IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 22 September 2023, accepted 23 October 2023, date of publication 30 October 2023, date of current version 7 November 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3328622

RESEARCH ARTICLE

Deep Stochastic Logic Gate Networks

YOUNGSUNG KIM
Department of Artificial Intelligence, Inha University, Incheon 22212, Republic of Korea
e-mail: yskim.ee@gmail.com

This work was supported in part by the Inha University Research Grant, and in part by the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korea Government [Ministry of Science and ICT (MSIT)] [Artificial Intelligence Convergence Innovation Human Resources Development (Inha University)] under Grant RS-2022-00155915.

ABSTRACT This paper introduces a novel regularization approach aimed at improving generalization performance by perturbing deterministic logical expressions. We incorporate logical inference into deep neural networks using logic gates and propose stochastic sampling to select appropriate logic gates from a predetermined set at each node, resembling sampling from a categorical distribution. While the Gumbel softmax relaxation facilitates effective sampling learning, the independence of perturbation from the maximum index operation ($\arg\max$) poses challenges in maintaining consistent sampling and preserving the original categorical probability order. To address this issue, we introduce scaled noise in the Gumbel process, followed by normalization to unnormalized probabilities. By leveraging randomness and introducing stochastic learning into deterministic logical transformations, we demonstrate enhanced classification accuracy. Extensive evaluations on publicly available datasets, including UCI (adult and breast cancer), MNIST, and CIFAR-10, establish the superiority of our method over softmax-based logical gate networks. Our contributions significantly advance the training of logic gate-based networks, inspiring further developments in deep logic gate network training.

INDEX TERMS Logic gates networks, reparameterization, sampling, stochastic process.

I. INTRODUCTION

The integration of logic and prior knowledge into learning algorithms, known as *inductive bias*, plays a crucial role in enhancing the generalization capabilities of machine learning models from limited training data to making accurate predictions on unseen data [1].

Early research in Artificial Intelligence (AI) witnessed the development of logic networks, such as Logic Theorist and General Problem Solver, which laid the foundation for symbolic reasoning and rule-based problem-solving [2], [3], [4]. Additionally, in 1943, McCulloch and Pitts introduced the first mathematical model of a neural network, composed of interconnected binary units, demonstrating its ability to perform logical operations and inspiring the concept of logic gates within neural networks [5]. The perceptron introduced by Rosenblatt in 1957 further advanced artificial neural networks by allowing learning through weight adjustments [6], leading to the development of modern deep learning models.

The associate editor coordinating the review of this manuscript and approving it for publication was Sangsoon Lim.

The integration of logic gate networks with neural networks presents a promising opportunity to create AI systems capable of both pattern recognition and reasoning based on explicit logical rules [7]. Logic gates, represented by binary inputs and outputs, were instrumental in early logic networks, constructing combinational logic circuits that performed complex logical operations. The incorporation of logic gate networks within modern neural networks empowers AI systems to reason based on explicit conditions, making decisions guided by logical rules, thereby offering a hybrid problem-solving approach that combines pattern recognition with logical reasoning. Furthermore, interpreting and extracting logical rules from trained neural networks are crucial aspects of this integration, fostering explainable AI and increasing the transparency and trustworthiness of AI systems.

However, a fundamental challenge in dealing with logical statements and logic gates lies in their discrete representation, requiring efficient techniques to handle categorical distributions. Sampling techniques play a vital role in various machine learning applications, enabling the generation of representative samples from probability distributions [8], [9]. The Gumbel-max trick [10] has emerged as a powerful method for drawing exact samples from unnormalized categorical distributions. By transforming the sampling process into an optimization problem, the Gumbel-max trick provides a sample that corresponds to the optimum of a perturbed energy function. The Gumbel-Softmax (GS) relaxation [11], [12] further extends this technique, allowing for error backpropagation during deep neural network training, enabling efficient gradient-based optimization. This breakthrough enables the end-to-end training of models involving discrete variables, making the Gumbel-max trick applicable in diverse areas, including natural language processing, reinforcement learning, and generative modeling [13].

In this paper, we propose to apply the Gumbel-Softmax relaxation to modeling in a way that selects an appropriate logic gate to build up complex combinational logic circuits following *neural networks (or multilayer perceptron)* and *backpropagation* machanism. We investigate the effectiveness of this approach for deterministic transformations, such as logic-based neural networks [14], which have not been extensively explored in the related literature. Furthermore, to facilitate end-to-end differentiable learning with discrete categorical distributions, we propose a simple but effective approach to naturally incorporate stochastic learning strategies into the logic gate networks (shown in Figure 1). Our proposed method holds promise for enhancing the performance and interpretability of logic gate networks in various AI applications.

The key contributions of this paper are as follows:

- We demonstrate the validity of the stochastic sampling process for generating maximum value-based weights from a deterministic categorical distribution. This process enables the construction of neural networks with differentiable training.
- We propose a `one-hot` encoding parameter to be multiplied with (or selecting) the logic gate during training. This ensures synchronization with the inference phase and promoting better generalization.
- We establish the effectiveness of regularized unnormalized probability through unit-length normalization in the Gumbel process.
- We empirically validate the viability of the stochastic logical inference process for learning logic gate networks in classification tasks.

These contributions are significant for understanding the training process of logic gate-based networks, particularly in finding effective combinations of discretized rules within deep neural networks. We have demonstrated that one effective approach is to employ differentiable categorical sampling techniques. We believe that our work can serve as inspiration for further exploration of training methods in deep logic gate networks.

II. RELATED WORKS

In this section, we review relevant literature related to logic gate networks, stochastic processes, normalization techniques, and their applications in machine learning.

A. LOGIC CIRCUIT AND SAMPLING

Digital logic circuit design involves creating electronic circuits to perform logical operations and process digital information. These circuits consist of interconnected logic gates, such as AND, OR, and NOT gates, that manipulate binary signals to perform desired computations or control functions [15], [16].

Sampling techniques play a vital role in digital logic circuit design, offering several advantages. They enable efficient exploration of the vast design space, allowing designers to evaluate and compare different circuit configurations and identify promising candidates that meet specific requirements [17], [18].

Sampling techniques are particularly useful in optimization tasks, where they help explore the solution space and refine circuit designs. Techniques like genetic algorithms and simulated annealing leverage sampling to converge towards better-performing designs, optimizing various metrics based on specific objectives.

Logic design with neural networks offers an alternative approach to digital logic circuit design, where neural networks can learn logic functions and automatically generate circuit structures. Machine learning algorithms, such as multilayer perceptrons (MLPs), convolutional neural networks (CNNs), or recurrent neural networks (RNNs), have been employed for logic design tasks, including logic synthesis, optimization, and verification [19], [20].

Moreover, combining sampling techniques with machine learning algorithms has been explored in digital logic circuit design. Integrating sampling methods, such as Monte Carlo sampling or Markov Chain Monte Carlo (MCMC), into the training or optimization process of machine learning models allows for improved exploration of the design space and more robust optimization [21].

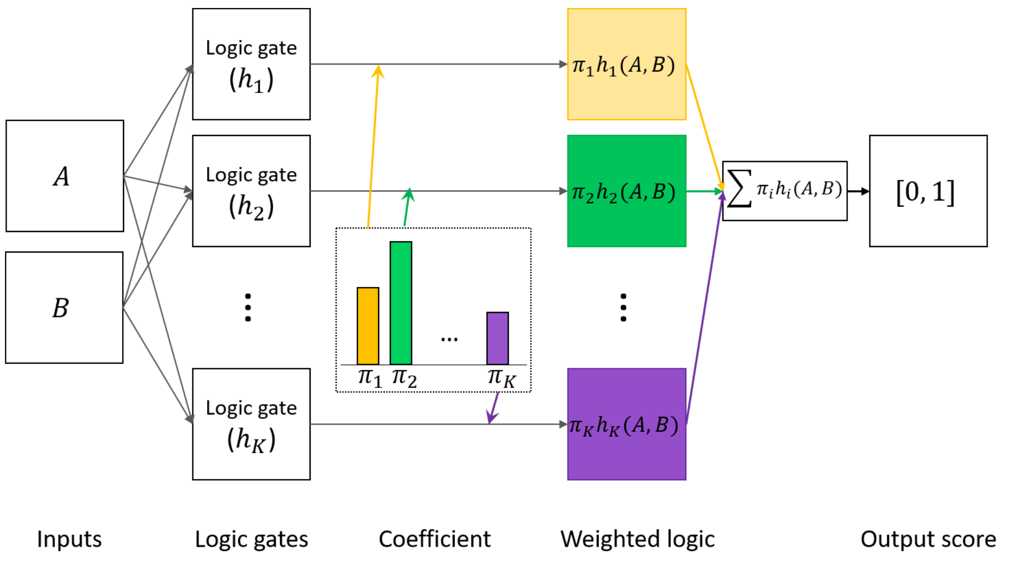
Generative models, like Variational Autoencoders (VAEs) or Generative Adversarial Networks (GANs), combined with sampling techniques, have been used to generate diverse circuit architectures, offering new possibilities for circuit synthesis and optimization [22], [23].

B. EXPLORATION AND EXPLOITATION

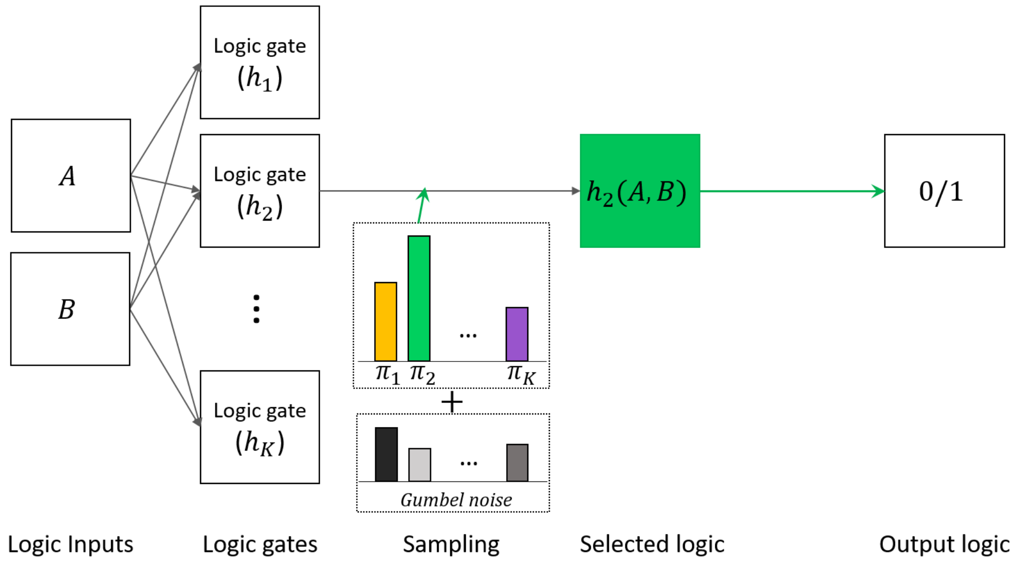
Balancing exploration and exploitation is crucial when sampling from a categorical distribution, especially in applications with exploration-exploitation dilemmas, such as reinforcement learning [24], [25], [26]. Pure exploitation may lead to sub-optimal solutions in the long run, while continuous exploration can also result in sub-optimality and unstable behavior.

Boltzmann exploration addresses this dilemma by gradually reducing the temperature of the Boltzmann distribution during training, shifting from exploration to exploitation [26], [27]. This approach can be combined with inverse transform sampling to draw samples from the categorical distribution.

Alternatively, samples from a tempered categorical distribution can be obtained using the Gumbel-max trick by changing the Boltzmann temperature explicitly or by



(a) Differentiable Logic Gate Networks [14]



(b) *Stochastic* Logic Gate Networks

FIGURE 1. (a) Differentiable Logic Gate Networks [14] and (b) our proposed *stochastic* logic gate networks. Different form (a), our proposed method contains a sampling process and produces a binary output (0 or 1).

scaling the Gumbel noise during independent sampling. The relationship between the Boltzmann temperature and Gumbel noise scaling has been utilized in Boltzmann-Gumbel exploration (BGE), guaranteeing sub-linear regret in stochastic multi-armed bandit problems [28]. This approach has also found applications in recommender systems [29].

C. COMBINATIONAL LOGIC CIRCUIT

Logic gates, fundamental building blocks of digital circuits, have inspired neural networks, where gates can have multiple inputs and a single or multiple outputs. Combinational logic [30] involves combining multiple basic logic gates to perform more intricate logical operations. By incorporating logic gates into neural networks [14], [19], [31], these models can process data and perform logical operations efficiently, similar to traditional digital circuits. Combinational logic circuits lack feedback, allowing real-time computations based on inputs and efficient decision-making.

The versatility of combinational logic circuits extends to neural networks, enabling the integration of simple logic gate circuits into complex structures for handling various tasks, from simple logical operations to complex pattern recognition and deep learning applications.

D. UNIT-LENGTH NORMALIZATION

Unit-length normalization [32] addresses the imbalance in probabilities that may arise due to varying logit magnitudes in classification models. By constraining logits' magnitudes to lie on the surface of a unit sphere with an l_2-norm of 1, this normalization ensures balanced exponentiated logits and evenly distributed softmax probabilities among classes. Unit-length normalization offers invariance to magnitude scaling, acts as a regularization method, and normalizes the effect of linear transformations on input features [33], [34].

The spherical softmax, or von Mises-Fisher softmax [35], preserves both magnitude and directional information of input logits or vectors. It handles large logits more effectively and is robust to outliers, offering advantages over the standard softmax.

E. NOISE AND TIKHONOV REGULARIZATION

The reason training with noise is equivalent to Tikhonov regularization lies in the impact of noise on the learning process and the similarity in their effects on the model [36].

When noise is added to the training data or model parameters, it introduces random perturbations in the learning process. These perturbations prevent the model from focusing too much on specific data points and instead encourage it to generalize better. By experiencing different variations of the data due to noise, the model becomes more robust and less likely to memorize the training data's noise.

Tikhonov regularization, also known as ridge regularization or l_2 regularization, adds a penalty term to the loss function during training. This penalty is proportional to the square of the model's parameter values. The regularization term discourages large parameter values and encourages the model to distribute its learning across many features rather than relying heavily on a few. Like noise, Tikhonov regularization also helps prevent overfitting and improves generalization.

The key insight from the paper [36] is that both noise during training and Tikhonov regularization serve to prevent overfitting and improve model generalization by imposing constraints on the learning process. By introducing noise or adding a regularization term, the models are encouraged to learn more robust and less sensitive representations.

III. PRELIMINARIES

In probability and statistics, the categorical distribution is a fundamental concept used to model discrete random variables with possible outcomes (categories). Each category has an associated probability. Categorical distributions are commonly encountered in various applications, including text generation, classification problems, and reinforcement learning [13].

The Gumbel distribution, on the other hand, is a continuous probability distribution used to model extreme value statistics. It plays a crucial role in the Gumbel-max trick, a powerful technique that allows for efficient and differentiable sampling from a categorical distribution [10]. By leveraging Gumbel-distributed random variables, the Gumbel-max trick provides a way to transform continuous random variables into discrete categorical outcomes. One particular variant of the Gumbel-max trick is the Gumbel-Softmax, which is widely used in machine learning [11], [12]. This is effective for training and generating samples in neural networks with discrete outputs [13].

In this section, we will review the details of categorical distributions, the Gumbel distribution, the Gumbel-max trick, and the Gumbel-Softmax. We will explore how they work together to enable efficient sampling and differentiable training in neural networks dealing with discrete variables.

A. SAMPLING FROM CATEGORICAL DISTRIBUTIONS

The task of selecting one or more categories from a set of K mutually exclusive and exhaustive options is known as sampling from K categories. Each category represents a distinct class or subset of data instances, and the goal is to obtain representative samples reflecting the dataset's characteristics.

Let $\mathcal{C}$ be the set of K distinct categories, and their class index $i \in \mathcal{C} = \{1, 2, \ldots, K\}$. To accomplish this, we introduce Sampling Probability to represent the likelihood of selecting each category i during the sampling process.

In practice, we can customize sampling probabilities (potential function) based on the importance or occurrence frequencies of each category in the data. This involves assigning values to random variables, where the likelihood is determined by potential functions considering interactions among observed variables [28]. These potential functions give rise to a "ragged" posterior probability landscape, but using Markov chain Monte Carlo (MCMC) [37] to sample from the Gibbs distribution in such landscapes can be excessively resource-intensive [38].

The categorical distribution assigns probabilities to K distinct classes. Each probability π_i must be greater than zero, and the sum of all probabilities should equal one ($\sum_{i=1}^{K} \pi_i = 1$).

In machine learning problems, a discrete distribution can be represented using an unconstrained vector $\phi \in \mathbb{R}^K$ of ϕ_i, which consists of arbitrary real values. The optimization process aims to find an optimum ϕ without applying constraints to individual ϕ values. This optimization can be expressed using either the Gibbs or Boltzmann distribution [13], [28]. Consequently, the representation of the categorical distribution, denoted by normalized probabilities $\boldsymbol{\pi}$, involves two parameterizations: unnormalized probabilities θ and unnormalized log-probabilities (logits) ϕ. To introduce temperature scaling, we adopt the parameter $\tau \in \mathbb{R}> 0$, which corresponds to ϕ/τ and is equivalent to $\log \boldsymbol{\theta}$ [13]. The unnormalized logits $\phi_{i;\tau}$ correspond to the i-th class and depend on the value of τ. Hence, the probabilities $\pi_{i;\tau}$ for each class $i \in \mathcal{C} = \{1, \ldots, K\}$ are obtained using a softmax function with the temperature parameter τ:

$$\pi_{i;\tau} = \frac{\exp(\phi_{i;\tau})}{\sum_{k \in \mathcal{C}} \exp(\phi_{k;\tau})} \tag{1}$$

where $\phi_{i;\tau} = \phi_i/\tau = \log \theta_{i;\tau}$.

The temperature τ in the categorical distribution controls the entropy of the distribution, influencing how the probabilities are spread across the categories: a lower temperature leads to more certainty, while a higher temperature results in more randomness [28]. In many cases, a common practice is to set τ to one, which simplifies the representation of the categorical distribution using unnormalized log-probabilities. Equation 1 describes the softmax function with the temperature parameter τ, frequently denoted as $\mathrm{Cat}(\phi, \tau)$ where ϕ is an unnormalized log-probability.

A random variable following the categorical distribution is denoted as I, and a realization (sample) represented by ω signifies the index of the sampled class. In certain cases, the sample can be represented as a one-hot embedding, a unit vector of length C, with a one at index ω and zeros elsewhere. Various algorithms exist for efficiently sampling from the categorical distribution, making it a widely used tool in many applications, including machine learning and statistical modeling [13].

B. GUMBEL DISTRIBUTION

An alternative method for formulating categorical probability without explicitly using softmax transformation or Boltzmann distribution is the Gumbel distribution and the Gumbel-max trick. The Gumbel distribution introduces a random function to the potential function, enabling the computation of the MAP prediction that maximizes the sampling probability or potential function [38]. This approach provides an efficient way to draw samples from the categorical distribution using the Gumbel-max trick (see Section III-C).

The Gumbel distribution [39] is an instance (type I) of the generalized extreme value distribution [9], which models optima and rare events. A Gumbel random variable - which is often referred to in this work as 'a Gumbel' - is parameterized by location and scale parameters $\mu \in \mathbb{R}$ and $\beta \in \mathbb{R}_{\geq 0}$, respectively as $Gumbel(\mu, \beta)$ (or $G(\mu, \beta)$ in short), and a random variable following this distribution with $G_{\mu,\beta}$. Both parameters are frequently omitted when standard settings are assumed (i.e. $G := G_{0,1}$). $G(i)$ is denoted as an i-th standard Gumbel and Gumbel variables are considered a set of identically and independently distributed (i.i.d.). Note that, μ and β are not the mean and variance of a Gumbel. Those mean and variance are $\mathbb{E}[G_{\mu,\beta}] = \mu + \gamma\beta$, $\mathbb{E}[(G_{\mu,\beta} - \mathbb{E}[G_{\mu,\beta}])^2] = \frac{\pi^2}{6}\beta^2$ respectively, where where $\gamma \approx 0.577$ is the Euler-Mascheroni constant and $\pi \approx 3.14$ is the constant pi.

1) GUMBEL NOISE SAMPLING

Gumbel noise is generated using the inverse cumulative distribution function (CDF) of the Gumbel distribution. If U is a random variable from a uniform distribution between 0 and 1, then $G = -\log(-\log(U))$. The cumulative distribution function (CDF)[1] is given by:

$$CDF(x) = \exp(-\exp(-x)) \tag{2}$$

[1]The probability density function (PDF) of the Gumbel distribution is given by: $f(x; \mu, \beta) = (1/\beta) \cdot \exp((\mu - x)/\beta) \cdot \exp(-\exp((\mu - x)/\beta))$ where x is the random variable (Gumbel-distributed variable), μ is the location parameter (a real number), β is the scale parameter (a positive real number).

To generate random variables from a Gumbel distribution with location parameter μ and scale parameter β, we can follow these steps:

1) Generate a random variable U from a standard uniform distribution between 0 and 1.
2) Compute the Gumbel noise (G) using the expression $G = -\log(-\log(U))$. The corresponding code are shown in Listing 1 of Appendix.
3) Scale and shift the Gumbel noise to obtain the desired Gumbel-distributed random variable with location parameter μ and scale parameter β. The transformation is given by $G_{\mu,\beta} = \mu + \beta \cdot G$.

Hence, the cumulative distribution function (CDF) of the Gumbel distribution with location parameter μ and scale parameter β is given by integrating the PDF: $CDF(x) = \exp(-\exp(-(x - \mu)/\beta))$.

C. GUMBEL-MAX TRICK

The Gumbel-max trick [39] is a method to sample from a categorical random variable $I \sim \mathrm{Cat}(\boldsymbol{\pi})$. It involves adding independent and identically distributed (i.i.d.) Gumbel noise samples to the unnormalized log-probabilities and selecting the index with the maximum value, which follows a Gumbel distribution:

$$\arg\max_{i \in \mathcal{C}}\{\phi_i + G^{(i)}\} \sim \mathrm{Cat}(\boldsymbol{\pi}) \tag{3}$$

The Gumbel-max trick can handle normalized probabilities $\boldsymbol{\pi}$ by exploiting the translation-invariance of the argmax function.[2] In simpler terms, adding standard Gumbel noise to the unnormalized value ϕ and then taking the maximum value is equivalent to sampling according to the Boltzmann distribution in Eq. 1.

In machine learning, the Gumbel-max trick is typically used with unnormalized probabilities to enable unconstrained optimization of ϕ (whereas $\boldsymbol{\pi}$ is a probability vector).

While the Gumbel-max trick traditionally employs standard Gumbel noise samples, it can also work with i.i.d. samples from $Gumbel(\mu, \beta)$, with $\mu \neq 0$ and $\beta \neq 1 (\geq 0)$. The resulting distributions of the sampled index and maximum are given by:

$$\arg\max_{i \in \mathcal{C}}\{\phi_{i;\tau} + G^{(i)}_{\mu,\beta}\} \sim \mathrm{Cat}\left(\frac{\exp(\boldsymbol{\phi}/(\tau\beta))}{\sum_{i \in \mathcal{C}} \exp(\phi_i/(\tau\beta))}\right) \tag{4}$$

and

$$\max_{i \in \mathcal{C}}\{\phi_{i;\tau} + G^{(i)}_{\mu,\beta}\} \sim \mathrm{Gumbel}(\mu + \beta \log Z', \beta) \tag{5}$$

Here, $G^{(i)}_{\mu,\beta}$ represents the i-th independent Gumbel random variable with location μ and scale β, and $Z' = \sum_{i \in \mathcal{C}} \phi_i/(\tau\beta)$.

[2]Scaling (or normalizing) unnormalized probabilities ϕ by any positive constant Z, results in a subtraction in logarithmic space: $\log(\phi/Z) = \log\phi - \log Z$, thus Eq. 3 becomes $\arg\max_{i \in \mathcal{C}}\{\phi_i + G^{(i)}\}$. This property is derived from the fact that Gumbel random variables adhere to Luce's choice axiom [40], making the Gumbel-max trick applicable even with unnormalized probabilities θ_i for a subset $B \subseteq \mathcal{C}$: $\arg\max_{i \in B}\{\phi_i + G^{(i)}\} \sim \mathrm{Cat}\left(\frac{1_{i \in B}\pi}{\sum_{i \in B}\pi_i}\right)$.

D. GUMBEL SOFTMAX

The Gumbel Softmax (GS) [12], also known as the Concrete distribution [11], provides a way to obtain soft samples from a categorical distribution, which is useful for gradient estimation. It introduces soft samples S_λ that spread the probability mass over multiple bins, in contrast to hard/discrete samples represented by one-hot embeddings.

The GS distribution $GS(\boldsymbol{\pi}, \lambda)$ is an exact relaxation of one-hot embeddings, and the soft sample $S_\lambda \in \mathbb{R}^{\mathcal{C}}_{\geq 0}$, with $|S_\lambda| = 1$, is defined as [13]:

$$S_{i;\lambda} = \frac{\exp((\phi_i + G^{(i)})/\lambda)}{\sum_{k\in C}\exp((\phi_k + G^{(k)})/\lambda)} \tag{6}$$

where the temperature parameter λ influences the entropy of the Gibbs distribution and, consequently, the characteristics of the generated samples. The value of λ is a critical hyperparameter in GS, as it affects the trade-off between bias and variance in gradient estimation. A high λ induces high bias but low variance, while a low λ reduces bias but increases variance, possibly leading to vanishing gradients as $\lambda \rightarrow 0+$. Setting λ has been a topic of investigation, with different experiments using constant values or annealing schemes [11], [12].

The Gumbel Softmax is widely used for gradient-based optimization in differentiable neural networks. Additionally, it has applications in solving combinatorial problems and addressing inverse problems with discrete multivariate random variables [41], [42].

E. DIFFICULTY WITH UNNORMALIZED LOGITS

Sampling from discrete distributions with unnormalized probability mass functions can be challenging. Additionally, incorporating stochasticity in deep learning models presents difficulties in computing gradients for error backpropagation, especially through stochastic nodes.

Recently, the Gumbel-max trick [39] has gained attention in deep learning due to the Gumbel-Softmax gradient estimator [11], [12], which relaxes the original trick. The Gumbel-max trick can be viewed as a type of perturb-and-MAP method that transforms sampling into an optimization problem. In this approach, the sample corresponds to the optimum of a perturbed energy function, where energies are represented by unnormalized log-probabilities. The optimization task involves finding the maximum of the perturbed log-probabilities, which is straightforward in unstructured settings with a limited number of classes. However, it becomes challenging in structured models with multiple dependent variables and a large sampling domain.

There are a few disadvantages of working with unnormalized logits or log probabilities:

1) NUMERICAL INSTABILITY

Unnormalized logits are raw, unconstrained values, which can lead to numerical instability when performing computations in the Gumbel-Softmax relaxation. Since the Gumbel noise ($G^{(i)}$) is added to the logits before applying the softmax function with the temperature parameter (λ), extreme values in the logits can cause exponential overflow or underflow during the exponentiation step. These numerical issues can result in inaccurate or `NaN` (Not-a-Number) values, leading to unstable and unreliable training.

2) LACK OF INTERPRETABILITY

Working with unnormalized logits in Gumbel-Softmax lacks direct interpretability in terms of probabilities. The logits represent the raw model outputs before any normalization is applied. Without the normalization step, it becomes challenging to understand the relative likelihoods or probabilities assigned to different classes or outcomes. Interpreting and reasoning about the model's decision-making process become more complex without the mapping to meaningful probabilities.

3) DIFFICULTY IN CONVERGENCE

During model training, the optimization algorithms, such as gradient-based methods, rely on the gradients to update model parameters effectively. However, unnormalized logits may lead to inconsistent or unreliable gradients due to the lack of normalization. This can make it difficult for the model to converge to an optimal solution during training. The absence of convergence hinders the learning process and adversely affects the model's performance [13].

IV. PROPOSED METHOD

Perturbed unnormalized probability-based maximum argument estimation has been extensively studied in prior research [43], [44] and has been widely adopted in machine learning due to its effectiveness from various perspectives. This approach involves introducing perturbations during model training to promote exploration and introduce stochasticity which is particularly valuable for tasks that require exploration or sampling-based approaches. By accounting for the inherent uncertainty in complex classification tasks, perturbation-based training prevents the model from overly focusing on specific data points and encourages better generalization.

In contrast to logical learning with softmax, which primarily involves deterministic transformations, we propose a novel stochastic approach to enhance generalization performance in logic gate networks. Specifically, we introduce a learning scaling mechanism to adjust the Gumbel noise term. The effectiveness of this approach in the context of logic gate learning has not been extensively explored in the literature.

Before incorporating noise, we address the issue of empirically inefficient unnormalized probability. To ensure numerical consistency in computation, we propose unit-length normalization, which scales the logits to lie on the surface of a unit sphere with a constant l_2-norm.

A. FORMULATION

Formally, a multilayer perceptron (or fully connected layers) is a function that transforms an n-dimensional real input to

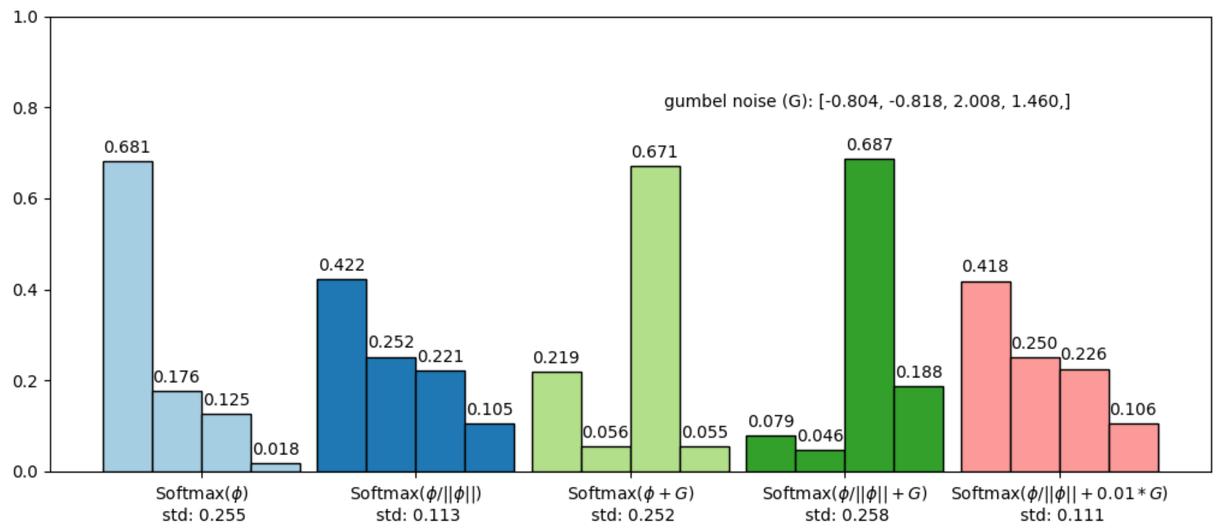


FIGURE 2. **Toy example of distribution of softmax of logits and softmax of pertubed logits. Unnormalized logits (ϕ), Unit-length normalized logits ($\phi/\|\phi\|$), Perturbed logits ($\phi + G$). The last one, our proposed scaled perturbed logit can perserve the original order of logits.**

a one-dimensional real output using n-dimensional weight parameters. The transformation is achieved through a linear combination of the input features $x_0, x_1, \ldots, x_n$ with their corresponding weights $w_0, w_1, \ldots, w_n$, represented as follows:

$$y = w_0 x_0 + w_1 x_1 + \ldots + w_n x_n \tag{7}$$

1) DIFFERENTIABLE LOGIC GATE NETWORKS

In the context of logic gate networks [14], the inputs consist of a two-element set, and the network produces sixteen outputs based on predefined Boolean functions applied to these inputs. The inputs are two-dimensional and are selected from a set of $C(n, k)$-dimensional inputs.

To calculate the sixteen outputs, the two-dimensional inputs undergo 16 ($=2^{2^k}$) different logic operations, representing all possible Boolean functions for the given two-dimensional input. Each of these results is then multiplied with a corresponding weight from a set of 16 weights.

Mathematically, the computation can be expressed as follows:

$$\begin{aligned} y &= w_1 h_1(x_1, x_2) + w_2 h_2(x_1, x_2) + \ldots + w_{16} h_{16}(x_1, x_2) \\ &= w_1 \cdot 0 + w_2 \cdot (x_1 \wedge x_2) + \ldots + w_{16} \cdot 1 \end{aligned} \tag{8}$$

where The functions $h_i(x_1, x_2)$, for i from 1 to 16, represent the different logic operations applied to the inputs, with $h_1(x_1, x_2)$ being the logical constant 0 and $h_{16}(x_1, x_2)$ being the logical constant 1. The specific Boolean functions are listed in Table 1.

To make binary logic networks differentiable probabilistic form, weights can be activated with the softmax function [14]. Using this probabilistic weight and each binary logics are computed as a weigthed summation. $\pi_i = \frac{\exp(w_i)}{\sum_{k=1}^{16} \exp w_k} \in [0, 1]$. Hence,

$$y = \sum_{k=1}^{16} \pi_i \cdot h_i(x_1, x_2) \tag{9}$$

where π_k represents probabilities associated with each $h_i(x_1, x_2)$.

However, there is a critical discrepancy between training and inference in [14]. In training phase, each weight is used with soft sample with softmax, but in the inference phase only one weight is activated using `one-hot` operation for efficient sparse operation to reduce inference time.

B. SAMPLING BASED LOGIC GATES NETWORKS

In this paper, both training and inference of logic networks, we use `one-hot` code for probability term to select the best binary logic gate to formulate the logic gates aggregation.

We propose use binary value for input and output through all logic gate networks. Denote $b \in \{0, 1\}$ binary variable. Thus, the logic gate aggregation with binary input-output values can be written as follows:

$$b = \sum_{i \in \mathcal{C}} \phi_i \cdot h_i(b_1, b_2) \tag{10}$$

where b is the resulting binary value (either 0 or 1). ϕ_i are unnormalized and the parameter to select logic gate and a categorical variable with categorical distribution $\mathrm{Cat}(\pi_1, \ldots, \pi_n), \mathcal{C} = \{1, 2, \ldots, K\}$, i is the index that iterates over all the values of ϕ_i and $h_i(b_1, b_2)$. ϕ_i represents some coefficients or probabilities associated with each $h_i(b_1, b_2)$. $h_i(b_1, b_2)$ is a function that takes b_1 and b_2 as input and returns a value.

By introducing sampling coefficients, Eq. 10 can be re-written as:

$$f(b_1, b_2) = \sum_{i=1}^{16} d_i \cdot h_i(b_1, b_2) \tag{11}$$

d_i is the one-hot coding for a specific index i, which is 1 and 0 for all other indices.

Consequently, this operation defined in Eq. 11 can be recursively applied through layers using the logic gate and selection operation $f(\cdot,\cdot)$:

$$f(f(b_1,b_2),f(b_3,b_4))=\sum_{i=1}^{16} d_i\cdot h_i(f(b_1,b_2),f(b_3,b_4)) \quad (12)$$

where b_1, b_2, b_3, and b_4 are binary inputs.

C. LOGIC NETWORKS WITH GUMBEL PROCESS

As described in the preliminary section, the Gumbel softmax trick provides an effective method for sampling from a categorical distribution. In this context, we utilize the Gumbel softmax trick to obtain optimal coefficients, which are used to select a logic gate.

By applying Eq. 4 to Eq. 6, Gumbel softmax with location μ, scale β, and temperatures $\{\lambda, \tau\}$ can be written:

$$S_{i;\{\mu,\beta,\tau,\lambda\}} = \frac{\exp((\phi_{i;\tau} + G^{(i)}_{\mu,\beta})/\lambda)}{\sum_{k\in C}\exp((\phi_{k;\tau} + G^{(k)}_{\mu,\beta})/\lambda)} \quad (13)$$

where $G_{\mu,\beta} = \mu + \beta\cdot G_{0,1}$, $\phi_{i;\tau} = \phi_i/\tau$, $\lambda > 0$ and $\tau > 0$.

$\lambda > 0$ is ensuring smoothness in learning and introduces a tradeoff: low temperature values lead to samples close to one-hot but with higher gradient variance, while high temperature values produce smoother samples but with smaller gradient variance [12]. For simplicity, we set the temperature value (λ) to 1 in this study. Hence,

$$S_{i;\{0,\beta,\tau\}} = \frac{\exp(\phi_{i;\tau} + G^{(i)}_{0,\beta})}{\sum_{k\in C}\exp(\phi_{k;\tau} + G^{(k)}_{0,\beta})} \quad (14)$$

where $G_{0,\beta} = \beta\cdot G_{0,1}$, $\phi_{i;\tau} = \phi_{i;\tau}$.

The individual unnormalized logits are modified by Gumbel noise. So, it is crucial to handle perturbed logits with great care. In practical applications, if the ranges of the two distributions are not properly aligned, unintended maximum indices may be returned. This issue arises from the fact that the argmax of perturbed log-probabilities follows a categorical distribution, while the maximum itself follows an independent Gumbel distribution, as discussed in the work [45]. As shown in Figure 2, a distribution of the perturbed logic is quite different from the original logit which make different maximum index estimation.

D. REGULARIZATION VIA NORMALIZATION

We address some disadvantages of unnormalized logits, which include numerical instability and lack of interpretability. Numerical instability arises from extreme values in the logits, leading to exponential overflow or underflow during the exponentiation step. To mitigate this, we propose unit-length normalization, which maps the logits onto the surface of a unit sphere, preserving their directional properties and preventing domination by extreme values.

Unit length logits are assigned to directions on a unit sphere, with the constraint that the associated vectors have a magnitude of 1. This normalization allows for meaningful interpretation of directionality and facilitates comparison of logits in terms of their directions rather than magnitudes. By defining multiple sphere spaces, each logical aggregation indicates a different direction of logical boolean function combination, leading to enhanced interpretability.

To introduce exploration and prevent overfitting of deterministic logic functions, we incorporate perturbation into the logits using stochastic Gumbel noise. By adding this noise on the surface of the unit sphere logits, we encourage exploration and obtain smoother distributions, as shown in Figure 2.

E. LEARNABLE SCALING VALUE FOR GUMBEL PERTURBATION

The randomness introduced by Gumbel noise is independent for each category, enabling the "argmax" operation to consistently select the category with the highest perturbed log-probability, regardless of other categories' noise.

However, Gumbel perturbation does not maintain the order of the original log probability distribution. When independent Gumbel noise is added to the unnormalized log probabilities, the resulting samples are perturbed versions of the original probabilities. Consequently, the sampled category index may differ from the original log probability distribution.

To achieve a more accurate approximation of the true categorical distribution, we repeatedly apply the Gumbel-max trick and collect a large number of samples. As the number of samples increases, the impact of Gumbel noise diminishes, leading to a sampling process that better aligns with the probabilities of the original unnormalized log probabilities. It is important to note that this repeated sampling process is not commonly applied to individual logic gate aggregations, as their sampling frequency is relatively low.

In this paper, we propose a simple yet effective method to reduce the scale of Gumbel noise by introducing a learnable parameter, β. To maintain parameter efficiency, we utilize the same β value for each individual hidden layer in deep neural networks. The modified Gumbel-Softmax operation, incorporating the learnable scaling parameter, β, is given by:

$$S_{i;\beta} = \frac{\exp(\phi_i/\|\boldsymbol{\phi}\| + \beta\cdot G^{(i)})}{\sum_{k\in C}\exp(\phi_k\|\boldsymbol{\phi}\| + \beta\cdot G^{(k)})} \quad (15)$$

where $\|\boldsymbol{\phi}\|$ replaces τ in Eq. 14, and $G = -\log(-\log(U))$, with U being a uniform random variable. By employing `one-hot` encoding during training and inference, we ensure that only one logic gate corresponding to the maximum value is activated at each node.

F. OUTPUT AGGREGATION AND RANKING

Following [14], we employ a straightforward aggregation and ranking layer to obtain the final class label mapping. The outputs are computed by aggregating the final scores as follows:

$$\hat{y}_i = \frac{1}{T}\sum_{j=i\cdot n/k+1}^{(i+1)\cdot n/k} b_j, \quad (16)$$

where b_j represents binary output values calculated from the last logic gate layer, and T is a normalization term.

TABLE 1. List of logic operations using two binary input values [14], [47].

$h_i : (A, B) \mapsto 0/1$					Operator	arithmetic	name
$h_i \setminus (A, B)$	$(0, 0)$	$(0, 1)$	$(1, 0)$	$(1, 1)$			
h_1	0	0	0	0	0	0	FALSE
h_2	0	0	0	1	$A \wedge B$	AB	AND
h_3	0	0	1	0	$\neg(A \Rightarrow B)$	$A - AB$	A AND NOT B
h_4	0	0	1	1	A	A	A
h_5	0	1	0	0	$\neg(A \Leftarrow B)$	$B - AB$	NOT A AND B
h_6	0	1	0	1	B	B	B
h_7	0	1	1	0	$A \oplus B$	$A + B - 2AB$	XOR
h_8	0	1	1	1	$A \vee B$	$A + B - AB$	OR
h_9	1	0	0	0	$\neg(A \vee B)$	$1 - (A + B - AB)$	NOR
h_{10}	1	0	0	1	$\neg(A \oplus B)$	$1 - (A + B - 2AB)$	XNOR
h_{11}	1	0	1	0	$\neg B$	$1 - B$	A OR NOT B
h_{12}	1	0	1	1	$A \Leftarrow B$	$1 - B + AB$	NOT B
h_{13}	1	1	0	0	$\neg A$	$1 - A$	NOT A
h_{14}	1	1	0	1	$A \Rightarrow B$	$1 - A + AB$	NOT A OR B
h_{15}	1	1	1	0	$\neg(A \wedge B)$	$1 - AB$	NAND
h_{16}	1	1	1	1	1	1	TRUE

TABLE 2. Experimental results on the UCI adult and breast cancer data sets. Averaged results over 10 runs are reported [14].

Adult	Acc.	#Param.	Infer. Time	Space
Decision Tree Learner	79.50%	$\approx$ 50	86ns	$\approx$ 130B
Logistic Regression	84.80%	234	63ns	936B
Neural Network	84.90%	3810	635ns	15KB
Diff Logic Net [14]	84.87%	1280	5.1ns	640B
Ours	84.99%	1280	5.1ns	640B
Breast Cancer	Acc.	#Param.	Infer. Time	Space
Decision Tree Learner	71.90%	$\approx$ 100	82ns	$\approx$ 230B
Logistic Regression	72.90%	104	34ns	416B
Neural Network	75.30%	434	130ns	1.4KB
Diff Logic Net [14]	75.57%	640	2.8ns	320B
Ours	76.00%	640	2.8ns	320B

G. STRAIGHT-THROUGH GUMBEL-SOFTMAX ESTIMATOR

In the context of the categorical distribution of ϕ, we adopt continuous relaxations of one-hot vectors, following the approach proposed in [12]. During the forward pass, we use $\arg\max$ to discretize $d(\phi)$ for sampling purposes. In the backward pass, we apply a continuous approximation by equating the gradient of $d(\phi)$ with that of $S(\phi)$, introducing the Straight-Through (ST) Gumbel Estimator, inspired by the biased path derivative estimator [46].

V. DISCUSSION

A. UNIT-LENGTH NORMALIZATION, PERTUBATION, AND TIKHONOV REGULARIZATION

Unit-length normalization is applied to make data comparable across different features and prevent certain features from dominating the learning process due to their larger scale. It scales feature vectors to have the same magnitude, equalizing feature importance during model training. Additionally, perturbation is introduced by applying a scale factor to the normalized logits, which introduces randomness and encourages exploration during training. The training process with noise is equivalent to Tikhonov regularization, as they share similarities in their effects on the learning process.

Tikhonov regularization adds a penalty term based on the square of model parameters to encourage the model to distribute learning across many features and prevent overfitting. This regularization imposes constraints on the model's parameter values, discouraging large parameter values and promoting a more balanced learning process.

Both techniques, unit-length normalization and perturbation, contribute to improved generalization by reducing feature dominance and introducing randomness to the learning process. The model becomes less sensitive to individual feature magnitudes and is more robust to noise and minor input variations. The reduction in variance among logit components, combined with the regularization effect, stabilizes the model's predictions and leads to more consistent and balanced confidences for different classes, ultimately enhancing the model's generalization performance. The impact of noise on the learning process and its similarity in effects with Tikhonov regularization further supports the effectiveness of perturbation-based training in achieving better generalization in deep learning frameworks.

VI. EXPERIMENTS

A. EXPERIMENTAL SETUP

1) DATASETS

We conducted evaluations using several datasets for different tasks: Adult Census [48] and Breast Cancer [49] datasets for non-visual tasks (using categorical feature), and MNIST [50] and CIFAR-10 [51] datasets for visual recognition tasks. Further details about these datasets can be found in Table 5 in the Appendix.

2) COMPARED METHODS

We compared our proposed method against various network architectures serving as baselines, including:

TABLE 3. Experimental results using MNIST [50]. Averaged results over 10 runs are reported [14].

MNIST	Acc.	#Param.	Space	T. [CPU]	T. [GPU]	**OPs**	FLOPs
Linear Regression	91.60%	4 010	16KB	$3\mu s$	2.4ns	(4M)	4K
Neural Network (*small*)	97.92%	118 282	462KB	$14\mu s$	12.4ns	(236M)	236K
Neural Network	98.40%	22 609 930	86MB	2.2ms	819ns	(45G)	45M
Diff Logic Net [14]	98.47%	384 000	188KB	$7\mu s$	(50ns)	384K	|
Ours	98.61%	384 000	188KB	$7\mu s$	(50ns)	384K	|
Binary Neural Networks					T. [FPGA]		
FINN [52]	98.40%			$(96\mu s)$	641ns	5.28M	
BinaryEye [53]	98.40%				$50\mu s$		
ReBNet [54]	98.29%				$3\mu s$		
LowBitNN [55]	99.20%				$152\mu s$		
Sparse Neural Networks					Sparsity		
Var. Dropout [56]	98.08%	4 000			98.50%	(8M)	8K
L_0 regularization [57]	98.60%				2/3	(200M)	200K
SET-MLP [58]	98.74%	89 797			96.80%	(180M)	180K
Sparse Function Net [62]	94.20%	$3 \times 1\,849$					> 2K

TABLE 4. Experimental results on CIFAR-10 [51]. Averaged results over 10 runs are reported [14].

CIFAR-10	Acc.	#Param	Space	T. [CPU]	T. [GPU]	**OPs**	FLOPs
Neural Network (color-ch. res. = 4)	50.79%	12.6M	48MB	1.2ms	370ns	(25G)	25M
Diff Logic Net (*small*) [14]	45.62%	48K	24KB	$1.3\mu s$	19ns	48K	|
Diff Logic Net (*medium*) [14]	57.37%	512K	250KB	$7.3\mu s$	29ns	512K	|
Diff Logic Net (*large*) [14]	60.49%	1.28M	625KB	$(18\mu s)$	(73ns)	1.28M	|
Diff Logic Net (*large*×2) [14]	61.60%	2.56M	1.22MB	$(37\mu s)$	(145ns)	2.56M	|
Diff Logic Net (*large*×4) [14]	61.51%	5.12M	2.44MB	$(73\mu s)$	(290ns)	5.12M	|
Ours (*small*)	47.11%	48K	24KB	$1.3\mu s$	19ns	48K	|
Ours (*medium*)	57.85%	512K	250KB	$7.3\mu s$	29ns	512K	|
Ours (*large*)	61.21%	1.28M	625KB	$(18\mu s)$	(73ns)	1.28M	|
Ours (*large*×2)	62.11%	2.56M	1.22MB	$(37\mu s)$	(145ns)	2.56M	|
Ours (*large*×4)	61.85%	5.12M	2.44MB	$(73\mu s)$	(290ns)	5.12M	|
Best Fully-Connected Baselines (color-ch. res. = 256)							
Regularized SReLU NN [58]	68.70%	20.3M	77MB	1.9ms	565ns	(40G)	40M
Student-Teacher NN [63]	65.80%	1M	4MB	$112\mu s$	243ns	(2G)	2M
Student-Teacher NN [63]	74.30%	31.6M	121MB	2.9ms	960ns	(63G)	63M
Sparse Neural Networks					Sparsity		
PBW (ResNet32) [59]	38.64%				99.9%	(140M)	(140K)
MLPrune (ResNet32) [60]	36.09%				99.9%	(140M)	(140K)
ProbMask (ResNet32) [61]	76.87%				99.9%	(140M)	(140K)
SET-MLP [58]	74.84%	279K	4.7MB		98.6%	(558M)	558K

- **Binary Neural Networks**: FINN [52], BinaryEye [53], ReBNet [54], LowBitNN [55]
- **Sparse Neural Networks** for MNIST: Var.Dropout [56], L_0 regularization [57], SET-MLP [58]
- **Sparse Neural Networks** for CIFAR-10: PBW (ResNet32) [59], MLPrune (ResNet32) [60], ProbMask (ResNet32) [61], SET-MLP [58]
- **Sparse Function Net** [62]
- **Fully-Connected Baselines**: Regularized SReLU NN [58], Student-Teacher NN [63]

We gathered the comparison results from the deep differentiable logic gate paper [14].

3) TRAINING SETTING

During training, we initialized the connections and parameters of each neuron randomly (generated according to random-seed). Our experiments utilized a consistent number of neurons in each layer (excluding the input layer) and comprised 4 to 8 layers, following the approach in [14]. The Adam optimizer with a learning rate of 0.1 was employed for model optimization. We did not apply data augmentation or dropout to our proposed method. For the learnable scale parameter β, we initialized it empirically, setting it inversely proportional to the square root of the number of neurons defined in one hidden layer of the networks.

For classification tasks, we grouped the output into k groups, each containing n/k elements, as indicated in Eq. 16. The number of 1s in each group corresponded to the classification score, and we retrieved the predicted class using the arg max operation on the class scores. Training was performed using softmax cross-entropy loss with normalized scores [14].

TABLE 5. Statistics of benchmark datasets.

Dataset	#Classes	#Train	#Test	#Attribute	Attribute type
UCI-Adult	2	30162	15060	14	Categorical, Integer
UCI-Breast Cancer	2	207	70	9	categorical
MNIST	10	60,000	10,000	28×28	Gray Image
CIFAR-10	10	50,000	10,000	$32 \times 32 \times 3$	Color Image

TABLE 6. Logic gate network architectures used in our proposed model following [14].

Dataset	Model	Layers	Neurons / layer	Total # Param	Temperature (T)
UCI-Adult	—	5	256	1, 280	20
Breast Cancer	—	5	128	640	20
UCI-MNIST	—	6	64, 000	384, 000	30
CIFAR-10	small	4	12, 000	48, 000	30
	medium	4	128, 000	512 000	100
	large	5	256, 000	1, 280, 000	100
	large×2	5	512, 000	2, 560, 000	100
	large×4	5	1, 024, 000	5, 120, 000	100

TABLE 7. Compared neural networks (MLP) baseline architectures (including a ReLU activation) [14].

Dataset	Layers	Neurons / layer	Total #Param.
Adult	2	32	3, 810
Breast Cancer	2	8	434
MNIST	7	2, 048	22 609 930
CIFAR-10	5	1, 024	12, 597, 258

Throughout the experiments, we utilized the full set of 16 operators, as it was observed to perform better than reducing the set of operators [14].

We reported computational performance using binarized images for inference times. Inference times (T.) per image were measured using an NVIDIA A6000 GPU and single-threaded CPU running at 2.5 GHz.

For model architecture details, please refer to Table 6 in the Appendix, following the model architecture defined in the Diff Logic Net paper [14] for both our proposed model and the baseline models.

B. RESULTS

We evaluated the performance of our proposed method on the Adult Census and Breast Cancer datasets, considering model memory footprint, evaluation speed, and accuracy as evaluation metrics. For visual recognition tasks, we utilized the MNIST and CIFAR-10 datasets.

We conducted evaluations of the Diff Logic Net on our computer and reported its accuracy performance. Our proposed method is implemented based on the open-source code of Diff Logic Net, publicly available at github.com/Felix-Petersen/difflogic.

1) ADULT AND BREAST CANCER

On the Adult Census and Breast Cancer datasets, our method outperformed the compared methods, including Diff Logic Net [14], while maintaining fast inference speed. The results can be found in Table 2.

2) MNIST

Our proposed method demonstrated superior performance compared to several state-of-the-art methods, such as Binary Neural Networks, Sparse Neural Networks, and Sparse Function Net, in terms of classification accuracy, except for LowBitNN [55] and SET-MLP [58].

During the inference phase, our proposed method, built upon Diff Logic Net [14], showed a significant advantage, with 10% fewer binary operations and 12 times faster GPU computation time compared to FINN [52] of Binary Neural Networks.

LowBitNN [55] achieved the best performance among the compared methods on FPGA-based accelerators, utilizing convolutional neural networks with a Support Vector Machine (SVM) classifier. However, their method's reliance on an 8-bit activation function and ad-hoc SVM classifier posed drawbacks, as they deviated from the end-to-end neural network learning paradigm. Additionally, the use of convolutional layers in LowBitNN proved effective for image recognition, highlighting a limitation of the Diff Logic Net based method, which does not incorporate convolutional layers.

SET-MLP [58] achieved slightly better accuracy with full precision computation, but it required more binary operations to achieve these results.

Var. Dropout [56], which adopted variational dropout to sparsify fully connected neural networks, showed lower accuracy and larger computational costs (OPs).

In terms of binary operations (OPs), our Diff Logic Net based model outperformed all the other compared methods, showcasing the lowest number of binary operations. The numbers in parentheses are either extrapolated or estimated.

3) CIFAR-10

For CIFAR-10 [51], we preprocess the images by applying a binary embedding. Binarized input values are transformed into binary images using three thresholds: 0.25, 0.5, and 0.75. For our (small and medium) models in Table 4, the binary images have 4 channels, whereas the large models in Table 4 use 32 channels with 31 thresholds [14]. On the other hand, the other compared methods use the original three color channels with 256 intensity values.

Our proposed method outperformed the original Diff Logic Net model [14] throughout all models while preserving all inference computation time. Some compared methods, with larger parameters and computing costs in terms of binary or floating operations, showed better classification accuracy.

Transfer learning-based model, Student-Teacher NN [63], had 1 million parameters and was 64% larger than our largest model (*large*×2), but showed 3.69% better accuracy. This model [63] required 2 million floating-point operations (FLOPs), while our model required 2.56 million bit-wise logic operations without additional computational optimization.

Sparse Neural Networks, ProbMask [61], and SET-MLP [58] showed better accuracy with higher computational cost (approx. 140K FLOPs per image, meaning 1-2 orders of magnitude more expensive than Diff Logic Net and our model) [14]. More detailed analysis can be found in [14], and more analysis about Binary Neural Networks is found in [64].

Sparse neural networks are neural networks in which only a selected subset of connections is used. For an overview of sparse neural networks, refer to [65].

C. COMPUTATIONAL COSTS

The experimental computation costs in our study were calculated and estimated following the approach described in [14]. The computational costs in detail computed by the authors of [14] for efficient neural networks, including Binary Neural Networks (BNNs) and Sparse Neural Networks (Sparse NNs) can be seen at that paper.

To provide a brief explanation of binary and floating operations on CPUs and GPUs, we refer to [14]. A FLOP (floating-point operation) generally corresponds to many binary OPs (bit-wise operations). In practice, a float32 adder/multiplier is much more expensive than performing a bitwise logical operation on int64 data types. CPUs can perform around 3-10 int64 bit-wise operations per cycle, while floating-point operations usually require a full clock cycle. A conservative estimate for converting a non-sparse model is 100 OPs per 1 FLOP. Sparse execution usually brings an overhead of 10-100 times, making 1000 binary OPs per 1 FLOP a very conservative estimate for sparse float32 models.

VII. CONCLUSION

In this paper, we introduced a novel approach to enhance the performance of deterministic logic gate networks using relaxed logic gate neural networks based on the Gumbel reparameterization process. By introducing perturbations to the normalized logits with a scale factor, we significantly improved the generalization performance of the logic gate network. Notably, our proposed method has no impact on inference time, as it is only applied during training with a limited number of learnable scale parameters. The transformation from a deterministic softmax-based coefficient formulation to a stochastic sampling-based approach for differentiable logic gate networks led to improved generalization in deep learning frameworks. Although our experiments primarily focused on straight shape logic gate networks, we believe that our approach can be extended to more flexible architectures with varying numbers of hidden neurons and deeper layers, making it applicable to a wide range of neural network models. Our work contributes to the advancement of logic gate neural network research and offers valuable insights for future combinational logic model development and applications. The relaxed logic gate framework opens up new possibilities for optimizing and enhancing the performance of various neural network models, providing promising avenues for further exploration in the field. Furthermore, we intend to expand our approach to encompass other applications, notably natural language processing.

APPENDIX

A. SCALE AND SHIFT OF THE GUMBEL NOISE

We can write codes for scaling and shifting of the Gumbel noise (mentioned in Section III-B) using a PyTorch library.

```python
import torch
def gumbel(logits: Tensor) -> Tensor:
    '''This is modifed based on gumbel_softmax of
    PyTorch libaray
    Examples::
        >>> logits = torch.randn(20, 32)
        >>> # Generate a gumbel noise:
        >>> gumbel(logits)
    '''
    gumbels = (
        -torch.empty_like(logits, memory_format=
    torch.legacy_contiguous_format).exponential_()
    .log()
    )
    return gumbels
```

LISTING 1. PyToch code for Gumbel noise generation.

B. DATASET STATISTICS

We provide dataset statistics in Table 5.

C. MODEL ARCHITECTURES AND HYPERPARAMETERS

We provide model architecture details in Table 6 for deep logic gate networks and the compared MLP in Table 7. We follow the model architectures of Diff Logic Net [14].

REFERENCES

[1] A. Goyal and Y. Bengio, "Inductive biases for deep learning of higher-level cognition," *Proc. Roy. Soc. A, Math., Phys. Eng. Sci.*, vol. 478, no. 2266, Oct. 2022, Art. no. 20210068.

[2] I. Newton, "Principia mathematica," in *Book III, Lemma V, Case*, vol. 1, p. 1687, 1934.

[3] A. Newell and H. Simon, "The logic theory machine—A complex information processing system," *IEEE Trans. Inf. Theory*, vol. IT-2, no. 3, pp. 61–79, Sep. 1956.

[4] A. Newell, J. C. Shaw, and H. A. Simon, "Report on a general problem solving program," in *Proc. IFIP Congr.*, Pittsburgh, PA, USA, vol. 256, 1959, p. 64.

[5] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[6] F. Rosenblatt, "The perceptron: A perceiving and recognizing automaton," Cornell Aeronaut. Lab., Buffalo, NY, USA, Tech. Rep., 1957.

[7] A. Newell and J. C. Shaw, "Programming the logic theory machine," in *Proc. Western Joint Comput. Conf. Techn. Rel.*, 1957, pp. 230–240.

[8] A. Katharopoulos and F. Fleuret, "Not all samples are created equal: Deep learning with importance sampling," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2525–2534.

[9] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn.*, 2009, pp. 41–48.

[10] C. J. Maddison, D. Tarlow, and T. Minka, "A* sampling," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 1–9.

[11] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–14.

[12] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-softmax," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2017, pp. 1–9.

[13] I. A. M. Huijben, W. Kool, M. B. Paulus, and R. J. G. van Sloun, "A review of the Gumbel-max trick and its extensions for discrete stochasticity in machine learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 2, pp. 1353–1371, Feb. 2023.

[14] F. Petersen, C. Borgelt, H. Kuehne, and O. Deussen, "Deep differentiable logic gate networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 35, 2022, pp. 2006–2018.

[15] C. H. Roth Jr., L. L. Kinney, and E. B. John, *Fundamentals of Logic Design*. Boston, MA, USA: Cengage Learning, 2020.

[16] J. M. Rabaey, *Digital Integrated Circuits a Design Perspective*. Pearson, 1999.

[17] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 10, pp. 2628–2639, Oct. 2020.

[18] W. Zeng, A. Davoodi, and R. O. Topaloglu, "Sampling-based approximate logic synthesis: An explainable machine learning approach," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.

[19] N. Cingillioglu and A. Russo, "DeepLogic: Towards end-to-end differentiable logical reasoning," 2018, *arXiv:1805.07433*.

[20] G. Huang, J. Hu, Y. He, J. Liu, M. Ma, Z. Shen, J. Wu, Y. Xu, H. Zhang, and K. Zhong, "Machine learning for electronic design automation: A survey," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 5, pp. 1–46, Jun. 2021.

[21] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 344–351.

[22] S. Oh, Y. Jung, I. Lee, and N. Kang, "Design automation by integrating generative adversarial networks and topology optimization," in *Proc. 2A: 44th Design Autom. Conf.* New York, NY, USA: American Society of Mechanical Engineers, vol. 51753, Aug. 2018, Art. no. V02AT03A008.

[23] L. Feng, W. Liu, C. Guo, K. Tang, C. Zhuo, and Z. Wang, "GANDSE: Generative adversarial network-based design space exploration for neural network accelerator design," *ACM Trans. Design Autom. Electron. Syst.*, vol. 28, no. 3, pp. 1–20, May 2023.

[24] H. Gupta, S. Taek Kong, R. Srikant, and W. Wang, "Almost Boltzmann exploration," 2019, *arXiv:1901.08708*.

[25] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proc. Mach. Learn.*, 1990, pp. 216–224.

[26] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1–9.

[27] R. Shetty, M. Rohrbach, L. A. Hendricks, M. Fritz, and B. Schiele, "Speaking the same language: Matching machine to human captions by adversarial training," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 4135–4144.

[28] N. Cesa-Bianchi, C. Gentile, G. Lugosi, and G. Neu, "Boltzmann exploration done right," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–9.

[29] A. Biswas, T. T. Pham, M. Vogelsong, B. Snyder, and H. Nassif, "Seeker: Real-time interactive search," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2867–2875.

[30] J. E. Ayers, *Digital Integrated Circuits: Analysis and Design*. Boca Raton, FL, USA: CRC Press, 2018.

[31] N. Cingillioglu, *End-to-End Neuro-Symbolic Learning of Logic-Based Inference*. London, U.K.: Imperial College London, 2022.

[32] W. Liu, R. Lin, Z. Liu, L. Xiong, B. Schölkopf, and A. Weller, "Learning with hyperspherical uniformity," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2021, pp. 1180–1188.

[33] T. Wang and P. Isola, "Understanding contrastive representation learning through alignment and uniformity on the hypersphere," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9929–9939.

[34] J. Xu and G. Durrett, "Spherical latent spaces for stable variational autoencoders," 2018, *arXiv:1808.10805*.

[35] T. R. Scott, A. C. Gallagher, and M. C. Mozer, "Von Mises–Fisher loss: An exploration of embedding geometries for supervised learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10612–10622.

[36] C. M. Bishop, "Training with noise is equivalent to Tikhonov regularization," *Neural Comput.*, vol. 7, no. 1, pp. 108–116, Jan. 1995.

[37] W. K. Hastings, "Monte Carlo sampling methods using Markov chains and their applications," *Biometrika*, vol. 57, no. 1, pp. 97–109, Apr. 1970.

[38] F. Orabona, T. Hazan, A. Sarwate, and T. Jaakkola, "On measure concentration of random maximum a-posteriori perturbations," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 432–440.

[39] E. J. Gumbel, *Statistical Theory of Extreme Values and Some Practical Applications: A Series of Lectures*, vol. 33. Washington, DC, USA: Government Printing Office, 1948.

[40] J. I. Yellott, "The relationship between Luce's choice axiom, Thurstone's theory of comparative judgment, and the double exponential distribution," *J. Math. Psychol.*, vol. 15, no. 2, pp. 109–144, Apr. 1977.

[41] E. Beck, C. Bockelmann, and A. Dekorsy, "Concrete MAP detection: A machine learning inspired relaxation," in *Proc. 24th Int. ITG Workshop Smart Antennas*, Feb. 2020, pp. 1–5.

[42] Y. Li, J. Liu, G. Lin, Y. Hou, M. Mou, and J. Zhang, "Gumbel-softmax-based optimization: A simple general framework for optimization problems on graphs," *Comput. Social Netw.*, vol. 8, no. 1, pp. 1–16, Dec. 2021.

[43] G. Papandreou and A. L. Yuille, "Perturb-and-MAP random fields: Using discrete optimization to learn and sample from energy models," in *Proc. Int. Conf. Comput. Vis.*, Nov. 2011, pp. 193–200.

[44] D. Tarlow, R. Adams, and R. Zemel, "Randomized optimum models for structured prediction," in *Proc. Artif. Intell. Statist.*, 2012, pp. 1221–1229.

[45] W. Kool, H. Van Hoof, and M. Welling, "Ancestral Gumbel-top-K sampling for sampling without replacement," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 1726–1761, 2020.

[46] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, *arXiv:1308.3432*.

[47] R. E. Simpson, *Introductory Electronics for Scientists and Engineers*. Boston, MA, USA: Allyn & Bacon, 1974.

[48] R. Kohavi and B. Becker. (1996). *Uci Machine Learning Repository: Adult Data Set*. [Online]. Available: https://archive.ics.uci.edu/ml/machine-learning-databases/adult

[49] M. Kelly, R. Longjohn, and K. Nottingham, "The UCI machine learning repository," [Online]. Available: https://archive.ics.uci.edu

[50] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[51] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, 2009.

[52] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2017, pp. 65–74.

[53] P. Jokic, S. Emery, and L. Benini, "BinaryEye: A 20 kfps streaming camera system on FPGA with real-time on-device image recognition using binary neural networks," in *Proc. IEEE 13th Int. Symp. Ind. Embedded Syst. (SIES)*, Jun. 2018, pp. 1–7.

[54] M. Ghasemzadeh, M. Samragh, and F. Koushanfar, "ReBNet: Residual binarized neural network," in *Proc. IEEE 26th Annu. Int. Symp. Field-Programmable Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 57–64.

[55] J. Zhan, X. Zhou, and W. Jiang, "Field programmable gate array-based all-layer accelerator with quantization neural networks for sustainable cyber-physical systems," *Softw., Pract. Exper.*, vol. 51, no. 11, pp. 2203–2224, Nov. 2021.

[56] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2498–2507.

[57] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," 2017, *arXiv:1712.01312*.

[58] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Commun.*, vol. 9, no. 1, p. 2383, Jun. 2018.

[59] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," 2015, *arXiv:1510.00149*.

[60] W. Zeng and R. Urtasun, "MLPRUNE: Multi-layer pruning for automated neural network compression," in *Proc. ICLR* , 2018, pp. 1–12.

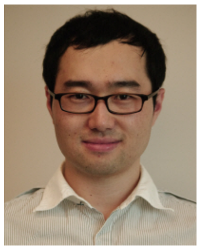
[61] X. Zhou, W. Zhang, H. Xu, and T. Zhang, "Effective sparsification of neural networks with global sparsity constraint," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 3598–3607.

[62] A. Gaier and D. Ha, "Weight agnostic neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.

[63] G. Urban, K. J. Geras, S. E. Kahou, O. Aslan, S. Wang, R. Caruana, A. Mohamed, M. Philipose, and M. Richardson, "Do deep convolutional nets really need to be deep and convolutional?" 2016, *arXiv:1603.05691*.

[64] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognit.*, vol. 105, Sep. 2020, Art. no. 107281.

[65] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 10882–11005, 2021.



YOUNGSUNG KIM received the B.S., M.S., and Ph.D. degrees in electrical and electronic engineering from Yonsei University, in 2006, 2008, and 2012, respectively. He is currently a Faculty Member with the Department of Artificial Intelligence, Graduate School of Electrical and Computer Engineering, Inha University. He was a Research Staff Member with the Samsung Advanced Institute of Technology (SAIT), from 2014 to 2022; and a Postdoctoral Associate with MIT SMART Center, from 2013 to 2014.