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

Shakir Yousefi
ETH Zürich
Switzerland
syousefi@ethz.ch

Andreas Plesner[*]
ETH Zürich
Switzerland
aplesner@ethz.ch

Till Aczel[*]
ETH Zürich
Switzerland
taczel@ethz.ch

Roger Wattenhofer
ETH Zürich
Switzerland
wattenhofer@ethz.ch

Abstract

Modern neural networks exhibit state-of-the-art performance on many existing benchmarks, but their high computational requirements and energy usage cause researchers to explore more efficient solutions for real-world deployment. Differentiable logic gate networks (DLGNs) learns a large network of logic gates for efficient image classification. However, learning a network that can solve simple problems like CIFAR-10 or CIFAR-100 can take days to weeks to train. Even then, almost half of the neurons remains unused, causing a *discretization gap*. This discretization gap hinders real-world deployment of DLGNs, 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 DLGNs. 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%. *Equal contribution.

1 Introduction

Deep neural networks have reached human-level performance across a wide array of tasks [1, 2]. However, these advances come at the cost of immense computational demands during training and inference, limiting their deployment in many real-world environments [3–6]. This has sparked growing interest in designing models that retain competitive accuracy while being more efficient [7–11].

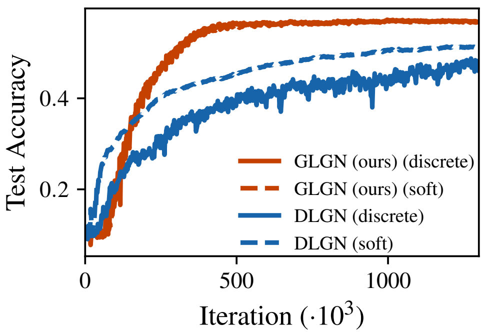

Figure 1: CIFAR-10 test accuracy comparison. Solid and dashed lines show discrete and soft performance, respectively. Gumbel LGNs (GLGN, red) demonstrate faster convergence and minimal discretization gap compared to DLGNs (DLGN, blue).

At their core, all computations on digital hardware reduce to Boolean operations such as AND, OR, and NOT. This motivates the question: *Can we express and execute machine learning models directly in the native language of hardware - namely, logic gates?*

One approach is logic gate networks (LGNs), which replaces arithmetic computations with compositions of discrete logical operations, thereby enabling efficient inference. While inference with LGNs is efficient, training them poses significant challenges. To address this, differentiable logic gate networks (DL-

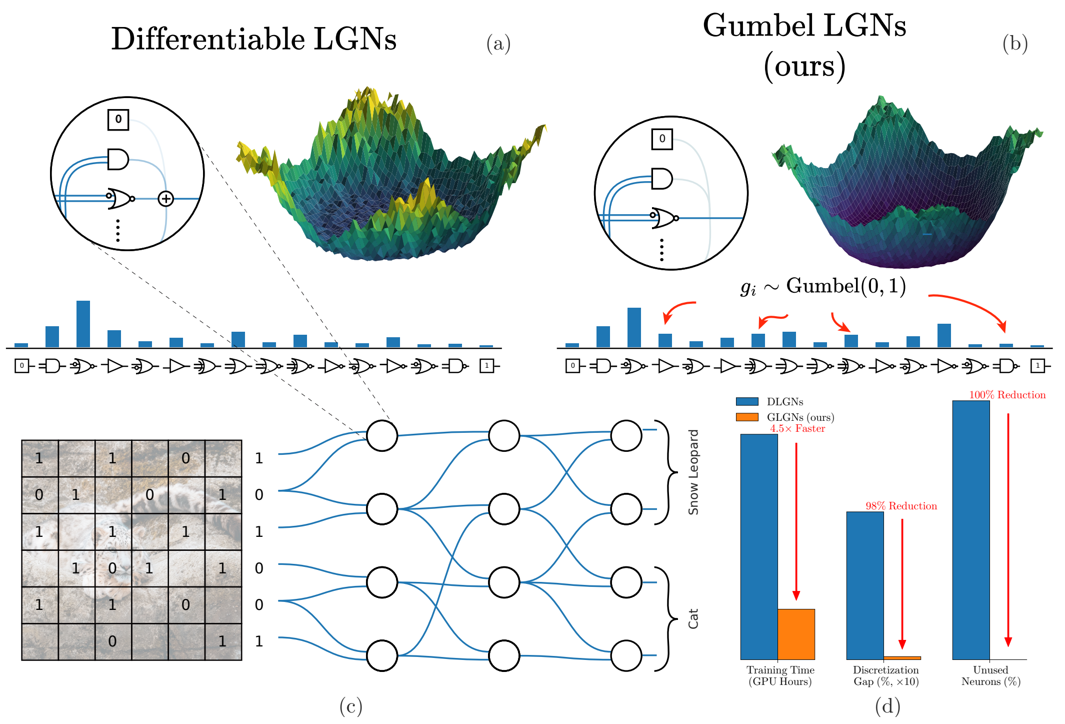

Figure 2: Overview figure. **(a)** DLGNs: Leftmost shows the internal structure of a node. We parameterize each node by weighing the 16 possible logic gates (shown below) and summing their output. This results in a hard, brittle, loss landscape, slowing convergence and increasing the discretization gap. **(b)** Gumbel LGNs: during training, we inject Gumbel noise on the 16 gate weights and select the logic gate with the highest weight. This results in a smoother loss landscape and aligns the training with the network at inference. This results in faster training and reduces the discretization gap. **(c)** Structure of DLGNs and Gumbel LGNs. Each neuron receives two inputs. The nodes in the final layer are aggregated by summation, thus producing class likelihoods. **(d)** Results of using Gumbel LGNs instead of DLGNs. We achieve up to $4.5\times$ faster convergence (in wall-clock time), a 98% reduction in discretization gap, and 100% reduction of unused neurons.

GNs) [7, 9] introduce continuous relaxations of both logical operations and gate selections, allowing the use of gradient-based optimization methods.

Despite their potential, we identify and propose solutions to two major challenges. **(1) Discretization gap**: DLGNs rely on continuous relaxations during training, but final models must be discretized for inference. This mismatch may result in notable degradation, reducing model accuracy by 3% on the same data. **(2) Slow convergence**: While LGNs are efficient at inference time, training DLGNs remains slow due to the reliance on differentiable relaxations, making DLGNs converge substantially slower than 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 [12]. 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 [13–16]. Our central hypothesis is that *smoother loss landscapes* make DLGN models more robust to discretization and facilitate faster and more stable training. Since the loss landscape is smoother, the gradient signal is better, thus making the networks converge faster. At the same time, the improved gradient signal also causes more neurons to collapse, thus reducing the impact of discretization.

We propose Gumbel Logic Gate Networks (Gumbel LGNs), which inject Gumbel noise into the gate selection process using the Gumbel-Softmax trick. By injecting Gumbel noise into the gate selection process, we introduce stochasticity that flattens the loss landscape, improving the optimization process and reducing discretization sensitivity. Empirically, we find that Gumbel LGNs exhibit faster convergence and smaller discretization gaps than baseline DLGNs.

To further reduce the discretization gap, we explore a training strategy inspired by techniques from discretization-aware training and neural architecture search (NAS): using continuous relaxations only in the backward pass, while enforcing discrete gates in the forward pass. Although ST estimator may slightly slow convergence, it substantially reduces the discretization gap. Moreover, it aligns training dynamics with inference-time behavior, and as it only influences training, there is no impact on inference.

To our knowledge, no prior work analyzes the discretization gap in DLGNs, and draw formal connections to the smoothness of the loss landscape. NAS and DARTS typically studies explored search spaces up to size 10^{18}, while our experiments operate over vastly larger parameter spaces, exceeding $10^{3,600,000}$, and show that NAS techniques scale to this setting.

Our contributions are as follows:

- **Empirical validation.** We design and execute experiments showing that Gumbel LGNs both speed up training and boost neuron utilization.
- **Theoretical analysis.** We prove that injecting Gumbel noise into DLGNs smooths their loss landscape by regularizing the Hessian's trace, thereby reducing the discretization gap and accelerating convergence.
- **Practical algorithmic insight.** We demonstrate that employing the straight-through estimator further closes the discretization gap in practice.

2 Background

Logic Gate Networks Logic Gate Networks (LGNs) represent an entire network as a composition of discrete logic operations, such as AND, OR, and XOR. LGNs consist of several layers, where each layer contains n *fixed* logic gates, referred to as neurons, that work in parallel. n is the width of the layer. Each neuron in a layer takes as input the output of two (random) neurons in the previous layer. We use the GroupSum operation to get class scores as done by Petersen et al. [7]. For classifying the input into k classes, the neurons in the final layer are grouped into disjoint groups with one group per class. The outputs of the neurons in group i are summed to give the likelihood of class i. Mathematically this is expressed as $s_i = \frac{1}{\tau^{\mathrm{GS}}} \sum_{j \in G_i} a_j$, where G_i is the i'th group, a_j is the output of neuron j, and τ^{GS} is a temperature parameter. We then predict the class with the highest score. For more details, see [17].

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 (DLGNs) [7, 9] 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, DLGNs 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 DLGNs evaluated in the differentiable setting (using Equation (1)) 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 $\pi \in \Delta^k$ [18–21]. Let $g \sim \mathrm{Gumbel}(0, 1)$ 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 (2).

$$i = \arg\max_j (g_j + \log \pi_j), \quad g_j \sim \text{Gumbel}(0, 1). \tag{2}$$

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 the Softmax with temperature τ as below:

$$\pi_i^{\text{Gumbel}} = \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_j \exp((\log \pi_j + g_j)/\tau)}, \quad \pi_i = \sum_{i=1}^{k} \frac{\exp z_i}{\sum_j \exp z_j}, \quad z_i \in \mathbb{R}. \tag{3}$$

3 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 [22–27]. These light models use various methods such as lookup tables [28], binary and quantized neural networks [29–32], and sparse neural networks [33–37].

Of particular interest in this context are differentiable logic gate networks (DLGNs), which have demonstrated state-of-the-art performance in image classification tasks [7, 9], as well as in rule extraction from observed cellular automata dynamics [17]. Our proposed improvements target convergence behavior and are orthogonal to the architectural innovations of the convolutional DLGN variant [9]; 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. [7, 9].

Kim [38] extend DLGNs by modifying the neuron parameterization introduced in Petersen et al. [7]. Specifically, they normalize the logits and remove the temperature parameter τ. In addition, they employ a straight-through (ST) estimator [39] and injectGumbel noise with a learnable scale parameter. However, Kim [38] do not analyze the discretization gap or study the convergence behavior of DLGNs. Moreover, they draw no formal connection between Gumbel noise and its role as an implicit Hessian regularizer or loss smoothener.

Differentiable Neural Architecture Search Neural Architecture Search (NAS) aims to automate the selection of high-performing model architectures from a large design space [40–42]. While early approaches were computationally expensive, subsequent efforts have focused on improving efficiency [43, 44]. Several works have addressed the issue of train–test performance discrepancies by proposing sampling-based training [45] or regularization techniques that bias architecture selection toward configurations with better generalization [46].

A seminal contribution in this domain is Differentiable Architecture Search (DARTS) by Liu et al. [47], 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 DLGNs.

More recently, Chen and Hsieh [16] 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 DLGNs by Petersen et al. [7, 9] 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} [48–50],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, DLGNs demonstrate the viability of DARTS 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 [51]. Typically, we aim to avoid these residual connections, as they do not increase the models' expressive power [46]. In relation, Petersen et al. [9] 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 [14, 52, 53], Foret et al. [15] introduced Sharpness-Aware Minimization (SAM) in Equation (4). This technique explicitly seeks flat minima by optimizing the worst-case loss within a perturbation neighborhood.

$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} L_S(\boldsymbol{w} + \boldsymbol{\epsilon}), \tag{4}$$

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 [15]. Since its introduction, SAM has inspired numerous follow-up studies focused on improving computational efficiency [54–59] as well as providing theoretical insights into its efficacy [60–63]. We refer to Appendix D for a detailed description of SAM.

4 Gumbel Logic Gate Networks

We introduce *Gumbel Logic Gate Networks* (Gumbel LGNs), which employ discrete sampling of logic gates via the Gumbel-Softmax trick [20, 21] with a straight-through (ST) estimator [39]. While conventional DLGNs 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 (2)). During backpropagation, the non-differentiable argmax is approximated using the Gumbel-Softmax (Equation (3)), enabling end-to-end training.

This approach is motivated by two 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 [15, 16]. (2) Inference-time alignment: In DLGNs, 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 DLGNs, we model each neuron as a distribution over binary, relaxed logic gates $\mathcal{S} = \{h_1, h_2, \ldots, h_{16}\}$, 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 (3). The output of the neuron with inputs (a, b) is then given below.

$$f_{\mathbf{z}}^{\text{soft}}(a,b) = \sum_{i=1}^{16} \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_j \exp((\log \pi_j + g_j)/\tau)} \cdot h_i(a,b) = \sum_{i=1}^{16} \pi_i^{\text{Gumbel}} \cdot h_i(a,b), \tag{5}$$

where $\tau > 0$ is a temperature parameter controlling the sharpness of the distribution. As $\tau \to 0$, the distribution increasingly peaks around the maximum-logit index [20]. 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 E for pseudo-code implementation of the training process. Concretely, during the forward pass, we sample Gumbel noise $\mathbf{g} \sim \mathrm{Gumbel}(0,1)^{16}$ and compute:

$$f_{\mathbf{z}}^{\text{discrete}}(a,b) = h_k(a,b) \tag{6}$$

using the gate h_k with maximum perturbed logit. During the backward pass, we use the soft Gumbel-Softmax relaxation (Equation (5)) 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 2 or Figure 11 in Appendix A 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 $g_i \sim \mathrm{Gumbel}(0,1)$. Adding Gumbel noise with $\tau \in \mathbb{R}$ to the logits can be seen as Monte-Carlo sample of the objective $J(\mathbf{z})$;

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

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

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

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

and set $\mathbf{a} = \mathbf{z}/\tau$ *and* $f(\mathbf{a}) = \mathcal{L}(\mathrm{softmax}(\mathbf{a}))$. *We then get the expression below.*

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

Proof. See Appendix B. □

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 [12]. The expected loss scales with

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

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*.

5 Empirical Evaluations

Our empirical evaluations focus on CIFAR-10 and CIFAR-100. Due to constrained resources, we limit experiments by default to 48 GPU hours. To ensure a fair comparison, we use the hyperparameters from [7] whenever possible rather than tuning the parameters, such as learning rate, ourselves. Appendix H.1 contains all the default parameters.

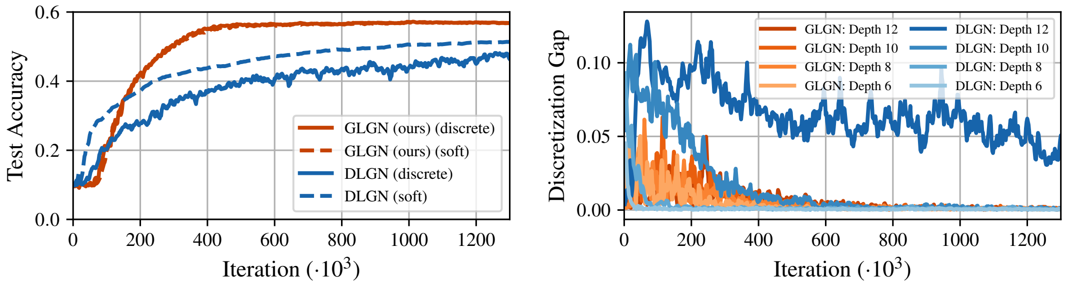

Figure 3: Performance of Gumbel LGNs and DLGNs on CIFAR-10. (**Left**) Test accuracy (width of 256k, depth of 12). (**Right**) Discretization gap for various depths. DLGNs experience larger gaps and slower reduction as the depth increases. In contrast, Gumbel LGNs have consistently low gaps and fast reduction as the network depth increases.

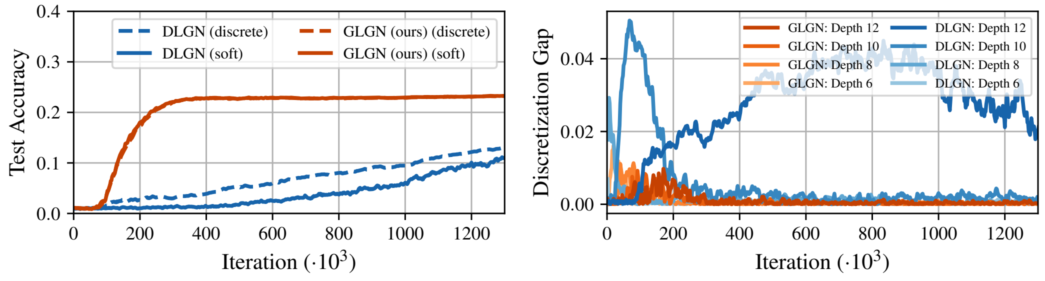

Figure 4: Performance of Gumbel LGNs and DLGNs on CIFAR-100. (**Left**) Test accuracy (width of 256k, depth of 12). (**Right**) Discretization gap for various depths. DLGNs experience larger gaps and slower reduction as the depth increases. In contrast, Gumbel LGNs have consistently low gaps and fast reduction as the network depth increases.

Discretization Gap Figure 3 shows 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 3. Gumbel LGN converges much faster than the DLGNs, with virtually no discretization gap. We also measure the runtime and see that Gumbel increases the runtime by roughly 5% per iteration (cf. Appendix K). Combined with runtime results from Table 4 in Appendix K, Gumbel LGN converges[1] $4.75\times$ 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.

We perform an additional evaluation on CIFAR-100 using the same settings. On Figure 4, we see that GLGNs already converge at around 400K iterations, whereas DLGNs do not converge. This is consistent with the results, we observe for CIFAR-10.

Gap Scaling with Depth On the right of Figure 3, we see the discretization gap for models of various depths for DLGNs and Gumbel LGNs. As the model depth increases, the expressive power of the networks theoretically increases. The DLGNs experience bigger discretization gaps as the depth increases, while our Gumbel LGNs are stable across depths. Hence, Gumbel LGNs do not struggle to converge as the networks are made deeper. Both methods experience a shift in when the maximum gap occurs. This is expected since increasing the model size usually delays when the accuracy plateaus, i.e., more iterations are needed to converge.

Shallow Network We fix the depth of the network at 6 and increase the width to 2048K. On Figure 5, we see that. Interestingly, the accuracy of the soft DLGN converges within 50K iterations. However, the discretized version does not seem to improve with continued training. We note that the

[1]For this, we match Gumbel LGNs' discrete accuracy with DLGNs' maximum discrete accuracy.

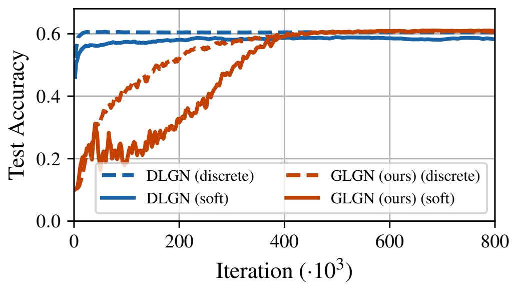

Figure 5: Test accuracy on CIFAR-10 for a shallow, wide network (width 2048k, depth 6).

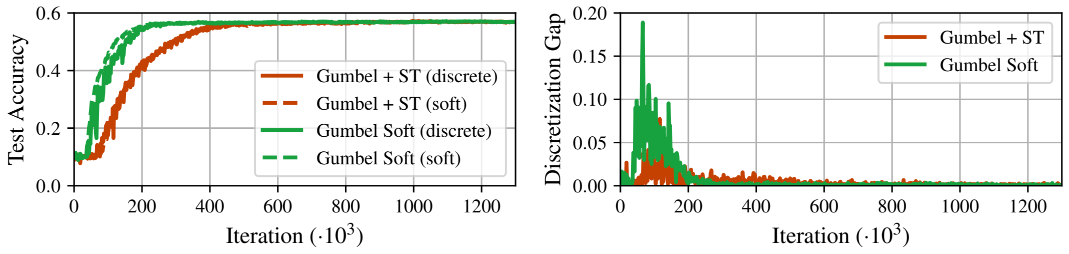

Figure 6: Straight-through (ST) estimator ablation (width of 256k, depth 12). The Gumbel LGNs uses hard gate choices in the forward pass as shown in (2) called the ST estimator. On the left, we show the test accuracy over training iterations; on the right, we show the discretization gap. Gumbel LGNs with ST estimator converge slightly slower in test accuracy, but the discretization gap is smaller.

final accuracy of GLGNs is higher than DLGNs. This suggests increasing the number of neurons through the width also contributes to the discretization gap.

Straight-Through Estimator Ablation 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 (2) 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 DLGNs with noisy logits and is denoted as Soft Gumbel.

In Figure 6, we see that imputing Gumbel noise alone impacts both convergence and discretization compared to DLGNs. However, we observe that including the ST estimator delays convergence for a fixed τ, but further reduces the discretization gap.

Curvature, Hessian, and Smoothness As we showed in Section 4, loss minimization for Gumbel LGNs implicitly reduces the curvature of minima by minimizing the trace of the Hessian. The impact of the trace compared to the loss depends on the temperature parameter τ. Thus, we perform an ablation study on the effect of τ, estimate the trace of the Hessian, and visualize the loss landscape.

Ablation over 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 fig. 7 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 $\tau = 1$.

Hessian Trace Approximation 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. [64–68]. We approximate the trace using Hutchinson's method with

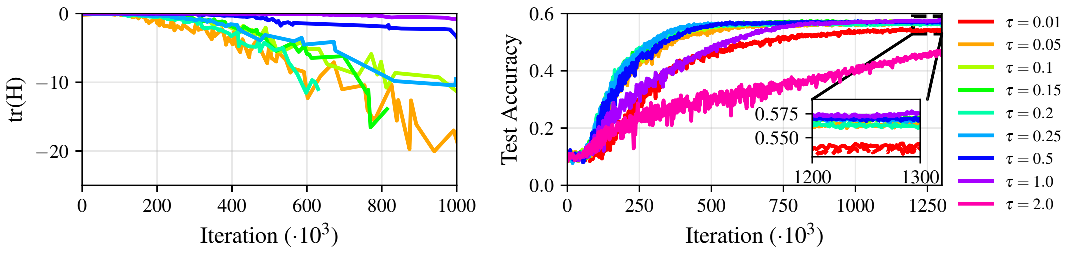

Figure 7: Ablation over the temperature τ for Gumbel LGNs. **Left)** Estimated Hessian trace using Hutchinson's method. The trace shrinks as τ decreases, indicating fewer large positive eigenvalues, thus suggesting a flatter loss surface, which may reduce the risk of loss increases when discretizing 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

200 Rademacher random vectors, where each coordinate is independently sampled from $\{-1, +1\}$ with equal probability [67]. Full experimental details are provided in Appendix F.

As shown in Figure 7, decreasing the τ-parameter reduces the estimated Hessian trace. This aligns with the theoretical insights from Section 4; lower τ values place greater weight on trace reduction. The trace is negative, this is expected: 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 [69–71]. Large negative eigenvalues often vanish, the trace remains influenced by many small eigenvalues, resulting in a negative overall trace.

Curvature Visualization To qualitatively assess the loss landscape curvature, we project the high-dimensional parameter space onto two-dimensional subspaces. Following Li et al. [72], 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 G. Figure 8 shows that Gumbel LGN has a visually smoother loss surface.

Entropy over Logic Gates Petersen et al. [7] 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 J), giving us a baseline distribution for neurons that have not learned. As neurons collapse, their entropy converges to 0. Figure 9 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 DLGNs have 49.81% unused gates, representing a 100.00% reduction by Gumbel.

6 Limitations

While Gumbel LGNs demonstrate significant improvements for deeper networks, limitations do remain. Our evaluation focuses primarily on CIFAR-10 and CIFAR-100, with limited exploration of more complex datasets. The temperature parameter τ requires tuning to balance convergence speed

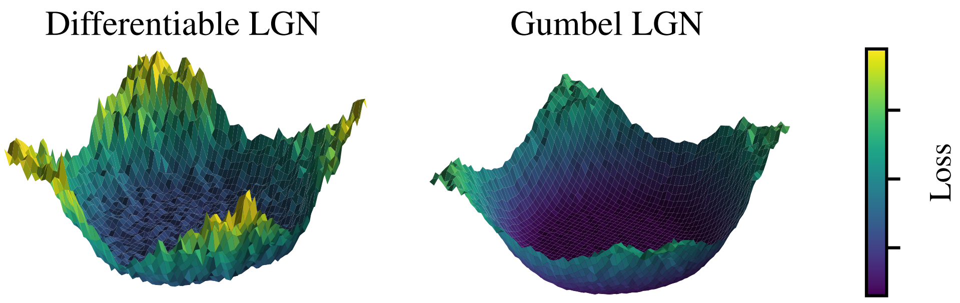

Figure 8: Visualization of loss landscapes. **Left:** Loss landscape of a Differentiable LGN. We see that the landscape is overall noisy. **Right:** Loss landscape of a Gumbel LGN with $\tau = 1.0$. We observe a much smoother loss landscape compared to the DLGNs.

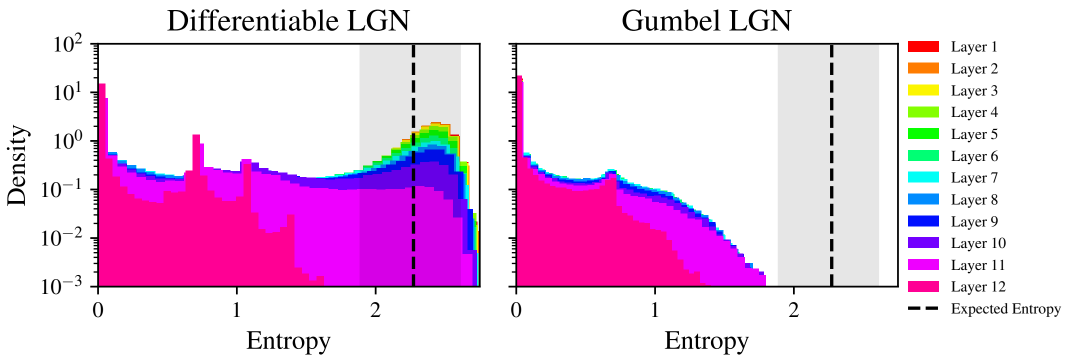

Figure 9: Entropy distribution for neurons in each layer for trained DLGNs and Gumbel LGNs models. The dashed black line indicates the expected entropy (cf. Appendix J) of neurons before training, and the shaded region is the 95% interval computed by sampling. We see that almost all the neurons in Gumbel LGNs have converged, while neurons in early layers of DLGNs still have high entropy.

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. Moreover, a comprehensive analysis of width scaling and the interplay between width and depth is left for future work.

7 Conclusion

We introduced Gumbel Logic Gate Networks (Gumbel LGNs), addressing two critical limitations of DLGNs: slow convergence during training and a large discretization gap. 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-100 and CIFAR-10 demonstrate that Gumbel LGNs converge up to $4.5\times$ faster in wall-clock time than DLGNs 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, e.g. adaptive temperature scheduling.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, December 2015. doi: 10.1109/ICCV.2015.123.

[2] Andreas Plesner, Tobias Vontobel, and Roger Wattenhofer. Breaking recaptchav2. In *2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1047–1056. IEEE, 2024.

[3] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, February 2016.

[4] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and Policy Considerations for Modern Deep Learning Research. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(09):13693–13696, April 2020. ISSN 2374-3468. doi: 10.1609/aaai.v34i09.7123.

[5] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The Computational Limits of Deep Learning, July 2022.

[6] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing Frontiers in Open Language Model Post-Training, April 2025. URL `http://arxiv.org/abs/2411.15124`. arXiv:2411.15124 [cs].

[7] Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep Differentiable Logic Gate Networks. In *Advances in Neural Information Processing Systems*, October 2022.

[8] Zachary Susskind, Aman Arora, Igor D. S. Miranda, Luis A. Q. Villon, Rafael F. Katopodis, Leandro S. de Araújo, Diego L. C. Dutra, Priscila M. V. Lima, Felipe M. G. França, Mauricio Breternitz, and Lizy K. John. Weightless Neural Networks for Efficient Edge Inference. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, PACT '22, pages 279–290, New York, NY, USA, January 2023. Association for Computing Machinery. ISBN 978-1-4503-9868-8. doi: 10.1145/3559009.3569680.

[9] Felix Petersen, Hilde Kuehne, Christian Borgelt, Julian Welzel, and Stefano Ermon. Convolutional differentiable logic gate networks. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 121185–121203. Curran Associates, Inc., 2024.

[10] Marta Andronic and George A. Constantinides. NeuraLUT-Assemble: Hardware-Aware Assembling of Sub-Neural Networks for Efficient LUT Inference. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 208–216, May 2025. doi: 10.1109/FCCM62733.2025.00077.

[11] Alan T. L. Bacellar, Zachary Susskind, Mauricio Breternitz Jr, Eugene John, Lizy K. John, Priscila M. V. Lima, and Felipe M. G. França. Differentiable Weightless Neural Networks, March 2025.

[12] Jacob Mitchell Springer, Sachin Goyal, Kaiyue Wen, Tanishq Kumar, Xiang Yue, Sadhika Malladi, Graham Neubig, and Aditi Raghunathan. Overtrained Language Models Are Harder to Fine-Tune, March 2025.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Flat Minima. *Neural Computation*, 9(1):1–42, January 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.1.1.

[14] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. In *International Conference on Learning Representations*, February 2017.

[15] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-Aware Minimization for Efficiently Improving Generalization, April 2021.

[16] Xiangning Chen and Cho-Jui Hsieh. Stabilizing Differentiable Architecture Search via Perturbation-based Regularization, January 2021.

[17] Pietro Miotti, Eyvind Niklasson, Ettore Randazzo, and Alexander Mordvintsev. Differentiable Logic CA: From Game of Life to Pattern Generation. https://google-research.github.io/self-organising-systems/difflogic-ca/, March 2025.

[18] E. J. Gumbel. 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.

[19] Chris J. Maddison, Daniel Tarlow, and Tom Minka. A* Sampling. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[20] Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, February 2017.

[21] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations (ICLR)*, 2017.

[22] Donghyun Park, Seulgi Kim, Yelin An, and Jae-Yoon Jung. 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.

[23] Shiya Liu, Dong Sam Ha, Fangyang Shen, and Yang Yi. Efficient neural networks for edge devices. *Computers & Electrical Engineering*, 92:107121, June 2021. ISSN 0045-7906. doi: 10.1016/j.compeleceng.2021.107121.

[24] Mohammad Al-Quraan, Lina Mohjazi, Lina Bariah, Anthony Centeno, Ahmed Zoha, Kamran Arshad, Khaled Assaleh, Sami Muhaidat, Mérouane Debbah, and Muhammad Ali Imran. 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.

[25] Rahul Mishra and Hari Prabhat Gupta. 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.

[26] Yusen Zhang, Yunchuan Qin, Yufeng Zhang, Xu Zhou, Songlei Jian, Yusong Tan, and Kenli Li. 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.

[27] Shahzaib Iqbal, Tariq M. Khan, Syed S. Naqvi, Asim Naveed, Muhammad Usman, Haroon Ahmed Khan, and Imran Razzak. 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.

[28] Satrajit Chatterjee. Learning and Memorization. In *Proceedings of the 35th International Conference on Machine Learning*, pages 755–763. PMLR, July 2018.

[29] Yichi Zhang, Junhao Pan, Xinheng Liu, Hongzheng Chen, Deming Chen, and Zhiru Zhang. 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, pages 171–182, New York, NY, USA, February 2021. Association for Computing Machinery. ISBN 978-1-4503-8218-2. doi: 10.1145/3431920.3439296.

[30] Yunqiang Li, Silvia-Laura Pintea, and Jan C Van Gemert. 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.

[31] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

[32] Chunyu Yuan and Sos S. Agaian. 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.

[33] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. 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.

[34] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

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

[36] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pages 10323–10337. PMLR, 2023.

[37] Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[38] Youngsung Kim. Deep Stochastic Logic Gate Networks. *IEEE Access*, 11:122488–122501, 2023. ISSN 2169-3536. doi: 10.1109/ACCESS.2023.3328622.

[39] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation, August 2013.

[40] Barret Zoph and Quoc Le. Neural Architecture Search with Reinforcement Learning. In *International Conference on Learning Representations*, February 2017.

[41] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[42] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. 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.

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

[44] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic Neural Architecture Search, April 2020.

[45] Jianlong Chang, xinbang zhang, Yiwen Guo, GAOFENG MENG, SHIMING XIANG, and Chunhong Pan. DATA: Differentiable ArchiTecture approximation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[46] Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair DARTS: Eliminating Unfair Advantages in Differentiable Architecture Search, July 2020.

[47] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search, April 2019.

[48] Arber Zela, Julien Siems, and Frank Hutter. NAS-Bench-1Shot1: Benchmarking and Dissecting One-shot Neural Architecture Search. In *International Conference on Learning Representations*, September 2019.

[49] Renbo Tu, Nicholas Roberts, Misha Khodak, Junhong Shen, Frederic Sala, and Ameet Talwalkar. NAS-Bench-360: Benchmarking Neural Architecture Search on Diverse Tasks. *Advances in Neural Information Processing Systems*, 35:12380–12394, December 2022.

[50] Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. Neural Architecture Search Benchmarks: Insights and Survey. *IEEE Access*, 11:25217–25236, 2023. ISSN 2169-3536. doi: 10.1109/ACCESS.2023.3253818.

[51] Yunjie Tian, Chang Liu, Lingxi Xie, Jianbin jiao, and Qixiang Ye. Discretization-aware architecture search. *Pattern Recognition*, 120:108186, December 2021. ISSN 0031-3203. doi: 10.1016/j.patcog.2021.108186.

[52] Gintare Karolina Dziugaite and Daniel M. Roy. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data, October 2017.

[53] Yiding Jiang*, Behnam Neyshabur*, Hossein Mobahi, Dilip Krishnan, and Samy Bengio. Fantastic Generalization Measures and Where to Find Them. In *International Conference on Learning Representations*, September 2019.

[54] Yong Liu, Siqi Mai, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Towards Efficient and Scalable Sharpness-Aware Minimization. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12350–12360, New Orleans, LA, USA, June 2022. IEEE. ISBN 978-1-6654-6946-3. doi: 10.1109/CVPR52688.2022.01204.

[55] Jiawei Du, Hanshu Yan, Jiashi Feng, Joey Tianyi Zhou, Liangli Zhen, Rick Siow Mong Goh, and Vincent Y. F. Tan. Efficient Sharpness-aware Minimization for Improved Training of Neural Networks, May 2022.

[56] Jiawei Du, Daquan Zhou, Jiashi Feng, Vincent Tan, and Joey Tianyi Zhou. Sharpness-aware training for free. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 23439–23451. Curran Associates, Inc., 2022.

[57] Peng Mi, Li Shen, Tianhe Ren, Yiyi Zhou, Xiaoshuai Sun, Rongrong Ji, and Dacheng Tao. Make sharpness-aware minimization stronger: A sparsified perturbation approach. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 30950–30962. Curran Associates, Inc., 2022.

[58] Yong Liu, Siqi Mai, Minhao Cheng, Xiangning Chen, Cho-Jui Hsieh, and Yang You. Random sharpness-aware minimization. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24543–24556. Curran Associates, Inc., 2022.

[59] Maximilian Mueller, Tiffany Vlaar, David Rolnick, and Matthias Hein. Normalization Layers Are All That Sharpness-Aware Minimization Needs, November 2023.

[60] Maksym Andriushchenko and Nicolas Flammarion. Towards Understanding Sharpness-Aware Minimization, June 2022.

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

[62] Kaiyue Wen, Tengyu Ma, and Zhiyuan Li. How Does Sharpness-Aware Minimization Minimize Sharpness?, January 2023.

[63] Tao Li, Pan Zhou, Zhengbao He, Xinwen Cheng, and Xiaolin Huang. Friendly Sharpness-Aware Minimization. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5631–5640, Seattle, WA, USA, June 2024. IEEE. ISBN 979-8-3503-5300-6. doi: 10.1109/CVPR52733.2024.00538.

[64] R. V. Mises and H. Pollaczek-Geiringer. 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.

[65] R. V. Mises and H. Pollaczek-Geiringer. Praktische Verfahren der Gleichungsauflösung Section 5-6. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 9(2):152–164, 1929. ISSN 1521-4001. doi: 10.1002/zamm.19290090206.

[66] Ilse CF Ipsen. Computing an eigenvector with inverse iteration. *SIAM review*, 39(2):254–291, 1997.

[67] Michael F Hutchinson. 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.

[68] Lloyd N Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 2022.

[69] Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[70] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[71] Mingwei Wei and David J Schwab. How noise affects the hessian spectrum in overparameterized neural networks. *arXiv preprint arXiv:1910.00195*, 2019.

[72] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf`.

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

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

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

[76] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[77] L. Bottou, C. Cortes, J.S. Denker, H. Drucker, I. Guyon, L.D. Jackel, Y. LeCun, U.A. Muller, E. Sackinger, P. Simard, and V. Vapnik. 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, pages 77–82 vol.2, October 1994. doi: 10.1109/ICPR.1994.576879.

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. 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.

[79] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009.

[80] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: An extension of MNIST to handwritten letters, March 2017.

[81] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms, September 2017.

[82] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. Deep Learning for Classical Japanese Literature, November 2018.

[83] Chhavi Yadav and Léon Bottou. Cold Case: The Lost MNIST Digits, November 2019.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Proof in Appendix B.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: See, for instance, Appendices G and H.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: CIFAR-10 is an open-source dataset and we provide the codebase.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: See Appendix H.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We were not able to rerun the experiments under multiple seeds due to compute constraints.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: See Appendix L.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: We conduct foundational research.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: To the best of our knowledge, there are no risks posed by the release of our code.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We only used open-sourced models and data.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide instructions for the codebase in the zip file.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: NA

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: NA

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: NA

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Logic Gate Network Visualizations

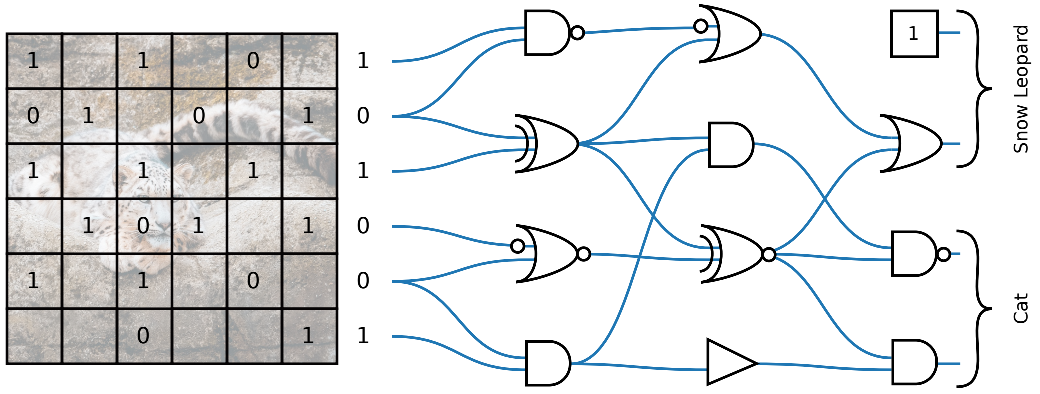

Figure 10: A diagram showing a standard logic gate network (LGN). Each logic gate receives two inputs from the previous layer. The image is first binarized before being passed into the network, and the output neurons are grouped, and each neuron in a group votes whether the image belongs to the group/class.

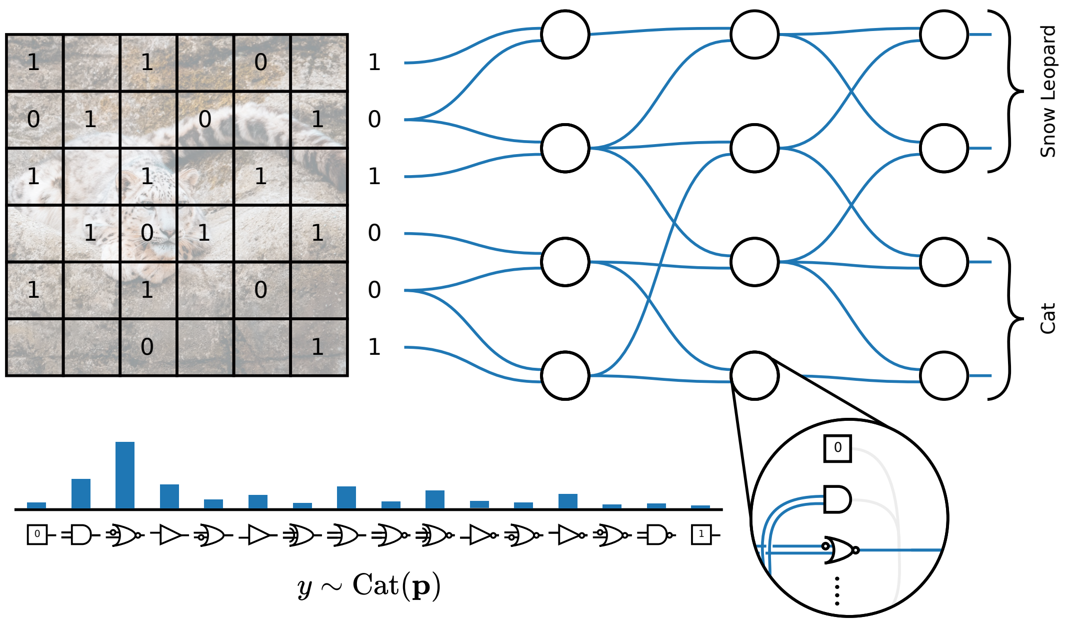

Figure 11: Forward pass through a Gumbel LGN. The top panel shows neurons producing class scores. **Bottom-left:** categorical distribution $\mathrm{Cat}(\mathbf{p})$ over relaxed logic gates, parameterized by learnable weights $\mathbf{z} \in \mathbb{R}^{16}$. **Bottom-right (zoom-in):** internal view of one neuron. The signal passes through a single selected relaxed logic gate (colors indicate which gate is chosen).

B Gumbel Smoothing

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

Lemma 2 (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*

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

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

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

□

Restating Lemma 1

Lemma 3 (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

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

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

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

which follows from $\mathbb{E}[g_i^2] = \operatorname{Var}(g_i) + \mathbb{E}[g_i]^2 = \pi^2/6 + \gamma^2$, $\mathbb{E}[g_i g_j] = \gamma^2$ 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

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

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

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

□

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

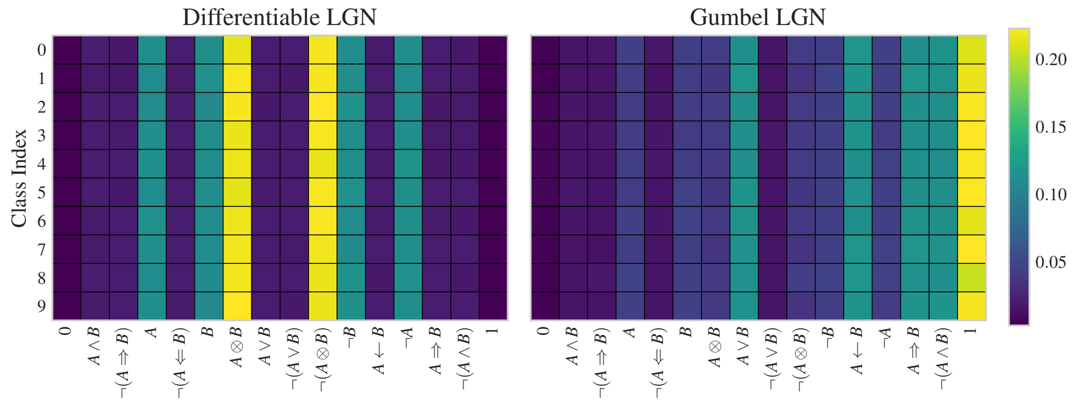

Figure 13: Gate distribution for each class in the CIFAR-10 classifiers. The gates are from the last layer before the groupsum is applied.

C Distribution over Logic Gates

Gate Distribution by Layer We look in Figure 12 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 DLGNs. At the same time, we see a sharp transition for DLGNs after layer 8. This could match the results in Figure 9, indicating that neurons in DLGNs 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.

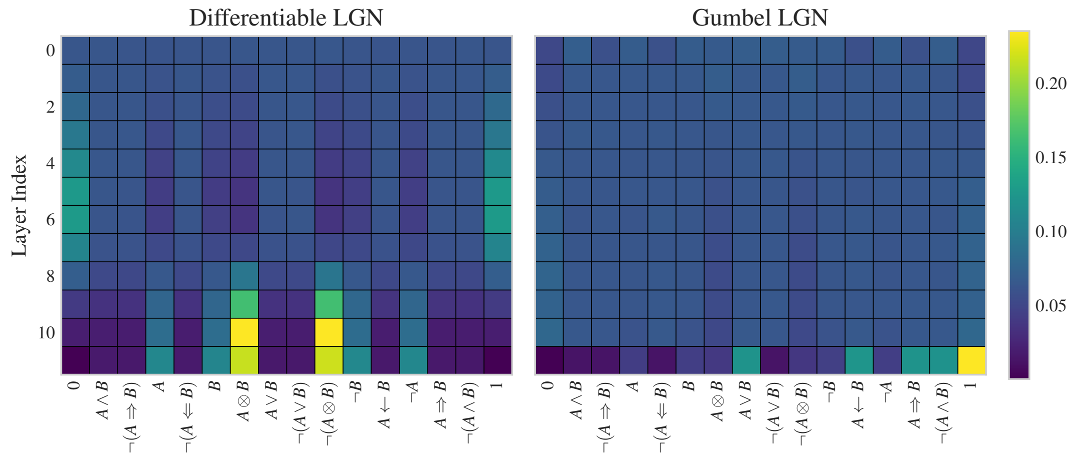

Figure 12: Gate distribution for the gates split by layer for a Differentiable LGN and a Gumbel LGN. Interestingly, the distribution is far more uniform for the Gumbel LGN in layers 1 to 11 than for the Differentiable LGN. In addition, the Gumbel LGN primarily has "1" gates in the final layer, which can be seen as a constant bias towards certain classes. We analyze this further in Appendix C.

Gate Distribution by Class We see in Figure 13 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.

D 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 [14, 52, 53], Foret et al. [15] 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 $\boldsymbol{w}$.

$$\min_{\boldsymbol{w}} L_S^{SAM}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2 \quad \text{where} \quad L_S^{SAM}(\boldsymbol{w}) \triangleq \max_{\|\boldsymbol{\epsilon}\|_p \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \boldsymbol{\epsilon}), \tag{7}$$

where $p \in [1, \infty[$ is the p-norm used (usually $p = 2$) and $\rho > 0$ is a hyperparameter [15]. This optimization objective arises from the PAC-Bayesian generalization bound shown in Equation (8) [73], and Foret et al. [15] use it to show Equation (9). 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 $1 - \delta$.

$$\mathbb{E}_{\boldsymbol{w} \sim \mathscr{Q}}[L_{\mathscr{D}}(\boldsymbol{w})] \leq \mathbb{E}_{\boldsymbol{w} \sim \mathscr{Q}}[L_{\mathcal{S}}(\boldsymbol{w})] + \sqrt{\frac{KL(\mathscr{Q} \| \mathscr{P}) + \log \frac{n}{\delta}}{2(n-1)}}, \tag{8}$$

$$L_{\mathscr{D}}(\boldsymbol{w}) \leq \max_{\|\boldsymbol{\epsilon}\|_2 \leq \rho} L_{\mathcal{S}}(\boldsymbol{w} + \boldsymbol{\epsilon}) + \sqrt{\frac{k \log \left(1 + \frac{\|\boldsymbol{w}\|_2^2}{\rho^2} \left(1 + \sqrt{\frac{\log(n)}{k}}\right)^2\right) + 4 \log \frac{n}{\delta} + \tilde{O}(1)}{n-1}}. \tag{9}$$

Since its introduction, SAM has inspired numerous follow-up studies focused on improving computational efficiency [54–59] as well as providing theoretical insights into its efficacy [60–63].

E Training GLGNs

Algorithm 1 Training STE-GLGNs

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

F 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 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 [74].

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

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

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 [67], 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 [75]. 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.

G 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 [76], 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.

H Experimental Configuration

H.1 Hyperparameters

Gap Scaling with Depth we use the same hyperparameters for this experiment for DLGNs 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. [7] 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.

H.2 Implementation

Our code extends the official PyTorch `Difflogic` library by Felix Petersen, i.e., the reference implementation provided alongside the Differentiable LGN paper [7]. 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.

I Softmax DiffLogic Performance on MNIST-like Baselines

For all datasets, we use the same model and experiment configs.[2] The models have six layers and a width of 64k. See Table 3 for the results.

Besides the classic MNIST [77, 78], CIFAR-10, and CIFAR-100 datasets [79], we also evaluate EMNIST (balanced and letters) [80], FashionMNIST [81], KMNIST [82], and QMNIST [83]. 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.

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

Table 3: The table shows the performance of DLGNs 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 [79]	100.0 %	100.0 %	0.0 %	52.04 %	50.72 %	1.31 %
CIFAR-100 [79]	83.44 %	80.39 %	3.05 %	23.86 %	23.1 %	0.76 %
EMNIST balanced [80]	95.52 %	94.87 %	0.65 %	84.57 %	84.28 %	0.29 %
EMNIST letters [80]	98.69 %	98.3 %	0.38 %	91.43 %	91.04 %	0.38 %
FashionMNIST [81]	99.02 %	98.17 %	0.85 %	90.37 %	90.0 %	0.36 %
KMNIST [82]	100.0 %	100.0 %	0.0 %	97.14 %	97.0 %	0.14 %
MNIST [77]	100.0 %	100.0 %	0.0 %	98.33 %	98.16 %	0.17 %
QMNIST [83]	100.0 %	100.0 %	0.0 %	98.33 %	98.17 %	0.16 %

J Expected Entropy

Lemma 4. *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 $p = (p_1, p_2, ...)$ 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 DLGNs 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%

K Runtime

In Table 4 the number of iterations per hour Softmax and Gumbel LGNs completed while training models for the results in Figure 3. 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.

L 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.