Light Differentiable Logic Gate Networks

Anonymous authors
Paper under double-blind review

Abstract

Differentiable logic gate networks (DLGNs) exhibit extraordinary efficiency at inference while sustaining competitive accuracy. But vanishing gradients, discretization errors, and high training cost impede scaling these networks. Even with dedicated parameter initialization schemes from subsequent works, increasing depth still harms accuracy. We show that the root cause of these issues lies in the underlying parametrization of logic gate neurons themselves. To overcome this issue, we propose a reparametrization that also shrinks the parameter size logarithmically in the number of inputs per gate. For binary inputs, this already reduces the model size by 4x, speeds up the backward pass by up to 1.86x, and converges in 8.5x fewer training steps. On top of that, we show that the accuracy on CIFAR-100 remains stable and sometimes superior to the original parametrization.

1 Introduction

Contemporary large, overparametrized neural networks have demonstrated remarkable expressivity (Allen-Zhu et al., 2019), but their computational cost necessitates efficiency improvements while sustaining their approximation accuracy (Gusak et al., 2022). With that goal, several approaches directly draw from the physical structure of the underlying hardware to parametrise model classes (Wang et al., 2020; Benamira et al., 2024; Bacellar et al., 2024; Hubara et al., 2016). Among them, differentiable logic gate networks maintain an unparalleled performance-efficiency trade-off (Petersen et al., 2022). Subsequent works have since advanced this model to convolutional or recurrent architectures (Petersen et al., 2024; Bührer et al., 2025). Yet, several issues like vanishing gradients, discretization errors, and high training cost impede scaling these models in depth.

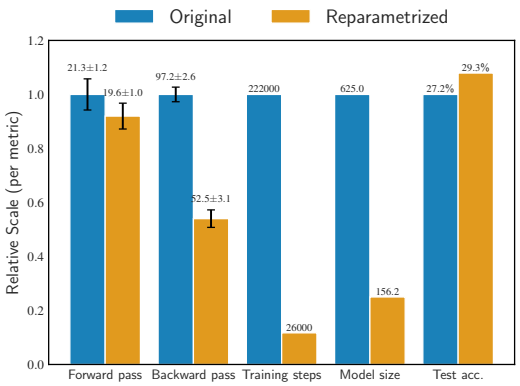


Figure 1: For a CIFAR-10 DLGN (Petersen et al., 2022), our reparametrized DLGNs require 4x less memory, converge in 8.5x fewer training steps, and perform the forward and backward passes in up to 8% and 45% less time, respectively. Details in Section 5 and Appendix B.4.

So far, prior works have mainly patched these problems with alternative parameter initialization schemes (Petersen et al., 2024; Yousefi et al., 2025). But these remedies do not fully resolve the issues, as they neglect that the primary root cause lies in the parametrization of logic gate neurons themselves. For that reason, scaling the convolutional DLGN from Petersen et al. (2024) in depth still grossly degrades its discretized accuracy (cf. Figure 4b).

In this work, we tackle the DLGN parameterization, study how it gives rise to the problems mentioned above, and propose a reparametrization that overcomes the problems; the reparametrization is illustrated in Figure 2. Over and above, we explicate the impact that initializations have on gradient stability and optimization dynamics in deep logic gate networks. In particular, we identify RIs as proposed by Petersen et al. (2024) as one of the

simplest instances of a larger class of negation-asymmetric heavy-tail initializations, and elucidate why such initialization schemes are particularly beneficial for the information flow in both the forward and backward pass during training. Combining such initializations with the reparametrization, we overcome the issues mentioned above and obtain logic gate networks that are more expressive, more scalable, and more efficient to train (cf. Figure 1).

Petersen et al. (2022) showed that DLGNs can process one million MNIST or CIFAR-10 images per second on a single CPU core, Petersen et al. (2024) later showed that convolutional DLGNs take less than 10 nanoseconds per CIFAR-10 image on an FPGA, and Bührer et al. (2025) showed that recurrent DLGNs require 20'000 times fewer logic operations to deliver performance comparable to RNN, GRU, and Transformer-based models in the WMT'14 German to English translation task (Bojar et al., 2014). These values show that DLGNs are very suitable for real-world deployment once the accuracy matches state-of-the-art models. To facilitate the research needed to close this accuracy gap, our reparametrization makes training more efficient without altering the inference dynamics that make DLGNs attractive. We find that models require 4x less VRAM to train, process backward passes up to 1.86x faster, and 8.5x fewer training steps.

2 Background on Logic Gate Networks & Related Work

2.1 Logic Gate Networks

In essence, differentiable logic gate networks differ from feed-forward neural networks in the parametrization of each neuron. In standard architectures, each neuron is a composition of vector-algebraic operations with non-linear activation functions (Fukushima, 1980; Schmidhuber, 2015; LeCun et al., 2015; Goodfellow et al., 2016). By contrast, differentiable logic gate networks (DLGNs) associate each neuron with a binary Boolean function $G : \{0,1\}^2 \to \{0,1\}$ (Petersen et al., 2022). That way, each neuron is connected to only two neurons in the previous layer. Combined with bit-level operations, this extreme sparsity renders DLGNs particularly suitable for high-performance inference on devices with low computational resources.

Adhering to the canonical ordering of Boolean functions (cf. Table 1), we denote the 16 binary Boolean functions by $G_i, 1 \leqslant i \leqslant 16$. We categorize these functions based on the number of non-zero outputs into four ANDs, four ORs, two constants, two XORs, and four pass-throughs, which merely forward one of the inputs, negated or non-negated. A layer of such neurons is referred to as the logic layer.

Naturally, the space of Boolean functions and the functions themselves are discrete, and thus do not immediately give rise to differentiable neurons. To apply gradient-based optimization methods, the original paper proposed to continuously relax each neuron to the probability simplex over all 16 functions (Petersen et al., 2022),

$$g(p,q) := \sum_{i=1}^{16} \omega_i g_i(p,q), \quad p,q, \in [0,1], \quad \omega_i \geqslant 0, \ \sum_j \omega_j = 1. \tag{1}$$

where each function g_i is a probabilistic surrogate of the deterministic G_i, defined as

$$g_i(p,q) := \mathop{\mathbb{E}}_{\substack{A\sim \mathrm{Ber}(p),\\ B\sim \mathrm{Ber}(q)}} [G_i(A,B)], \quad p,q \in [0,1]. \tag{2}$$

Such a surrogate is necessary to deal with real-valued inputs p, q during training, for which the underlying G_i are not defined. Accordingly, we will refer to ω_i as the weight of g_i.

Moving back to the parameters of each neuron, the authors decided to initialize the weights in Equation (1) via a softmax of i.i.d. random variables

$$\omega_i = \frac{\exp(\Omega_i)}{\sum_j \exp(\Omega_j)}, \quad \Omega_i \overset{i.i.d.}{\sim} \mathcal{N}(0, \sigma^2). \tag{3}$$

Likewise, a softmax operation is used to eventually obtain differentiable class scores from this network for classification tasks. In particular, for C classes, a layer coined GroupSum

partitions the gate outputs of the final logic layer into C contiguous bins and accumulates them to obtain the corresponding logits.

At inference, all these softmax operations are replaced by argmax operations. This rounds each neuron to the binary gate g_i with the highest weight ω_i, which yields a logic gate circuit that can be directly embedded in hardware such as FPGAs or ASICs (Zia et al., 2012). Naturally, this rounding operation entails a discretization error that might further reduce performance at deployment. We hence refer to both versions of the network as the continuous and discretized DLGN.

Contending with both this discretization error and vanishing gradients, Petersen et al. (2024) observed superior performance when they replaced the Gaussian initialization in Equation (3) by an RI, which deterministically assigns a high initial weight to the pass-through gate function $G_4(A, B) = A$,

$$\Omega_i = \begin{cases} 5, & i = 4 \\ 0, & i \neq 4 \end{cases}, i = 1, \ldots, 16. \tag{4}$$

Similar to the original idea of residual connections (He et al., 2016), this pass-through bias stabilized training. On top of that, it notably reduces the number of non-trivial logic gates that remain after the discretized DLGN undergoes a logic simplification. That way, they obtained a logic gate circuit that achieves a test accuracy of 85% on CIFAR-10 with less than 29 million gates, which is far less than what competitive networks required (Petersen et al., 2024, Sec. 5.1).

Albeit effective, this initialization is still subject to limitations that arise from the underlying parametrization, which we will pinpoint in Section 3. But first, we present other related work and explain how they differ from DLGNs in both their reparametrized and original form.

2.2 Other Related Work

Several works have exploited that learning circuits of logic gates with more than two inputs allows for embedding more functional expressivity on the same hardware (Umuroglu et al., 2020; Bacellar et al., 2024). On the contrary, DLGNs were practically limited to learn logic gates with very few inputs, as processing 2^{2^n} parameters per logic gate with n inputs quickly becomes intractable. With our reparametrization that reduces the number of parameters to 2^n, advancing DLGNs to process more than two inputs per gate becomes a viable option.

In contrast to our reparametrization, these works do not directly estimate the outputs of the logic gates. Instead, they use a different representation class and quantize this class to logic gates after training. However, these indirect representations either fall short of exploiting the expressivity of logic gates (Umuroglu et al., 2020) or are costlier to parametrize (Andronic & Constantinides, 2023; 2025). We provide a detailed comparison in Appendix G.

3 Reparametrizing Logic Gate Neurons

3.1 Weaknesses of the current parametrization

We demonstrate that redundancies in the parametrization are the primary cause of vanishing gradients and large discretization errors.

3.1.1 Gradient stability

Each Boolean function G_i has a negated counterpart. Adhering to the canonical ordering of Boolean functions (cf. Table 1), we denote this counterpart by $G_{\neg i} := G_{17-i} \equiv \mathbf{1} - G_i \equiv \neg G_i$. The same holds for the probabilistic surrogates g_i. Under this condition, choosing independent weights for each g_i and its negated counterpart $g_{\neg i}$ is fatal, as it provokes self-cancellations in the partial derivatives, progressively diminishing the gradient norm during backpropagation.

To see this, we equally denote $\omega_{\neg i} := \omega_{17-i}$ to expose the symmetry in Equation (1) as

$$g(p,q) := \sum_{i=1}^{8} \omega_i g_i(p,q) + \sum_{i=1}^{8} \omega_{\neg i} g_{\neg i}(p,q) \tag{5}$$

$$= \sum_{i=1}^{8} \omega_i g_i(p,q) + \omega_{\neg i}(1 - g_i(p,q)). \tag{6}$$

Having i.i.d. ω_i, this translates to a weighted sum of sign-symmetric random variables in the partial derivatives

$$\frac{\partial g(p,q)}{\partial p} = \sum_{i=1}^{8} (\omega_i - \omega_{\neg i}) \frac{\partial g_i(p,q)}{\partial p}. \tag{7}$$

Initializing Ω_i with the default variance $\sigma = 1.0$ will concentrate the gradient norm around 0 (cf. Figure 17a) and entail vanishing gradients with high probability, as Petersen et al. (2024) have already encountered. Notably, even with a variance as large as $\sigma^2 = 16.0$, many partial derivatives remain concentrated at zero (cf. Figure 17b).

RIs as proposed by Petersen et al. (2024) successfully break this sign-symmetry to

$$\frac{\partial g(p,q)}{\partial p} = \sum_{i=1}^{8} (\omega_i - \omega_{\neg i}) \frac{\partial g_i(p,q)}{\partial p} \overset{(3)}{=} \sum_{i=1}^{8} \frac{e^{\Omega_i} - e^{\Omega_{\neg i}}}{\sum_j e^{\Omega_j}} \frac{\partial g_i(p,q)}{\partial p} \overset{(4)}{=} \frac{e^{\Omega_4} - 1}{e^{\Omega_4} + 15}. \tag{8}$$

Once more, symmetric overparametrization traps RIs in a tension between maintaining gradient stability and stalling optimization for other gate functions (cf. Appendix E.1).

While sign-symmetries interfere with the gradient signal in a destructive way, there are also other redundancies in the parametrization that contribute to the discretization error.

3.1.2 Discretization error

When converting the continuous relaxation to a logic gate circuit, the softmax-to-argmax rounding principle (cf. Section 2.1) discretizes each neuron to the logic gate function with the highest weight ω_i. But with the redundancies in this parametrization, the logic gate that is rounded to is not necessarily the one that the neuron is closest to. For example, assume a neuron with weight 0.4 for the one pass-through gate $G_4(A,B) = A$, weight 0.3 for the other pass-through gate $G_6(A,B) = B$, and weight 0.3 for the OR function $G_8(A,B) = A \vee B$. For the four binary inputs $(0,0),(0,1),(1,0),(1,1)$, the neuron will output $0, 0.6, 0.7, 1$. Clearly, this output behaviour is closest to the OR function G_8, although the argmax is the pass-through gate G_3. Argmax is effective only when applied to inputs that are exclusive and independent.

Redundancies in the parametrization are the leading cause of vanishing gradients. In the following, we present an exact, redundancy-free parametrization.

3.2 Input-wise parametrization

In fact, each binary function $G : \{0,1\}^2 \to \{0,1\}$ has a unique decomposition

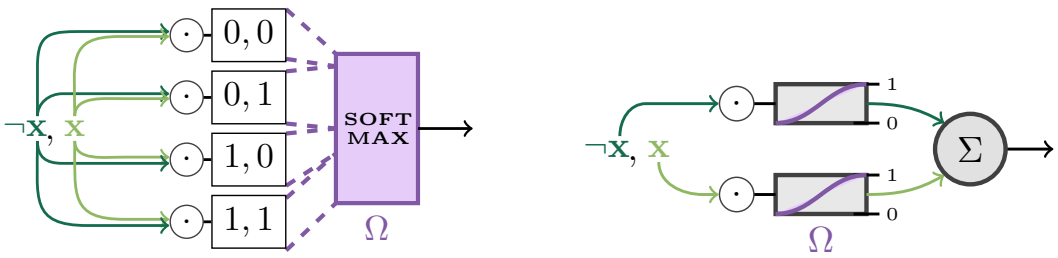
$$G = \alpha_{00} E_{00} + \alpha_{01} E_{01} + \alpha_{10} E_{10} + \alpha_{11} E_{11}, \tag{9}$$

where $\alpha_{ij} \in \{0,1\}$, and E_{ij} is the indicator function $\mathbb{1}\{(k,\ell) = (i,j)\}$. This exact representability also transfers to the probabilistic surrogates

$$g = \alpha_{00} e_{00} + \alpha_{01} e_{01} + \alpha_{10} e_{10} + \alpha_{11} e_{11}, \tag{10}$$

where $e_{ij}(p,q) = \mathbb{E}[E_{ij}(p,q)]$ as in Equation (2). Relaxing the binary coefficients α_{ij} to the continuous interval $\omega_{ij} \in [0,1]$ and rounding back via $\omega_{ij} > 0.5$, we obtain the exact parametrization

$$\begin{aligned} g_\omega(p,q) = {} & (1-p) \cdot (1-q) \cdot \omega_{00} \\ & + (1-p) \cdot q \cdot \omega_{01} \\ & + p \cdot (1-q) \cdot \omega_{10} \\ & + p \cdot q \cdot \omega_{11}. \end{aligned} \tag{11}$$



(a) Original parametrization (b) Input-wise parametrization

Figure 2: Illustrating the reparametrization for logic gates with one input. It requires only 2^n learnable parameters Ω for n inputs, opposed to 2^{2^n} for the original parametrization.

Similar to Petersen et al. (2022), one could learn such a bounded coefficient $\omega_{ij} \in [0, 1]$ by mapping a real parameter $\Omega_{ij} \in \mathbb{R}$ to an activation function $\rho : \mathbb{R} \to [0, 1]$. We will defer the specific function choice to Appendix C.1.1 and stick with the standard sigmoid function for now, i.e. $\rho(x) := \frac{1}{1+\exp(-x)}$.

Since the basis of the class of Boolean functions with n inputs has cardinality 2^n, this equally expressive parametrization requires logarithmically fewer parameters than the softmax parametrization used by Petersen et al. (2022), which assigns an individual parameter to each of the 2^{2^n} Boolean functions. For the class of binary functions used here, this already shrinks the model size by a factor of 4, and renders learning higher-dimensional Boolean functions computationally more viable. We hence also refer to this reparametrization as input-wise parametrization (**IWP**), and use the abbreviation **OP** for the original parametrization.

3.3 No gradient stability without appropriate initializations

We now show that the IWP eliminates the pathways causing gradient cancellations and discretization errors. Any remaining gradient instability arises from other architectural factors, particularly parameter initialization.

To begin with, rounding the outputs of g_ω to their closest binary numbers clearly attains minimal errors with respect to any Minkowski norm and any other norm that is based on a uniform distance metric between outputs of the function. Proof in Appendix E.3.

Moving on with gradient stability, the partial derivative now becomes

$$\frac{\partial g_\omega(p, q)}{\partial p} = (1 - q)(\omega_{10} - \omega_{00}) + q(\omega_{11} - \omega_{01}) \tag{12}$$

$$= \mathop{\mathbb{E}}_{B \sim \mathrm{Ber}(q)} [\omega_{1B} - \omega_{0B}]. \tag{13}$$

An i.i.d. parameter initialization with sufficiently low variance would still entail cancellations, but, opposed to the OP, the IWP itself does not compound this problem further. Heavy-tail initializations that concentrate most weights ω_{ij} close to $0, 1$ would already resolve these cancellations inside a neuron to a sufficient extent, as we explain exhaustively in Appendix E.4. But a heavy tail alone is not enough in general. As long as the parameter initialization treats each function and its negated counterpart independently, gradients will distribute sign-symmetrically between different neurons during backpropagation. The more subsequent gates a neuron passes its output to, the more likely it is that the sum of partial derivatives that it receives during backpropagation will concentrate at 0. Therefore, appropriate initialization schemes should be negation-asymmetric as well.

A residual initialization (RI) as proposed by Petersen et al. (2024) that assigns a high initial bias to the pass-through $G_4(A, B) = A$ is a simple instance satisfying both requirements. More complex instances, like an AND-OR initialization that concentrates each gate either to the AND or OR function, are also feasible in principle. However, it turns out that RIs entail a gate output distribution (cf. Figure 3) that organizes the optimization of logic gate networks consecutively from earlier to later layers, which is advantageous for training deep networks. We substantiate this argument in Appendix E.4.1, where we study the class of heavy-tail, negation-asymmetric initialization schemes in more detail.

To conclude, we pair our IWP with RIs and show that the result is more scalable in depth and expressive complexity.

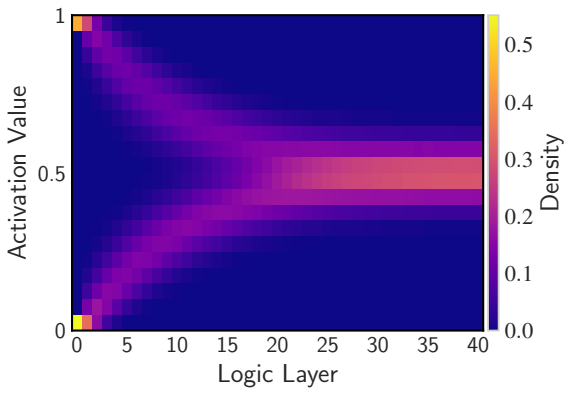


Figure 3: Distribution of gate outputs for an IWP DLGN right after residual initialization (RI), averaged over 100 images of CIFAR-100. That way, RI postpones gate learning in later layers until earlier layers are more refined. This incremental refinement allows to learn complex deep networks.

4 Experiments

To verify our claims of better gradient stability, discretization accuracy, and training efficiency of our IWP, we adopt the original DLGN models and the experimental training setup from Petersen et al. (2022). We also cover the models and experimental setup from Petersen et al. (2024) to show the benefits apply to CLGNs as well.

4.1 Benchmarks

In both works, the networks were evaluated on several image classification benchmarks, with CIFAR-10 (Krizhevsky, 2009) as the most challenging dataset. However, the shallow models used there already perfectly fit the training dataset after a few iterations, which restricts the measurability of further expressive benefits when scaling the networks in depth. Thus, we decided to lift the complexity of the task in two ways: Firstly, we transition to CIFAR-100, a 100-class extension of CIFAR-10 (Krizhevsky, 2009). Secondly, we employ random resized crops and horizontal flips as standard data augmentations (PyTorch Core Team, 2023).

We need to account for the 10-fold class increase in the final prediction head of the model. Apart from that, no further adjustments to the original experimental setup for CIFAR-10 are required. The class increase can be encountered in two different ways. Following recommendations of Petersen et al. (2024, Appendix A.2), we explore both options; see Appendix D.2.1 for details.

As a trade-off between computational feasibility and expressiveness, we finally decided to consider the medium-sized M models for both papers. When we refer to the DLGN and CDLGN in the experiments, we hence always mean the specific CIFAR-10 M architecture from Petersen et al. (2022, Appendix A.1) and Petersen et al. (2024, Appendix A.1.1), respectively. To estimate uncertainty, we train each model on three different seeds.

4.2 Implementation of reparametrization

To implement our IWP and the adjusted initialization schemes in the given Python and CUDA implementation, we merely have to override the weight initialization and the forward and backward functionality according to Equations (11) and (12).

While we assumed the sigmoid function as the binary gate output estimator ρ in Equation (11) for the sake of exposition, we observed slightly superior expressivity with a rescaled sinusoidal estimator $\sin_{01}(x) = 0.5 + 0.5\sin(x)$ and adopted that one for subsequent experiments. See Appendix C.1.1 for details.

4.3 Scaling models in depth

Eventually, we want to reliably assess how increasing model depth affects performance for both parametrizations. To scale both DLGNs and CDLGNs in a comparable, architecture-agnostic way, we introduce a depth-scale parameter $D \in \mathbb{N}$, and obtain depth-scaled networks by placing D (convolutional) logic layers instead, where only one was placed in the original architecture. Appendix D.1 presents implementation details of this depth scaling.

5 Results

5.1 Reparametrization reduces vanishing gradients

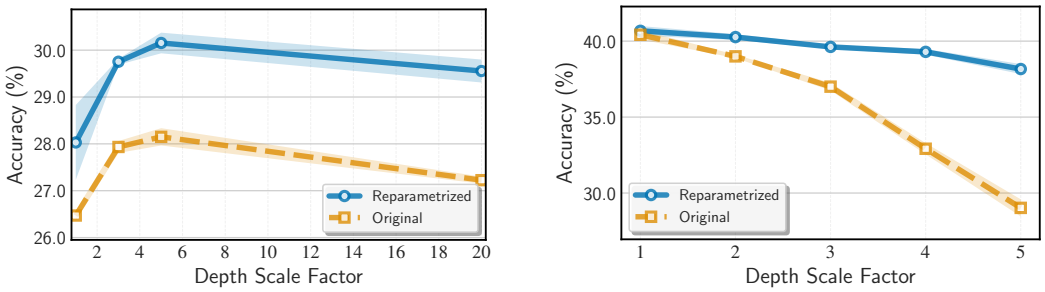
To begin with, vanishing gradients as the major hindrance for scaling DLGNs in depth, the input-wise parametrization drastically reduces the shrinkage of gradient norm as we backpropagate over layers. As Figure 7a showcases, the gradient norm undercuts machine precision after 16 logic layers already, and vanishes to 10^{-34} over 40 layers, when the OP is used. But as already discussed in Section 3.3, an IWP alone without appropriate negation-asymmetric, heavy-tail initializations can not reduce the gradient norm shrinkage sufficiently and also ends up with an average gradient norm of 10^{-16} after 40 layers.

5.2 Residual initializations scale best with depth

The residual initialization (RI), although biasing towards a single gate function only, proves most effective for training deep DLGNs. On the one side, all other single-gate biases quickly concentrate the gate outputs at one value (cf. Figure 21) where their gradients become 0 and stifle gradient flow (cf. Figure 18b). On the other side, some multi-gate biases appeared competitive alternatives to RI, such as the AND-OR initialization, which exerts a theoretically more appealing anticoncentration that retains inputs close to 0 and 1 over the layers (cf. Figure 21e). Nonetheless, these methods remain slightly inferior to RI in terms of both gradient stability (cf. Figure 18a) and accuracy (cf. Figure 20a). While the former drawback is rather obvious because the pass-through gate G_4 is unparalleled in retaining a uniformly high gradient of 1, the latter relates to the more intricate discrepancy in the optimization dynamics that each of the two initialization schemes gives rise to. As discussed in Section E.4.2, RIs order optimization of neurons from earlier to later layers. On the contrary, AND-OR initializations allow for non-uniform updates of he four gate outputs for neurons at later layers right from the beginning. This additional freedom ,however ,seems not only detrimental to the accuracy of the continuous relaxation. Surprisingly, despite its anti-concentration, this alternative initialization grossly compounds to the discretization error as we further increase depth (cf. Figure 20b).

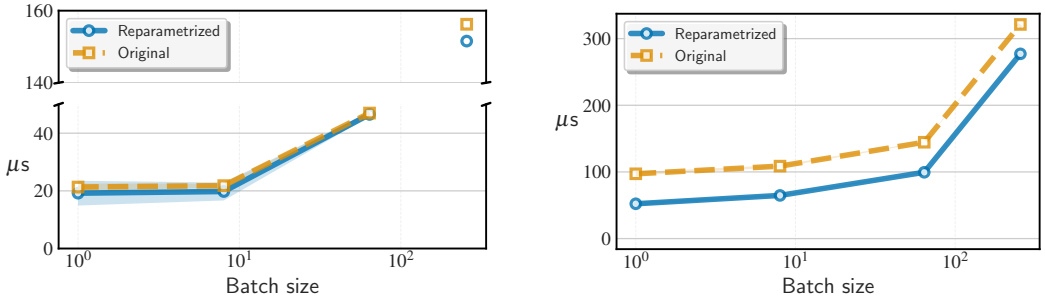
5.3 Original parametrization scales worse with depth

RIs also suppress the undesirable properties of the OP but cannot fully level them out, demonstrating that an inappropriate underlying parametrization can irreversibly condition shortcomings for optimization. The IWP addresses these weaknesses, and pairing it with an RI achieves superior performance to the OP with an RI. For one, the IWP with RI still retains a higher gradient norm than the OP with RI (cf. Figure 7b). For another, we observe a clear gap in the predictive performance as well. For the DLGN, this gap is already apparent with baseline depth scale $D = 1$. Moreover, the IWP consistently maintains this gap, which the OP cannot close even with 20-fold depth (cf. Figure 4a) and eventually plateaus at roughly 28% test accuracy. For the CDLGN, the shallow baseline network performs almost equivalently. But increasing depth now begins to expose a drastic



(a) LGN architecture (b) CLGN architecture

Figure 4: Discretized test accuracy, averaged over three seeds, when scaling the CIFAR-10 M DLGN (Petersen et al., 2022) and CDLGN (Petersen et al., 2024) in depth.



(a) Forward pass (b) Backward pass

Figure 5: Training times for the DLGN with 20-fold depth. Mean and standard deviation were computed over 20 batches of CIFAR-100.

performance gap that culminates in a more than 1.3 times better test accuracy of the IWP for $D = 5$ (cf. Figure 4b).

We observe that this gap is mainly attributable to a large discretization error for the OP. Figure 8b shows that the accuracy of the continuous OP CDLGN trails the IWP only by a few percent. Unfortunately, the increasing depth ceases to benefit performance for the IWP as well at some point, at least when not increasing the number of optimization steps. Henceforth, we suppose that this is caused by shared underlying characteristics of the overall DLGN architecture, and discuss potential reasons later in Section 6.

5.4 Training Efficiency

By reducing the number of real parameters per neuron from 16 to 4, we shrink the model size by a factor of 4 (cf. Figure 1). This reduction also reduces the working set size during the forward and backward passes in the CUDA kernel. This advantage becomes particularly apparent for small batch sizes, where the parameter tensors dominate the memory footprint. For an 80-layer DLGN trained with batch size 1, we observe a 1.86x speedup for the backward pass and a 1.11x speedup for the forward pass (cf. Figure 5).

However, for large batch sizes, as they are typically used during training of such large models, the parametrization plays an increasingly negligible role in the overall memory and operation usage, and the relative speedup over the OP fades. We discuss further potential efficiency improvements in Appendix B.4.

Besides the models being lighter, the significant benefit of our IWP lies in the better gradient signal. In Figure 9 (Appendix B.4.1), we see that IWP converges in 8.5x fewer training steps than the OP, and as the steps are slightly faster, this means we can converge more than 8.5x faster in terms of wall clock time.

6 Discussion

IWP DLGNs are not prone to performance degradation for increasing depth, as they mitigate the discretization error and improve gradient stability. However, scaling these networks in depth did not yield large expressivity benefits. And DLGNs still have a considerable generalization gap despite data augmentations. We want to discuss how to overcome both problems in the following, and present further avenues for future research.

6.1 Remaining expressivity bottlenecks in DLGNs

Although scaling DLGNs in depth provides slight benefits, the expressive advantage of deep DLGNs fades beyond a certain depth despite the IWP. This is expected, as the CDLGN baseline with depth $D = 1$ already contains 15 learnable gate layers, comparable to a 4-fold deeper DLGN. Reducing initialization variance does not alleviate this expressivity bottleneck (cf. Figure 6a). Instead, we identify possible bottlenecks in the randomized, fixed connection topology and input preprocessing, causing this limitation. We hypothesize that this limitation does not arise from the expressivity of the model class itself, but rather from information loss in the way that real-valued inputs are discretized to four binary values. We show in Appendix B.5 that convolutional neural networks (CNNs) (Krizhevsky et al., 2012) are worse than CLGN when provided with low-resolution inputs. And there, we find evidence that the random initialization of connections prevents DLGNs from exploiting the structure in the binary encoding scheme. An encoding-aware connection heuristic or even learned connections, as in Bacellar et al. (2024), might overcome this limitation.

6.2 Closing the generalization gap of DLGNs

Even before discretization, IWP DLGNs only slightly outperform the OP on test accuracy, despite a substantial increase in the training set (cf. Figure 8). Dataset augmentations alone do not close this gap, and standard techniques like dropout (Srivastava et al., 2014), random interventions, or residual connections (He et al., 2016) fail to improve test performance (cf. Appendix F). Designing constraints that promote generalizable functionality in DLGNs remains an open problem.

6.3 Learning gates with more inputs

As discussed in Section 2.2, advancing DLGNs to learn logic gates with more than two inputs finally becomes a viable option. Learning a logic gate circuit with six input gates could not only yield expressive benefits, but also result in more efficient hardware embeddings on modern FPGAs that typically admit six inputs to their lookup tables (Bacellar et al., 2024; Zia et al., 2012). We leave this avenue to be explored in future research.

7 Conclusion

We proposed an input-wise parametrization (IWP) of logic gate networks with tailored initializations that allow scaling DLGNs in depth without degrading performance, while reducing parameter count logarithmically in the number of inputs per gate. IWP enables research for learning logic gate circuits that are not only far deeper, but also far more complex in every logic gate itself by increasing the number of gate inputs. Moreover, the IWP substantially shrinks the model size and reduces the training time for small batch sizes. Yet the efficiency gains in training time vanish for large batch sizes, further necessitating performance optimizations to train large-scale DLGNs under ordinary resource constraints.

Finally, closing the generalization gap in DLGNs has become an even more pressing open problem, because the IWP also notably increases the expressivity of the training dataset. But in view of the appealing performance-efficiency trade-off, DLGNs continue to lend themselves for deployment on computationally restricted hardware like real-time systems or edge devices, and advancing their potential thus remains a promising avenue for future research.

Reproducibility

The source code of IWP DLGNs and the associated experiments is provided in the supplementary material. There, a step-by-step guide explains how to set up the runtime environment in which we conducted our experiments, and how to reproduce any particular experiment in this environment. That way, we hope to make our experiment infrastructure as conveniently accessible as possible. To guarantee the reproducibility of our experiments, we restrict PyTorch to deterministic algorithms and fix the seeds of the random number generators that are used for the randomized initialization of weights and connections and for data loading.

For source code and experiments on the convolutional extension of logic gate networks, the source code of Petersen et al. (2024) has not yet been made publicly available. We can hence provide no further details at this time, and we kindly ask the reader to directly correspond with Petersen et al. (2024) for further inquiries.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/62dad6e273d32235ae02b7d321578ee8-Paper.pdf.

Marta Andronic and George A. Constantinides. Polylut: Learning piecewise polynomials for ultra-low latency FPGA lut-based inference. In *International Conference on Field Programmable Technology, ICFPT 2023, Yokohama, Japan, December 12-14, 2023*, pp. 60–68. IEEE, 2023. doi: 10.1109/ICFPT59805.2023.00012. URL https://doi.org/10.1109/ICFPT59805.2023.00012.

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)*, pp. 208–216. IEEE, 2025. doi: 10.1109/FCCM62733.2025.00077.

Marta Andronic, Jiawen Li, and George A. Constantinides. Polylut: Ultra-low latency polynomial inference with hardware-aware structured pruning. *IEEE Trans. Computers*, 74 (9):3181–3194, 2025. doi: 10.1109/TC.2025.3586311. URL https://doi.org/10.1109/TC.2025.3586311.

Alan Tendler Leibel Bacellar, Zachary Susskind, Mauricio Breternitz Jr, Eugene John, Lizy Kurian John, Priscila Machado Vieira Lima, and Felipe M.G. França. Differentiable weightless neural networks. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 2277–2295. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/bacellar24a.html.

Adrien Benamira, Thomas Peyrin, Trevor Yap, Tristan Guérand, and Bryan Hooi. Truth table net: Scalable, compact and verifiable neural networks with a dual convolutional small boolean circuit networks form. In Kate Larson (ed.), *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence, IJCAI-24*, pp. 13–21. International Joint Conferences on Artificial Intelligence Organization, 8 2024. doi: 10.24963/ijcai.2024/2. URL https://doi.org/10.24963/ijcai.2024/2. Main Track.

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. Findings of the 2014 workshop on statistical machine translation. In Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Christof Monz, Matt Post, and Lucia Specia (eds.), *Proceedings of*

the Ninth Workshop on Statistical Machine Translation, pp. 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-3302. URL https://aclanthology.org/W14-3302/.

Simon Bührer, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Recurrent deep differentiable logic gate networks, 2025. URL https://arxiv.org/abs/2508.06097.

Hugo C.C. Carneiro, Felipe M.G. França, and Priscila M.V. Lima. Multilingual part-of-speech tagging with weightless neural networks. *Neural Networks*, 66:11–21, 2015. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2015.02.012. URL https://www.sciencedirect.com/science/article/pii/S0893608015000465.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36: 193–202, 1980.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Julia Gusak, Daria Cherniuk, Alena Shilova, Alexandr Katrutsa, Daniel Bershatsky, Xunyi Zhao, Lionel Eyraud-Dubois, Oleh Shliazhko, Denis Dimitrov, Ivan Oseledets, and Olivier Beaumont. Survey on efficient training of large neural networks. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 5494–5501. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/769. URL https://doi.org/10.24963/ijcai.2022/769. Survey Track.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.90.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, pp. 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. ISBN 9781107038325. First edition published June 2014; corrected via arXiv version 2021 (arXiv:2105.10386).

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/0d3496dd0cec77a999c98d35003203ca-Abstract-Conference.html.

Felix Petersen, Hilde Kuehne, Christian Borgelt, Julian Welzel, and Stefano Ermon. Convolutional differentiable logic gate networks. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/db988b089d8d97d0f159c15ed0be6a71-Abstract-Conference.html.

PyTorch Core Team. *PyTorch Documentation*. PyTorch, 2023. https://pytorch.org/docs/.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Yaman Umuroglu, Yash Akhauri, Nicholas J Fraser, and Michaela Blott. Logicnets: Co-designed neural networks and circuits for extreme-throughput applications. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*, pp. 291–297, Los Alamitos, CA, USA, sep 2020. IEEE Computer Society.

V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971. doi: 10.1137/1116025. URL https://doi.org/10.1137/1116025.

Erwei Wang, James J. Davis, Peter Y. K. Cheung, and George A. Constantinides. Lutnet: Learning fpga configurations for highly efficient neural network inference. *IEEE Transactions on Computers*, 69(12):1795–1808, 2020. doi: 10.1109/TC.2020.2978817.

Shakir Yousefi, Andreas Plesner, Till Aczel, and Roger Wattenhofer. Mind the gap: Removing the discretization gap in differentiable logic gate networks, 2025. URL https://arxiv.org/abs/2506.07500.

Razia Zia, Muzaffar Rao, Arshad Aziz, and Pervez Akhtar. Efficient Utilization of FPGA Using LUT-6 Architecture. *Applied Mechanics and Materials*, 241-244:2548–2554, 12 2012. doi: 10.4028/www.scientific.net/AMM.241-244.2548.

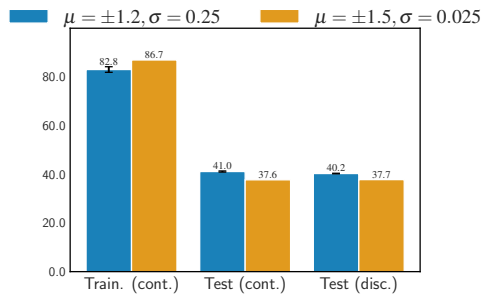
A Usage of LLMs

We have used LLMs to polish the writing of this paper and for code generation through chats, Cursor, and Claude code. ChatGPT, Claude, Gemini, and Grammarly were employed for spellchecking, refining and condensing text, and reviewing to improve clarity and readability. Furthermore, ChatGPT, Claude, and Cursor were used to assist with code completion and generate visualizations. These tools served as auxiliary aids for writing and implementation, while all core research ideas, experimental design, and interpretation of results are our own.

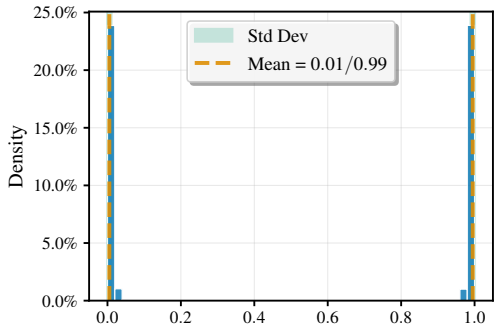
B Further Experiment Results

B.1 Deep networks do not require lower initialization variance

Deeper IWP DLGNs with RIs do neither converge faster nor improve test accuracies (cf. Figure 6a) when lowering the initialization variance and concentrating the weights ω_{ij} closer to $0, 1$, as illustrated in Figure 6b. We believe that this is also attributable to the implicit organization of neuron optimization for RIs (cf. Appendix E.4.2).

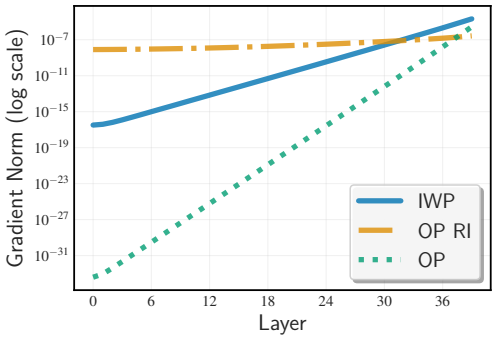


(a) Accuracy of CDLGN with 2-fold depth on CIFAR-100

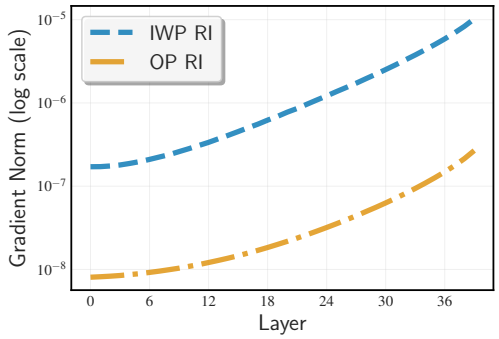


(b) More concentrated distribution for $\mu = 1.5, \sigma = 0.125$

Figure 6: Reducing the initialization variance by concentrating the weights ω_{ij} even further at the tails $0, 1$ for deeper models does not improve performance.



(a) IWP with inappropriate initialization



(b) IWP with RIs

Figure 7: Gradient norm decrease of an IWP DLGN with 40 layers.

B.2 Vanishing Gradients in Deep DLGNs

While the IWP eliminates cancellations inside a neuron, cancellations between partial derivatives of different neurons are out of the control of the parametrization. For that reason, IWP alone does not reduce the gradient norm shrinkage sufficiently, and also ends up with an average gradient norm of 10^{-16} after 40 layers. Avoiding this requires appropriate negation-asymmetric, heavy-tail initializations as already discussed in Section 3.3.

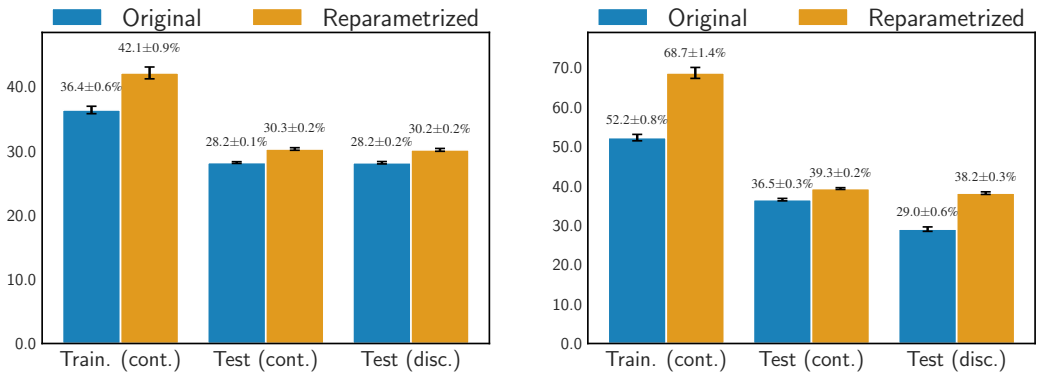
B.3 Discretization error of OP

The discretization error is one major reason for the performance decrease of the OP for deeper models. For five-fold depth, the discretization gap is already substantial for both the DLGN and CDLGN architecture (cf. Figure 8).

B.4 Training Efficiency

B.4.1 Faster convergence of IWP

We show in Figure 9 a roofline plot (running maximum) of the test accuracy for 20-fold depth models under our IWP and the OP. We show in red the maximum accuracy of the OP, which is achieved after 222000 steps. Meanwhile, our IWP achieves this after only 26000 steps. Thus, we can converge 8.5x faster in the number of training steps. As shown in Figure 5, the steps under IWP are as fast or faster than the OP. Thus, we can also train more than 8.5x faster in terms of wall-clock time.



(a) LGN architecture (b) CLGN architecture

Figure 8: Accuracies of the DLGN and CDLGN with five-fold depth on CIFAR-100

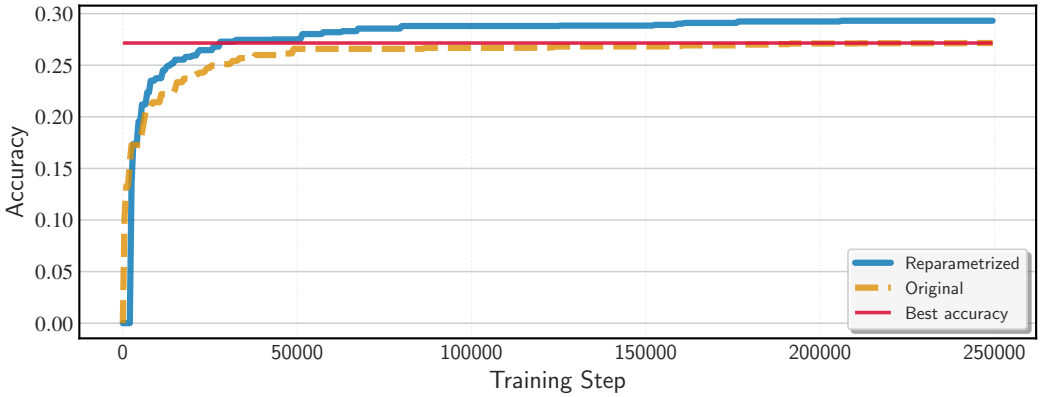


Figure 9: For the DLGN with 20-fold depth, we juxtapose the best discretized accuracy that has been achieved so far during training for both parametrizations. The OP reaches its best accuracy after 222000 steps, which is indicated by the red roofline. The IWP already surpasses this accuracy after only 26000 steps. It hence achieves more than 8.5x faster convergence.

B.4.2 Minimal Efficiency Impact of Gate Output Estimator

We observe that the choice of the gate output estimator $\rho(x)$ has a noticeable impact on both the runtime of the forward computation, but only a minimal effect on the backward pass. We compare the sinusoidal gate output estimator $\rho(x) = 0.5+0.5\cdot\sin(x)$ with a custom double-capped linear $\rho(x) = \max(0, \min(1, x))$, whose gradient is set to 1 throughout. This not only avoids arithmetic operations during the forward and backward pass, but it also alleviates memory requirements because the constant gradient does not require saving the particular input tensor for the backward pass. Although the forward pass speeds up by 22%, the computationally dominant runtime of the backward pass reduces only by 4% (cf. Figure 10). After all, we have not measured whether a linear straight-through estimator can meet the performance of the sinusoidal estimator.

B.5 Information Loss in Preprocessing Stage

To convert the real-valued inputs $x \in [0, 1]$ to binary encodings, Petersen et al. (2022) adopt the thermometer encoding $x_{th} := (x > t_1, t > t_2, \ldots, x > t_k)$, where $t_i = i/k + 1$ are evenly spaced thresholds in $[0, 1]$ (Carneiro et al., 2015). The number of thresholds directly determines the discretization resolution, and an increase should hence further decrease the

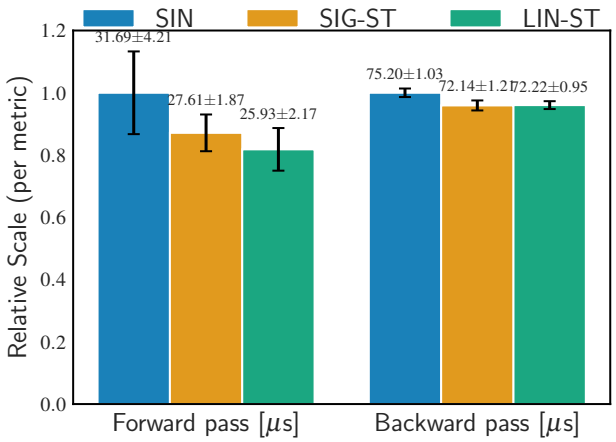
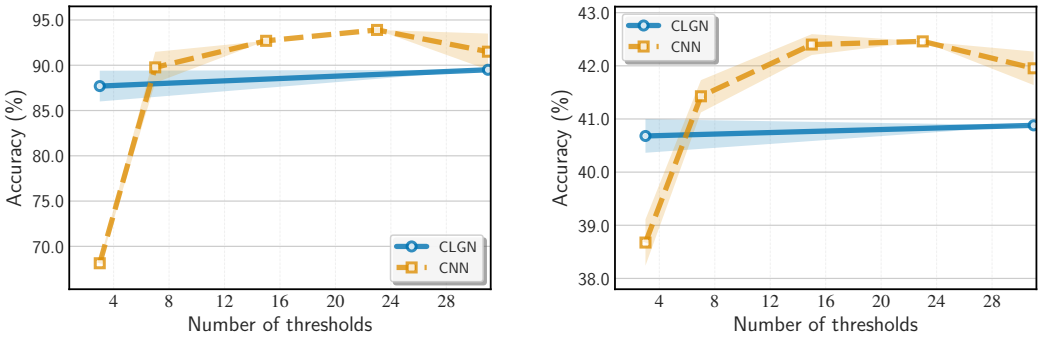


Figure 10: Runtime of the forward and backward pass for an IWP DLGN with different gate output estimators. The default sinusoidal estimator (SIN) is compared to a straight-through sigmoid (SIG-ST) and the linear straight-through estimator (LIN-ST) as introduced in Appendix B.4.2. Measurements are taken for a DLGN of 20-fold depth and are averaged over 20 batches with 25 CIFAR-100 instances.



(a) Training accuracy (cont.)

(b) Test accuracy (disc.)

Figure 11: Approximation improvement for increased resolution in the thermometer encoding for the CDLGN and a standard CNN architecture.

approximation error. For a standard CNN architecture (cf. Figure 12), there is indeed a noticeable improvement. But for the convolutional DLGN, such an improvement fails to appear (cf. Figure 11).

Since the DLGN architecture does not lag behind the CNN architecture in expressivity (cf. Figure 11a), we hence locate the bottleneck in the random, fixed initialization of connections. In the early layers, an encoding-aware connection heuristic or even learned connections as in Bacellar et al. (2024) might overcome this limitation.

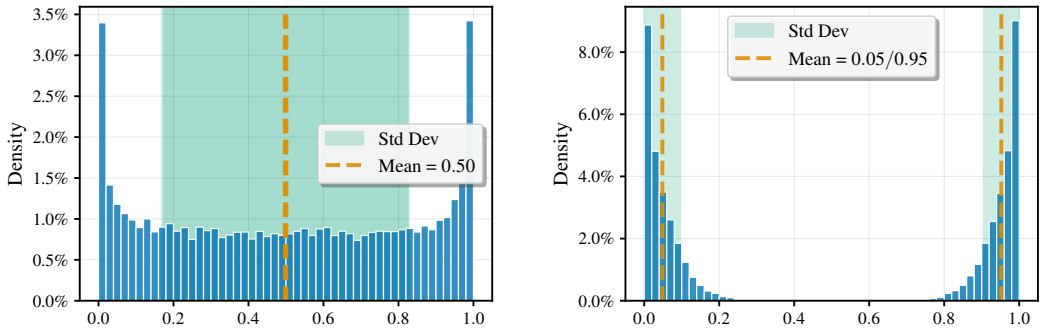
C Implementation Details for Reparametrization

C.1 Estimation Function of Logic Gate Outputs

While the OP slightly benefited from weight decay (Petersen et al., 2024), we have to disable it for our IWP. The reason is that for both the sigmoid and sinusoidal estimators, a weight w close to 0 corresponds to an undecisive gate output $\omega \simeq 0.5$. Weight decay hence actively encourages a high discretization error and entails weaker performance at inference.

Layer #	Description
1	Conv2D (in_channels=93, out_channels=256, kernel=3x3, padding=1)
2	ReLU
3	Conv2D (in_channels=256, out_channels=512, kernel=3x3, padding=1)
4	ReLU
5	MaxPool2D (kernel=2x2, stride=2)
6	AdaptiveAvgPool2D (output_size=1x1)
7	Flatten
8	Linear (in_features=512, out_features=256)
9	ReLU
10	Linear (in_features=256, out_features=100)

Figure 12: CNN Architecture built via `torch.nn` for CIFAR-100 classification in Figure 11.



(a) $\mu = 0.0, \sigma = 1.0$ (b) $\mu = 1.2, \sigma = 0.25$

Figure 13: Initial distribution of coefficients ω_{ij} when initialized with and without a RI for the sinusoidal output estimator, i.e. $\omega_{ij} = 0.5 + 0.5 \cdot \sin(\Omega_{ij}), \Omega_{ij} \sim \mathcal{N}(\mu, \sigma)$.

C.1.1 Sinusoidal estimator

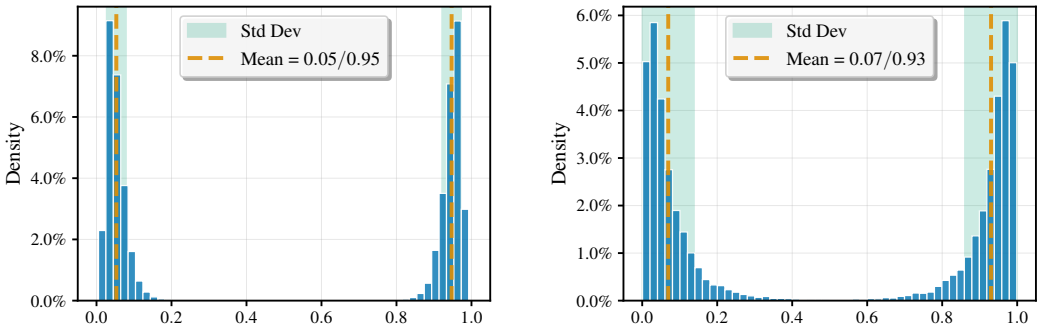
Heavy-tail initializations as motivated in Section 3.3 can be adjusted by adjusting shift and variance of the normal initialization. We choose $\mu = 1.2$ and $\sigma = 0.25$, which results in a distribution like Figure 13b.

C.1.2 Sigmoid estimator

For the sigmoid estimator that is more commonly used in logistic regression, we can similarly adopt heavy-tail initializations by shifting the weights Ω_{ij} by 3.0 (cf. Figure 14).

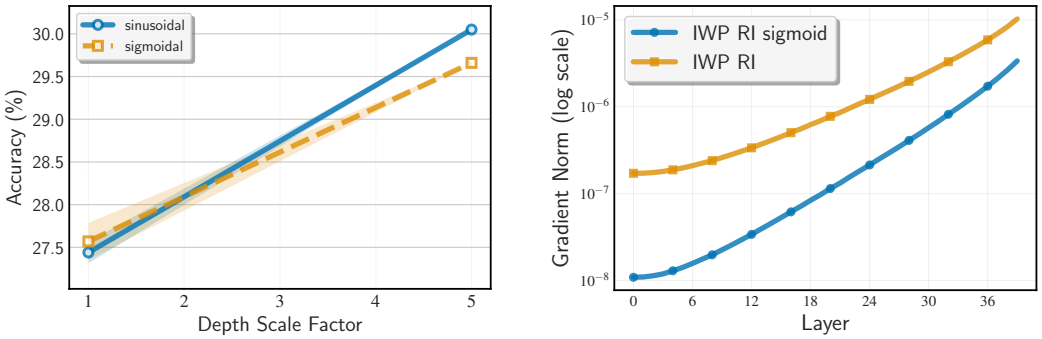
C.1.3 Performance and gradient stability

Although the sigmoid function has been widely adopted for its theoretically desirable properties, its gradients vanish faster for large input values. At the same time, the periodicity of the sinusoidal estimator avoids such a dead end. But for a heavy-tail initialization as in Figure 14a that does not shift the weights too strongly into this flat region, the gradient is still sufficiently high to allow deviation from the initialization region. Although the gradient norm is initially smaller across layers compared to the sinusoidal (cf. Figure 15b), we observe that the gradient norm recovers quickly after a few batches only and approaches the curve of the sinusoidal. However, logic gate networks with a sinusoidal estimator still achieve slightly superior accuracies (cf. Figure 15a), which is why we eventually stuck with them.



(a) $\mu = 3.0, \sigma = 0.5$ (b) $\mu = 3.0, \sigma = 1.0$

Figure 14: Initial distribution of coefficients ω_{ij} when initialized with and without a RI for the sigmoid output estimator, i.e. $\omega_{ij} = (1 + \exp(\Omega_{ij}))^{-1}, \Omega_{ij} \sim \mathcal{N}(\mu, \sigma)$.



(a) Performance comparison. (b) Average gradient norm over layers.

Figure 15: Performance and gradient stability comparison for the sigmoid and the sinusoidal gate output estimator.

D Experiment Infrastructure

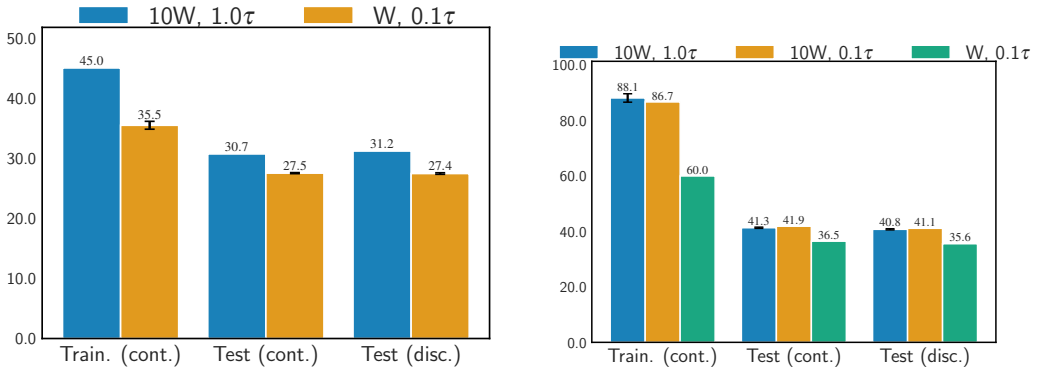
D.1 Scaling DLGNs in depth

As the original DLGN uses a uniform width for all logic layers, we can simply scale the DLGN in depth by placing D logic layers everywhere a logic layer was placed in the original architecture.

For the CDLGN architecture, we place a block of D convolutional logic layers instead of one, but apply the max pooling layer only once at the end. Because the kernel size, padding, and stride in the original architecture (Petersen et al., 2024, Sec. 3.4) preserve the spatial dimensions of the data tensors, no further adjustments are needed. As for the original DLGN, channel increases and decreases are only performed once at the initial and final convolutional logic layer of the block. Finally, we do not restrict the CDLGN architecture to partition the range of channels into separate, independent streams as motivated by Petersen et al. (2024, Sec. 3.4) for more efficient hardware embeddings and data movement during training, but allow connections to be formed between any combination of channels.

D.2 Deviations from original experimental setup

Scaling DLGNs in depth increases the overall computational cost for training. To ensure that gradient descent converges even for deep models, we increase the number of training iterations from 200,000 to 250,000. Furthermore, when training sufficiently deep CDLGNs, GPU memory limitations hinder us from loading batches of original size 100. To ensure



(a) LGN architecture

(b) CLGN architecture

Figure 16: Performance comparison for different final logic layer widths and temperatures in class score accumulation.

comparable optimization conditions for these models, we hence employ batch accumulation for depths $D \geqslant 4$. In particular, we accumulate four batches of size 25 in one backward pass for depths $D = 4, 5$, and tested that it behaves identically to training on the original batch size 100.

D.2.1 10-fold class increase for CIFAR-100

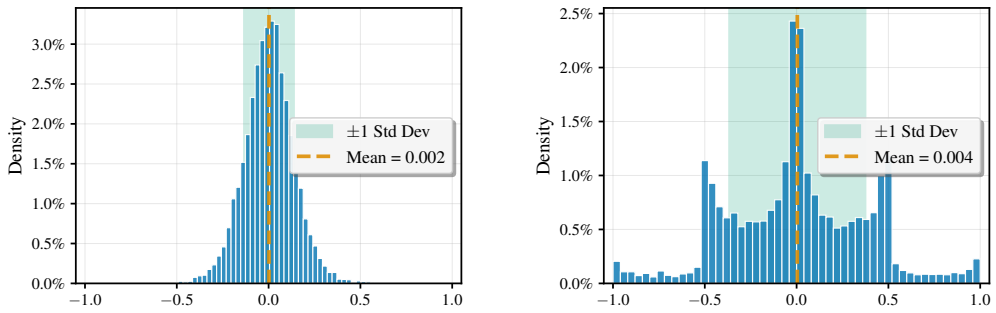
The 10-fold class increase can be encountered in two different ways: On the one hand, one could keep the final logic layer unchanged and accumulate 10 times fewer gate outputs per class in the GroupSum layer. Petersen et al. (2024) proposed the heuristic to shrink the softmax temperature by the square root of the class increase $\sqrt{10}$ in such a case for optimal performance. On the other hand, one could increase the final logic layer to 10-fold width, which does not change the number of gate outputs per class and hence does not require any temperature adjustment.

For both the DLGN and CDLGN, increasing the width 10-fold further improves performance (cf. Figure 16). At the same time, decreasing the temperature as proposed by Petersen et al. (2024) indeed maintained optimal performance, with only minor changes when decreasing the temperature further by $\sqrt{10}$ (cf. Figure 16b).

But for our experiments, we do not consider the choice between keeping the width and decreasing the temperature or increasing the width to keep the absolute temperature a crucial one. The reason is that we merely focus on different parametrizations of each neuron that leave their functional characteristics unchanged. We hence do not expect the trends that we observe when scaling these networks in depth to alter across these slightly varying widths of the final logic layer. To cover both options, we choose to keep the width and decrease the temperature for the DLGN, and keep the temperature and increase the width for the CDLGN.

D.3 Runtime Measurements

Our objective is to assess the runtime performance of both parametrizations in a comparable way. To rule out possible discrepancies that are unrelated to the IWP, we build a Python subclass of the original classes for logic layers that can execute both our IWP and the OP. At runtime, a Boolean variable determines which parametrization is chosen. Apart from the different weight initialization and the invocation of the custom autograd function, the footprint of this algorithm on the machine is hence identical. We measure the past nanoseconds for an entire forward and backward pass each, and enforce synchronization at both time points to ensure that the total computation of all streams on the GPU is captured in the time measurements.



(a) Logit initialization variance $\sigma^2 = 1.0$ (b) Logit initialization variance $\sigma^2 = 16.0$

Figure 17: Self-cancellations in the sign-symmetric sum $\sum_{i=1}^{8}(\omega_i - \omega_{\neg i})$ concentrates the gradients around zero (cf. 17a), as long as the initialization variance σ of the logits is not overly high (cf. 17b). Empirical distribution for $N = 10^4$ gradient samples $\frac{\partial g(p,q)}{\partial p}$ with $q = 0.5$.

To quantify uncertainty, we take measurements for 20 different, randomly sampled batches.

E Theoretical Analysis of Parametrization

E.1 Vanishing gradients in OP

Although RIs successfully suppress vanishing gradients, the symmetric parametrization still traps them in a dichotomy between gradient stability and stalling optimization towards other gate functions. On the one hand, Petersen et al. (2024)'s choice of $z = 5$ will sufficiently preserve the gradient norm, as it will decrease by at most $\frac{e^z-1}{e^z+15} \approx 0.9$. On the other hand, already slightly decreasing to $z = 3$ would again elicit vanishing gradients after only a few layers, as $\frac{e^z-1}{e^z+15} < 0.55$.

E.2 Algebraic interpretation of the IWP

To understand the redundancies from an algebraic viewpoint, we can regard the space of binary functions $\mathcal{G}_2 := \{G : \{0,1\}^2 \to \{0,1\}\}$ as a vector space over the field $\mathbb{Z}_2$. Firstly, seven of the eight aforementioned negation symmetries correspond to linear dependencies $0 = G_i + G_{\neg i} + 1$ between elements in $\mathcal{G}_2$. Secondly, the redundancy that led to the suboptimal rounding in the example on the discretization error can be captured in the linear dependency $0 = G_3 + G_6 + G_8 + 1$.

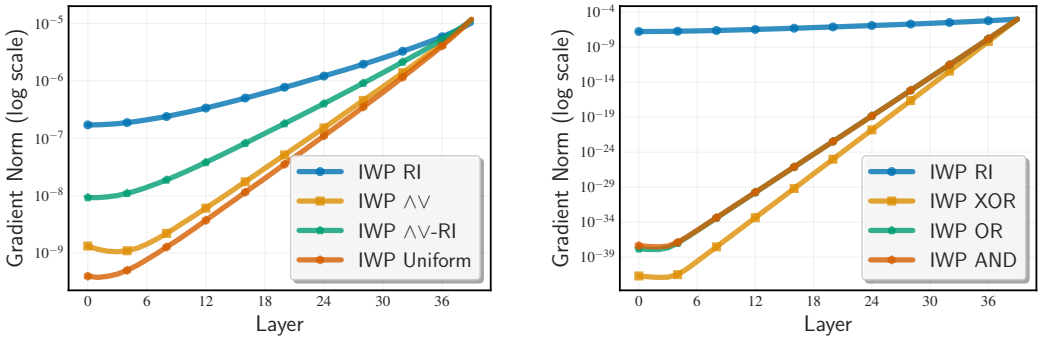
E.3 Minimal rounding error of the IWP

When rounding the gate estimator g_ω to a logic gate g_α, we round each output estimator ω_{ij} to its closest binary number $\alpha_{ij} := \arg\min_{b\in\{0,1\}} |\omega_{ij} - b|$.

This achieves a minimal discretization error $\|g_\omega - g_\alpha$ in terms of any Minkowski norm $\|f - g\|_p := \sqrt[1/p]{\sum_{x\in\{0,1\}^2} |f(x) - g(x)|^p}$, because for any binary input $x = (i, j)$, the term $|g_\omega(x) - g_\alpha(x)| = |\omega_{ij} - \alpha_{ij}| = \min_{b\in\{0,1\}} |\omega_{ij} - b|$ by definition.

E.4 Remaining causes of vanishing gradients in IWP

Even with heavy-tail initializations that concentrate the ω_{ij} close to $0, 1$, destructive interference between gradient signals can still arise for precisely three reasons. Still, all of them are out of the control of the parametrization.



(a) RI vs. multi-gate biases

(b) Single-gate biases

Figure 18: Gradient norm decrease for different heavy-tail initializations of an IWP DLGN with 40 layers. While RIs stand out as the only stable single-gate bias, other multi-gate biases also retain stable gradients.

The first reason is destructive interferences that arise from the probabilistic relaxation of the Boolean functions. For example, for binary inputs $(1, 1)$, the gradient of the OR function $g_8(p, q) = p + q - pq$ will be 0 for both inputs. We obtain a symmetric case with input $(0, 0)$ and the AND function $g_2(p, q) = pq$.

Opposed to that, the remaining two reasons both relate to the parameter initialization of the DLGN architecture. We divide them into cancellations inside a neuron and between neurons.

Inside a neuron, cancellations can arise if the two terms in 12 have different signs. This happens precisely if $\omega_{11} > \omega_{10}$ and $\omega_{01} < \omega_{00}$, or vice versa, which holds only if the relaxation is close to the XOR function $g_7(p, q) = p + q - 2pq$ or its negated counterpart NXOR. Similar to the first reason, this behaviour is not problematic and even intended as long as the inputs carry information about the desired output. If the gate outputs ω_{ij} are close to $0, 1$, the gradient norm will remain close to 1. But in the case of low information, where both inputs $p, q \simeq 0.5$ are highly uncertain, the gradients of the probabilistic surrogate of XOR and NXOR will both collapse to 0 and annihilate the gradient signal. Depending on the logic gate distribution, this undesirable scenario will, however, inevitably occur as we scale logic gate networks in depth (cf. Figure 21). A heavy-tailed initialization of the logic gate distribution alone does not suffice to prevent this. In particular, we will observe later that even RIs suffer from this information collapse. But in theory, this is only problematic if XOR functions are present in the network, which is not the case for RIs.

Finally, even if we can avoid cancellations inside a neuron, gradients from different neurons might still cancel when they pass the same neuron. Because of the negation symmetry in Boolean functions, a parameter initialization that treats each function and its negated counterpart independently will result in sign-symmetric gradients across different neurons during backpropagation. If the gate output of a neuron is used as the input of multiple subsequent neurons, this gate will receive a sum of sign-symmetric partial derivatives. The more gates this neuron is connected to, the more this sum will concentrate at 0.

E.4.1 Heavy-tail, negation-asymmetric initializations

We maintain that an ideal initialization scheme should satisfy three properties to scale logic gate networks in depth: heavy tail, information preservation, and negation asymmetry. The normal initialization $\Omega_{ij} \overset{i.i.d.}{\sim} \mathcal{N}(0, 1)$ violates all of these properties. The resulting coefficients ω_{ij} will concentrate symmetrically around 0.5 and evoke vanishing gradients, as Figure 18a illustrates.

First of all, one could ensure a heavy-tail distribution of the coefficients ω_{ij} at 0 and 1 by shifting the normal distribution in a negative or positive direction. The overall sign

combination in $\Omega_{ij} \overset{i.i.d.}{\sim} \mathcal{N}(\pm\mu_{ij}, 1)$ hence attributes a high initial bias towards one of the sixteen logic gate functions. Choosing the pass-through gate $G_4(A, B) = A$ for all neurons recovers the idea of RIs.

Indeed, if we restrict ourselves to choosing only a single function for all neurons, RIs are the only viable approach. While the constant functions have no gradient anyway, the AND, OR, and XOR functions alone rapidly concentrate the intermediate feature distribution to $1, 0$, and 0.5, as Figure 21 exemplifies. At that point, their gradients collapse to 0 and stifle any information in the input. In terms of our three necessary properties, these initializations fall short of information preservation.

On the other hand, the pass-through gate G_4 does not change the input value p, and maintains a gradient of 1 with respect to that input p, independent of what value p takes. However, as we increase the model depth, the intermediate feature distribution will also collapse to 0.5 with RIs (cf. Figure 3). This is because even small initial uncertainties in the coefficients, i.e. $|\alpha_{ij} - \omega_{ij}| \simeq 0.05$, will accumulate over the layers. But because no gate is initially close to the XOR functions when employing RIs, this high uncertainty in later layers is harmless. On the contrary, we will discuss in the following subsection E.4.2 why this increasing uncertainty can even benefit the optimization of deep logic gate circuits.

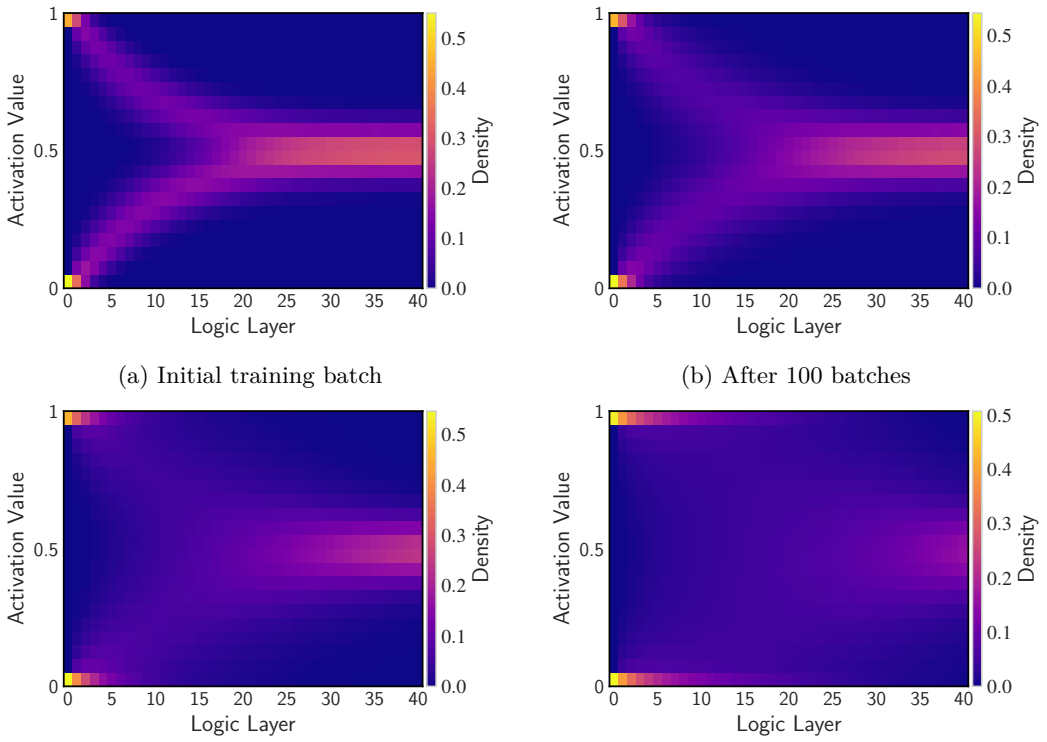
For heavy-tail initializations that bias towards a single function in all neurons, RIs are hence indeed the unique scalable choice. But we might also combine multiple logic gate functions into a heavy-tail initialization. In the extreme case, each logic gate could bias towards one of all sixteen functions with uniform probability 1/16. But this brings us back to the third and last property, namely, negation asymmetry.

Allowing both a Boolean function and its negated counterpart will provoke cancellations if sign-symmetric partial derivatives merge during backpropagation. Fortunately, this condition only holds for architectures with drastically increasing width between layers. For the architecture of Petersen et al. (2022) with uniform width, even negation-symmetric initializations such as the uniform initialization will retain sufficiently stable gradients (cf. Figure 18a).

But this might not hold in general. Formally speaking, any subset of the binary functions $G \subseteq \mathcal{G}_2$ that does not contain a function and its negated counterpart is a feasible negation-asymmetric subset. In particular, such a subset can be obtained by fixing one output to 0 or 1 and taking half of the binary functions that coincide with this mapping. For example, by enforcing $00 \mapsto 0$, we admit the constant 0, the two pass-through gates, three AND functions, one OR function, and the XOR function. Therefore, an alternative to the RI is to combine the AND and OR functions into an AND-OR initialization. Indeed, the complementary concentration behaviour of the AND and OR functions avoids the information collapse at 0.5 that RIs inevitably entail. Instead, Figure 21e depicts how the feature distribution balances at values close to 0 and 1, and hence reduces the uncertainty in the signal in later layers. However, this alone does not render the AND-OR initialization more desirable than RIs. Conversely, while a collapse at 0.5 might be harmful in general, we explain in the next section why it actually benefits the optimization process in the case of RIs.

E.4.2 Residual initializations delay feature learning at later layers

When initializing all neurons with a pass-through gate, Figure 3 displays how the features eventually concentrate at 0.5 at later layers. At those layers, it holds that $1 - p \simeq p \simeq q \simeq 1 - q$, hence the gradient update $\frac{\partial \mathcal{L}}{\partial \omega_{ij}} = \frac{\partial \mathcal{L}}{\partial g_\omega} \frac{\partial g_\omega}{\partial \omega_{ij}}$ is roughly equal for all i, j. Because of that, the neurons in the later layers will maintain their pass-through function until the uncertainty reduces sufficiently. This pass-through enforcement at the later layers allows the network to begin with optimizing the earlier layers first. The more the earlier neurons approach specific gates, the more declines the uncertainty of their outputs, allowing the later layers to refine their functionality. Practically, the model first optimizes a shallow logic gate circuit and increasingly advances this circuit in depth over time. Figure 19 showcases this consecutive gate collapse at earlier layers and uncertainty decrease at later layers over the course of training. This implicit organization of feature learning not only tames the overall discretization error but will also lead to faster convergence. On the contrary, the initial



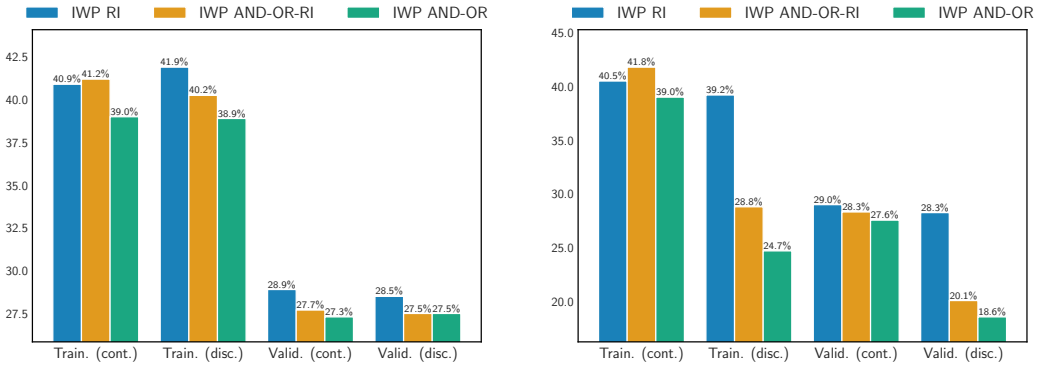
(a) Initial training batch

(b) After 100 batches

(c) After 200 batches

(d) After 500 batches

Figure 19: Distribution of intermediate gate outputs of an IWP DLGN with RIs. Measurements were taken at different timesteps over the course of training, where each batch comprises 100 CIFAR-100 images.



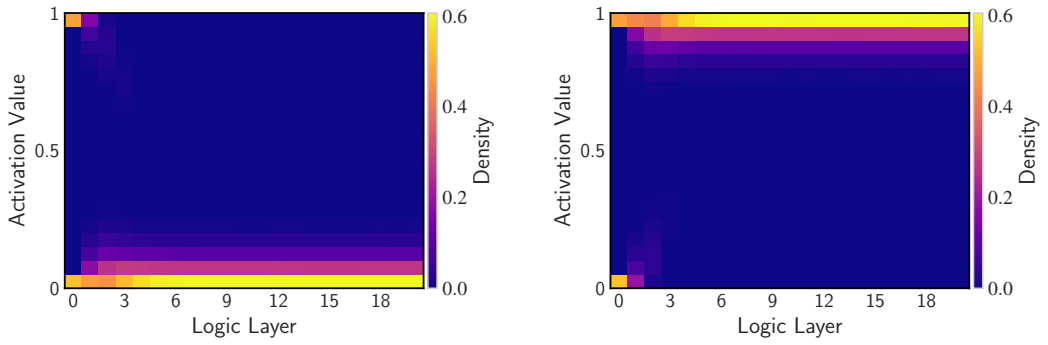
(a) 5-fold depth

(b) 20-fold depth.

Figure 20: In contrast to the structured layer optimization of DLGNs with RIs that steadily maintains a low discretization error, the simultaneous layer optimization for AND-OR initialization drastically increases the discretization error and harms overall performance.

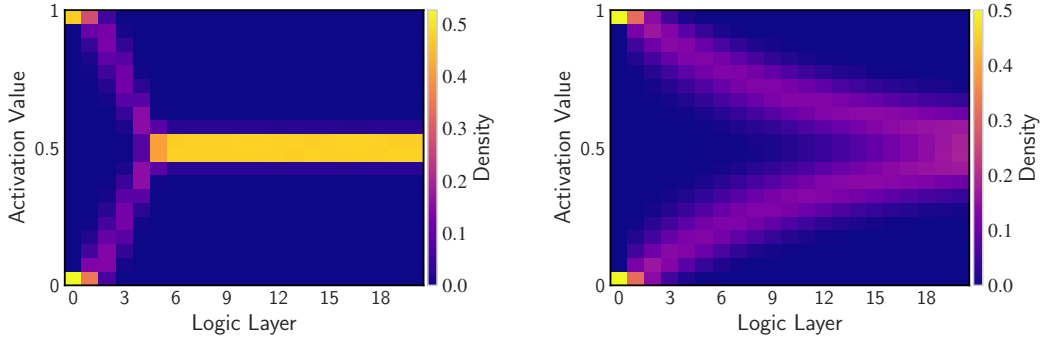
feature distribution of the AND-OR initialization will allow neurons at all layers to update their coefficients in a non-uniform fashion at the same time. The drawbacks of such a more chaotic optimization process become noticeable as we scale those networks in depth.

While the discretized accuracy of both initializations remains similar for shallower logic gate networks with 4 or 20 layers, scaling these networks to 80 layers exposes a clear discretization gap for the AND-OR initialization. At the same time, the RI maintains a low rounding



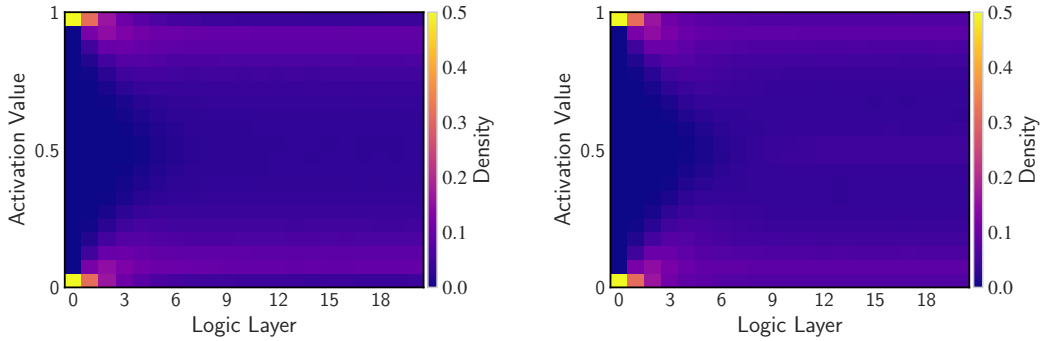
(a) heavy-tail towards AND

(b) heavy-tail towards OR



(c) heavy-tail towards XOR

(d) heavy-tail towards pass-through (RI)



(e) heavy-tail towards AND and OR

(f) heavy-tail towards all 16 gate functions

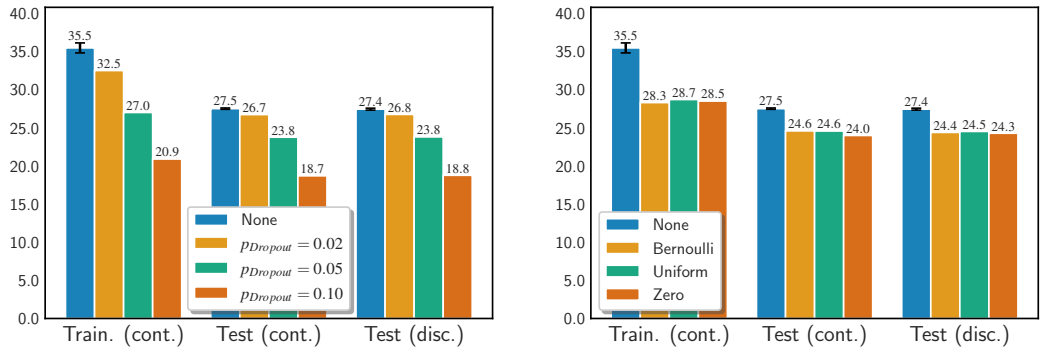
Figure 21: Initial distribution of intermediate gate outputs, averaged over 100 CIFAR-100 images, when initialized with different heavy-tail initializations.

error over the course of training and exhibits slightly superior predictive performance (cf. Figure 20). Similar drawbacks also hold for a uniform initialization or an initialization that combines AND, OR, and pass-through gates (cf. Figure 21).

To conclude, pairing our exact IWP with RIs results in logic gate networks that are scalable in depth and can harness the associated expressive benefits.

F Regularizing Logic Gate Networks

To mitigate the generalization error, we try to impose several constraints on the DLGN architecture that have benefited standard neural network architectures. Unfortunately, the methods that we have tried did not raise the test accuracies further, leaving the generaliza-



(a) Dropout

(b) Random gate interventions

Figure 22: Accuracies of the DLGN with dropout and random gate interventions.

tion gap an open problem. In the following, we present the measures we have taken, how we implemented them for logic gate networks, and how they impacted performance.

F.1 Dropout

When applied in standard feed-forward neural networks, dropout (Srivastava et al., 2014) typically randomly zeroes neurons. For the logic gate network, the zeroing operation is, however, only a neutral operation in the algebraic sense when we apply it in the summation in the GroupSum layer. For logic gates, the zero is not a neutral element, but on equal terms with its binary complement 1. We hence decide to realise dropout by randomly masking logic gate outputs at the final logic layer. To determine which outputs are affected, we randomly select channels of the input tensor and mask the outputs of all gates that are path-connected to inputs from at least one of these channels. For all affected gates, we ensure that they receive no gradient update. Each channel or feature dimension is selected independently with a probability $p_{dropout} > 0$. This selection is repeated for every single batch in training. For $p_{dropout} = 0.02$, roughly 30,000 of the 120,000 logic gates in the final layer are masked. For $p_{dropout} = 0.05$, this number increases to 70,000, and culminates in 100,000 for $p_{dropout} = 0.1$.

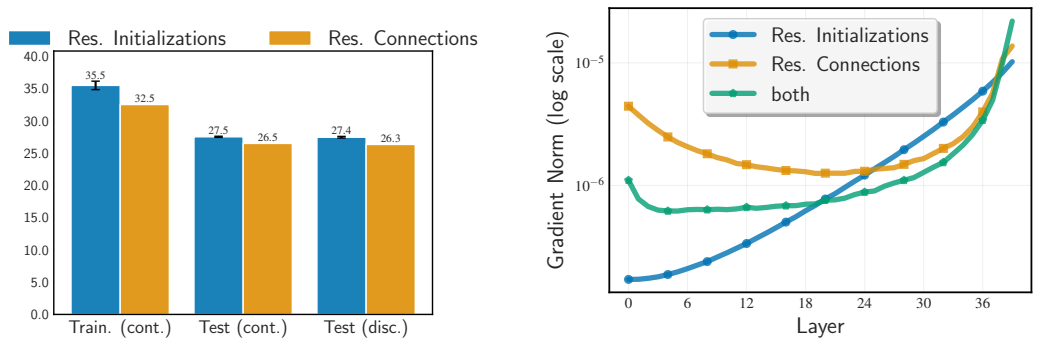
However, Figure 22a shows that the test accuracies degrade with increasing dropout probability. This regularization strategy does not, hence, seem beneficial.

F.2 Randomized gate interventions

Similarly, we try to randomly intervene in the output of each gates in the network with a probability $p_{intervene} > 0$. We explore several strategies to replace the actual gate output: from a simple replacement by a constant value to replacement by a random uniform $b \sim U([0, 1])$ or a symmetric Bernoulli $b \sim B(0.5)$. We explore the impact of magnitude for the intervention probability $p_{intervene}$ and find $p_{intervene} = 0.05$ to yield the best results in the end. Indeed, the generalization gap narrows substantially, but the test accuracies still trail the unregularized DLGN for all intervention strategies (cf. Figure 22b).

F.3 Residual connections

Finally, we explore if the network benefits from enforcing explicit residual connections (He et al., 2016) between layers instead of RIs. From the first to the last layer, a linearly increasing fraction of gates are fixed to directly pass their output to a unique neuron in the subsequent layer. We ensure that incoming residual streams from earlier layers are continued until the last layer. That way, each layer is guaranteed to receive a fraction of unreduced gradient signals, even when the remaining weights are not initialized with a heavy tail, but a standard Gaussian. Unsurprisingly, the gradient norms of DLGNs with residual connections



(a) Accuracy

(b) Gradient stability

Figure 23: Accuracies and gradient norms of the DLGN with residual connections compared to RIs.

are even more stable than for RIs, which still include some uncertainty in the weights ω_{ij} (cf. Figure 23b). However, Figure 23a indicates that both the training and test accuracies suffer slightly from this functional constraint. Although they half the number of learnable parameters and allow to retain gradients norms without heavy-tail initializations, residual connections do not seem to play a beneficial role for generalization.

G Related Work

G.1 Learning single logic gates

Several works have exploited that learning circuits of logic gates with more than two inputs allows to embed more functional expressivity on the same hardware (Umuroglu et al., 2020; Bacellar et al., 2024).

The reason is that a single logic gate with n inputs has a VC dimension of 2^n (Vapnik & Chervonenkis, 1971). On the contrary, a circuit of binary logic gates with n inputs has a strictly smaller discriminative power, as the VC dimension of subcircuits merely accumulates additively and not multiplicatively (Andronic & Constantinides, 2025).

On the contrary, DLGNs were practically limited to learn logic gates with very few inputs, as processing 2^{2^n} parameters per logic gate with n inputs quickly becomes intractable. With our IWP that reduces the number of parameters to 2^n, advancing DLGNs to process more than two inputs per gate becomes a viable option.

In contrast to our IWP, these works do not directly estimate the outputs of the logic gates. Instead, they use a different representation class and quantize this class to logic gates after training. However, these indirect representations either fall short of exploiting the expressivity of logic gates (Umuroglu et al., 2020) or are costlier to parametrize (Andronic & Constantinides, 2023; 2025). To begin with, Bacellar et al. (2024) do not relax the logic gate at all and approximate gradients via a finite difference method that accumulates all 2^n function values in a weighted sum. Most other works relax each logic gate to a continuous function class during training and quantize it back afterwards (Umuroglu et al., 2020; Andronic & Constantinides, 2023; 2025). Our IWP also falls within this category. However, these works differ from our IWP in that these function classes either do not completely exploit the expressivity of logic gates or require more parameters to train. On the one hand, Umuroglu et al. (2020) merely regress an affine transformation $w^T x + b$ that is fed through an activation function after batch normalization. Here, x is the input vector, and w, b are learnable weights and bias. Although the parameter size of each neuron grows only linearly in the number of logic gate inputs, this relaxation can also express only a small subset of Boolean functions. Andronic & Constantinides (2023) hence extends this relaxation to kernelized regression $w^T \phi(x) + b$ with a polynomial kernel ϕ that maps x to all monomials

Table 1: All binary logic functions with real-valued relaxations and gradients

id	G_i	$G_i(0,0)$	$G_i(0,1)$	$G_i(1,0)$	$G_i(1,1)$	g_i	$\frac{\partial g_i}{\partial A}$	$\frac{\partial g_i}{\partial B}$
1	0	0	0	0	0	0	0	0
2	$A \wedge B$	0	0	0	1	AB	B	A
3	$\neg(A \to B)$	0	0	1	0	$A(1-B)$	$1-B$	$-A$
4	A	0	0	1	1	A	1	0
5	$\neg(B \to A)$	0	1	0	0	$B(1-A)$	$-B$	$1-A$
6	B	0	1	0	1	B	0	1
7	$A \oplus B$	0	1	1	0	$A+B-2AB$	$1-2B$	$1-2A$
8	$A \vee B$	0	1	1	1	$A+B-AB$	$1-B$	$1-A$
9	$\neg(A \vee B)$	1	0	0	0	$1-A-B+AB$	$-1+B$	$-1+A$
10	$\neg(A \oplus B)$	1	0	0	1	$1-A-B+2AB$	$-1+2B$	$-1+2A$
11	$\neg B$	1	0	1	0	$1-B$	0	-1
12	$B \to A$	1	0	1	1	$1-B+AB$	B	$-1+A$
13	$\neg A$	1	1	0	0	$1-A$	-1	0
14	$A \to B$	1	1	0	1	$1-A+AB$	$-1+B$	A
15	$\neg(A \wedge B)$	1	1	1	0	$1-AB$	$-B$	$-A$
16	1	1	1	1	1	1	0	0

of degree at most D, where D is a configurable parameter. The size of w hence scales to n^D, where $n = \dim(x)$ is the number of inputs. To completely cover the class of Boolean functions, one needed to scale D to n in order to incorporate the conjunction of all n inputs. The resulting weights would then have dimension n^n, which is larger than our 2^n. Finally, Andronic & Constantinides (2025) learn even larger neural networks within each logic gate relaxation.

G.2 Unrelated advancements

Finally, these works contributed several advancements that do not relate to the parametrization, such as learning and simplifying the connection topology or regularization.

G.2.1 Learning connections

Petersen et al. (2024) maintained that randomly initializing the connections between logic gate functions ab initio and leaving them fixed during training does not degrade performance. Instead, Bacellar et al. (2024) learn these connections via a softmax relaxation. This degree of freedom however comes at the cost of learnable weight matrixes whose dimensions correspond to the widths of contiguous layers.

G.2.2 Regularization

While Andronic et al. (2025) employ pruning strategies that incorporate the connection topology of the hardware, Bacellar et al. (2024) exert regularization on the Fourier transform of each logic gate (O'Donnell, 2014).

G.2.3 Classification head

To convert the logic gate outputs into a classification, DLGNs counts the bits for each class and outputs the class index with the highest sum. To avoid the additional overhead of embedding these operations in FPGA hardware, Bacellar et al. (2024) replace them by learnable lookup tables.