1994; LeCun et al., 1998), CIFAR-10, and CIFAR-100 datasets (Krizhevsky & Hinton, 2009), we also evaluate EMNIST (balanced and letters) (Cohen et al., 2017), FashionMNIST (Xiao et al., 2017), KMNIST (Clanuwat et al., 2018), and QMNIST (Yadav & Bottou, 2019). These are black and white images, as in MNIST, but with other or more classes. We refer the reader to the original papers for details and examples.

[4]Thus, the CIFAR numbers are not representative of the optimal performance.

Table 3. The table shows the performance of Differentiable LGNs on many datasets. The key takeaway is that the discretization gap for MNIST-like datasets is minimal. The discretization gap is the difference between the discrete and soft performance. The numbers are averaged over five runs.

	Train			Test		
	Accuracy			Accuracy		
Dataset	Soft	Discrete	Disc. gap	Soft	Discrete	Disc. gap
CIFAR-10 (Krizhevsky & Hinton, 2009)	100.0 %	100.0 %	0.0 %	52.04 %	50.72 %	1.31 %
CIFAR-100 (Krizhevsky & Hinton, 2009)	83.44 %	80.39 %	3.05 %	23.86 %	23.1 %	0.76 %
EMNIST balanced (Cohen et al., 2017)	95.52 %	94.87 %	0.65 %	84.57 %	84.28 %	0.29 %
EMNIST letters (Cohen et al., 2017)	98.69 %	98.3 %	0.38 %	91.43 %	91.04 %	0.38 %
FashionMNIST (Xiao et al., 2017)	99.02 %	98.17 %	0.85 %	90.37 %	90.0 %	0.36 %
KMNIST (Clanuwat et al., 2018)	100.0 %	100.0 %	0.0 %	97.14 %	97.0 %	0.14 %
MNIST (Bottou et al., 1994)	100.0 %	100.0 %	0.0 %	98.33 %	98.16 %	0.17 %
QMNIST (Yadav & Bottou, 2019)	100.0 %	100.0 %	0.0 %	98.33 %	98.17 %	0.16 %

P. Expected Entropy

Lemma P.1. *The expected entropy of a newly initialized neuron in a Differentiable LGN is* $\approx \log 16 - \frac{1}{2} \approx 2.27$.

Proof. A neuron has $n = 16$ gates that it makes a choice over and gate i has (i.i.d.) logit $z_i \sim N(0,1)$ and probability $p_i = \frac{\exp z_i}{C}$ where $C = \sum_{j=1}^{n} \exp z_j$. For the rest of the proof, we assume the number of gates n is not fixed, and show that the expected entropy converges to $\log n - \frac{1}{2}$.

The expected entropy of $p = (p_1, p_2, \ldots)$ is

$$\begin{aligned}
\mathbb{E}[H(p)] &= -\mathbb{E}\left[\sum_{i=1}^{n} \frac{\exp z_i}{C} \log \frac{\exp z_i}{C}\right] \\
&= \mathbb{E}\left[\log C\right] - \mathbb{E}\left[\sum_{i=1}^{n} \frac{z_i \exp z_i}{C}\right] \\
&= \mathbb{E}\left[\log \sum_{j=1}^{n} \exp z_j\right] - \mathbb{E}\left[\sum_{i=1}^{n} \frac{z_i \exp z_i}{\sum_{j=1}^{n} \exp z_j}\right].
\end{aligned}$$

Here, we have as $n \to \infty$:

$$\mathbb{E}\left[\sum_{i=1}^{n} \frac{z_i \exp z_i}{\sum_{j=1}^{n} \exp z_j}\right] = \mathbb{E}\left[\frac{\frac{1}{n}\sum_{i=1}^{n} z_i \exp z_i}{\frac{1}{n}\sum_{j=1}^{n} \exp z_j}\right] = \frac{\mathbb{E}[z_i \exp z_i]}{\mathbb{E}[\exp z_j]}.$$

Using $\mathbb{E}[\exp z_1] = \sqrt{e}$ and $\mathbb{E}[z_1 \exp z_1] = \sqrt{e}$, the above gives us that as $n \to \infty$:

$$\begin{aligned}
&\mathbb{E}\left[\log \sum_{j=1}^{n} \exp z_j\right] - \mathbb{E}\left[\sum_{i=1}^{n} \frac{z_i \exp z_i}{\sum_{j=1}^{n} \exp z_j}\right] \\
&= \log n + \mathbb{E}\left[\log\left(\frac{1}{n}\sum_{j=1}^{n} \exp z_j\right)\right] - \frac{\mathbb{E}[z_i \exp z_i]}{\mathbb{E}[\exp z_j]} \\
&= \log n + \log \mathbb{E}[\exp z_j] - \frac{\mathbb{E}[z_i \exp z_i]}{\mathbb{E}[\exp z_j]} = \log n + \frac{1}{2} - 1 = \log n - \frac{1}{2}.
\end{aligned}$$

We only need to plug in $n = 16$ to get the last part. □

Table 4. Iterations per hour and the relative change from Differentiable LGNs to Gumbel LGNs.

	Iterations per hour		
	Softmax	Gumbel	Change
Depth 6	38708	38375	-0.86%
Depth 8	36292	34000	-6.31%
Depth 10	32792	31167	-4.96%
Depth 12	29417	27583	-6.23%
Mean			-4.59%

Q. Runtime

In Table 4 the number of iterations per hour Softmax and Gumbel LGNs completed while training models for the results in Figure 2. We also calculate the relative difference between the two. Gumbel is slightly slower due to the noise sampling; however, as it converges much faster in terms of iterations, the net effect is that it converges 4.5 times faster in wall-clock time.

R. Computational Resources

The experiments were done on an internal cluster with RTX 3090s and RTX 2080 Tis. In total, we have logged 1284 GPU hours for the experiments and testing. A significant part of the compute was spent on exploration.

figures/gate_distribution_by_class.pdf

Figure 10. Gate distribution for each class in the CIFAR-10 classifiers. The gates are from the last layer before the groupsum is applied.