SONG: SELF-ORGANIZING NEURAL GRAPHS

Anonymous authors
Paper under double-blind review

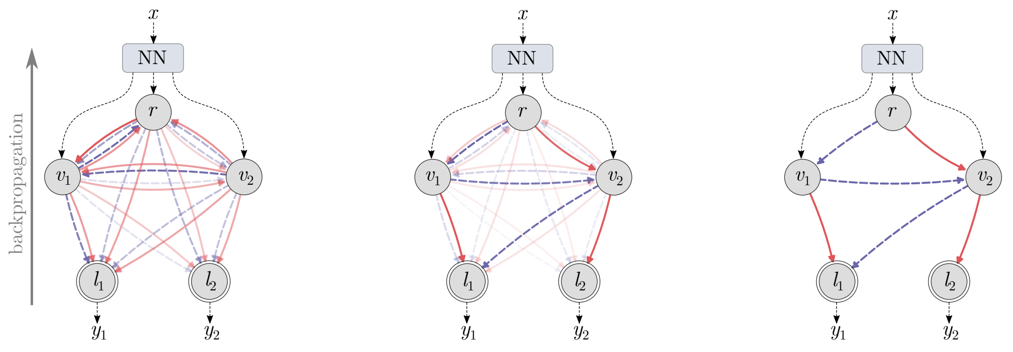
ABSTRACT

Recent years have seen a surge in research on combining deep neural networks with other methods, including decision trees and graphs. There are at least three advantages of incorporating decision trees and graphs: they are easy to interpret since they are based on sequential decisions, they can make decisions faster, and they provide a hierarchy of classes. However, one of the well-known drawbacks of decision trees, as compared to decision graphs, is that decision trees cannot reuse the decision nodes. Nevertheless, decision graphs were not commonly used in deep learning due to the lack of efficient gradient-based training techniques. In this paper, we fill this gap and provide a general paradigm based on Markov processes, which allows for efficient training of the special type of decision graphs, which we call Self-Organizing Neural Graphs (SONG). We provide an extensive theoretical study of SONG, complemented by experiments conducted on Letter, Connect4, MNIST, CIFAR, and TinyImageNet datasets, showing that our method performs on par or better than existing decision models.

1 INTRODUCTION

Neural networks (NNs) and decision trees (DTs) are two exceptionally powerful machine learning models with a rich and successful history in machine learning. However, they typically come with mutually exclusive benefits and limitations. NNs outperform conventional pipelines by jointly learning to represent and classify data (Krizhevsky et al., 2012). However, they are widely opaque and suffer from a lack of transparency and explainability (Rudin, 2019). On the other hand, it is easy to explain predictions of DTs because they depend on a relatively short sequence of decisions (Wan et al., 2020). However, they usually do not generalize as well as deep neural networks (Frosst & Hinton, 2017). As a result, a strong focus is recently put on joining the positive aspects of both models (Frosst & Hinton, 2017; Suárez & Lutsko, 1999; Kontschieder et al., 2015; Nauta et al., 2020; Murthy et al., 2016; Tanno et al., 2019; Alaniz et al., 2021; Wan et al., 2020). There are methods that combine NNs and soft decision trees with partial membership in each node (Frosst & Hinton, 2017; Suárez & Lutsko, 1999; Kontschieder et al., 2015; Nauta et al., 2020). Others use trees to explain NNs (Zhang et al., 2019; Bastani et al., 2017) or to obtain their optimal hierarchical structure (Murthy et al., 2016; Tanno et al., 2019; Alaniz et al., 2021). Finally, some models replace the final softmax layer of a neural network with a hierarchical binary decision tree (Wan et al., 2020; Morin & Bengio, 2005; Mikolov et al., 2011).

While decision trees can increase the performance and interpretability of NNs, they usually suffer from exponential growth with depth (Shotton et al., 2012), repeating nodes (Frosst & Hinton, 2017), and suboptimal structure, often selected manually before training (Wan et al., 2020). Hence, more and more attention is put on combining NNs with decision graphs instead of trees (Baek et al., 2017; Veit & Belongie, 2018; He et al., 2016; Mullapudi et al., 2018; Keskin & Izadi, 2018). Decision graphs have a few advantages when compared to decision trees. They have a flexible structure that allows multiple paths from the root to each leaf. As a result, nodes are reused, resulting in simpler and smaller models, which solves the replication problem (Oliver, 1993) and provides models easier to comprehend by humans (Breslow & Aha, 1997). Moreover, decision graphs require substantially less memory while considerably improving generalization (Shotton et al., 2016). Nevertheless, decision graphs are not commonly used in deep learning due to a lack of efficient gradient-based training techniques.



(a) random initialization (b) intermediate phase (c) trained SONG

Figure 1: Training stages of SONG that uses gradient descent to modify the graph structure and transition probabilities. Based on an input x, the backbone neural network (NN) extracts a vector representation, which is passed to SONG to obtain a prediction for each class (y_1 and y_2). At the beginning of training, a graph has root r, nodes v_1 and v_2, leaves l_1 and l_2, and randomly initialized edges (a). In the successive training iterations, the entropy of edge weights grows (b), finally resulting in a sparse binary graph, with two strong edges outgoing from each node (c). Notice that SONG contains two alternative sets of edges between the nodes (dashed blue arrows and solid red arrows, respectively) that are combined based on the input (see Figure 2 for details).

In this paper, we introduce Self-Organizing Neural Graphs (SONGs)[1], a special type of decision graphs that generalize methods like Soft Decision Tree (SDT) (Frosst & Hinton, 2017) and Neural-Backed Decision Trees (NBDT) (Wan et al., 2020), and as a differentiable solution are applicable to any deep learning pipeline (Figure 1). Moreover, in contrast to the fixed structure of the existing methods (Wan et al., 2020; Frosst & Hinton, 2017), SONGs can strengthen or weaken an edge between any pair of nodes during training to optimize their structure. We illustrate this process in Figure 1. In the beginning, the edges have random weights. However, in successive steps of training, the structure is corrected with backpropagation, and it gets sparse and converges to the binary directed acyclic graphs (Platt et al., 1999) (see Figure 1c). We prove this statement in Section 4.

Our contributions can be summarized as follows:

- We introduce Self-Organizing Neural Graphs (SONGs), a new paradigm of end-to-end decision graph training based on Markov processes that simultaneously learn the optimal graph structure and transition probabilities.
- Our model is fully differentiable and thus suitable for combined training with other deep learning models.
- We prove empirically and theoretically that SONGs during training converge to sparse binary acyclic graphs and can be interpreted as diagrams of consecutive decisions.
- Our method performs on par or outperforms decision trees trained in a similar setup and does not require the graph/tree structure to be predefined before training.

2 Related works

Decision trees Numerous Decision Tree (DT) algorithms have been developed over the years (Quinlan, 2014; Loh, 2011; Shafer et al., 1996; Mehta et al., 1996) and after the success of deep learning, much research relates to combining DTs with neural networks. As a result, Soft Decision Tree (SDT) was introduced, allowing for the partial membership of a sample in the nodes that make up the tree structure (Suárez & Lutsko, 1999), also trained in distillation setup (Frosst & Hinton, 2017). This idea was also used in (Kontschieder et al., 2015) that trains a set of classification trees and a backbone network in an end-to-end manner. Moreover, it was recently used in (Nauta et al., 2020) to faithfully visualize the model using nodes with prototypes (Chen et al., 2018) instead of classifiers. Trees were also used to explain the previously trained black box models (Zhang et al., 2019; Bastani et al., 2017). More advanced methods automatically generate deep networks with a tree structure in a multi-step or

[1]The code is available at: [anonymized]

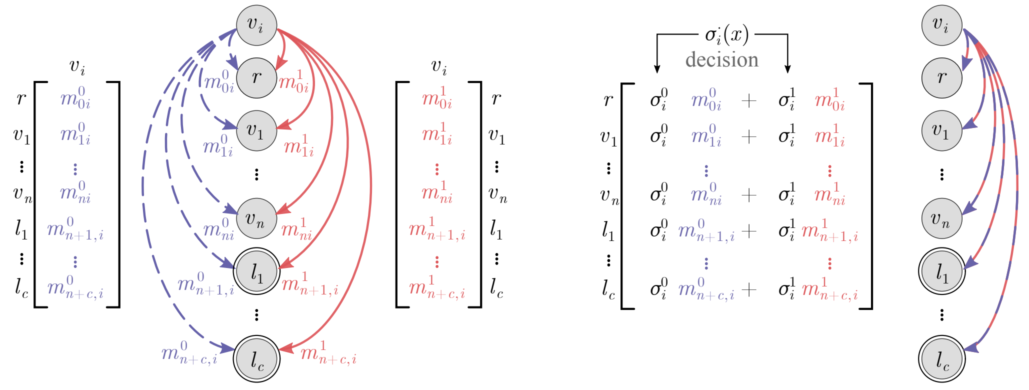


Figure 2: SONG contains two alternative transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ that aggregate the probability of moving from a particular node v_i to all other nodes. In (a), they are represented as dashed blue and solid red arrows, respectively. Each node obtains input data x and makes a binary decision with probabilities σ^0_i and σ^1_i of using one transition or another. As $\sigma^0_i + \sigma^1_i = 1$, SONG can be transformed to a standard directed graph by combining $m^0_{\cdot i}$ and $m^1_{\cdot i}$, as presented in (b). During training, both $\sigma^{\cdot}_i$ and $m^{\cdot}_{\cdot i}$ are trained to obtain the optimal decision graph as presented in Figure 1 of the paper.

an end-to-end manner (Murthy et al., 2016; Tanno et al., 2019; Alaniz et al., 2021; Wan et al., 2020; Ahmed et al., 2016). In contrast to the presented methods, our approach has a structure of a graph trained together with other model parameters in an end-to-end manner.

Decision graphs A decision graph is a well-studied classifier and has been used to solve many real-world problems (Sudo et al., 2018). When implemented as Directed Acyclic Graphs (DAG), it leads to accurate predictions while having lower model complexity, subtree replication, and training data fragmentation compared to decision trees (Shotton et al., 2016). However, most of the existing algorithms for learning DAGs involve training a conventional tree that is later manipulated into a DAG (Kohavi & Li, 1995; Oliveira & Sangiovanni-Vincentelli, 1996; Oliver, 1992; Chou, 1991) and, as such, are difficult to be directly adopted into neural networks. Hence, alternative approaches were proposed, like (Baek et al., 2017), which maintains the structure of the standard convolutional neural networks (CNNs) but uses additional routing losses at each layer to maximize the class-wise purity (like in growing decision trees) using data activation according to the class label distribution. Another method (Veit & Belongie, 2018) introduces identity skip-connections similar to ResNets (He et al., 2016) that are executed or skipped depending on the gate response for an input. A similar gate mechanism was used in (Mullapudi et al., 2018) to choose branches specialized for different inputs, whose outputs are combined to make the final predictions. Finally, (Keskin & Izadi, 2018) embeds infinitely many filters into low dimensional manifolds parameterized by compact B-splines and maximizes the mutual information between spline positions and class labels to specialize for classification tasks optimally. Such a mechanism significantly reduces runtime complexity. In contrast to existing methods, SONG is a directed graph that can be adapted to any deep architecture and trained in an efficient gradient-based manner.

3 Self-organizing neural graphs

To adequately describe the Self-Organizing Neural Graph (SONG), we first define a more abstract structure that we call Soft Binary Directed Graph (SBDG). SBDG is considered binary because there are two alternative sets of edges, and soft because those sets are combined into one target set of edges depending on the input. Then, based on SBDG, we define SONG and describe how to use them as a decision model. Finally, we present method limitations and show how to overcome them with additional regularizers. The below definitions correspond to single-label classification for the clarity of description. However, they could be easily extended to other tasks, like multi-label classification or regression.

Soft binary directed graphs Soft Binary Directed Graph (SBDG) is a directed graph, which can be viewed as a probabilistic model. It is defined as graph $G = (V, E^0, E^1)$, with V corresponding to a set of nodes and E^0, E^1 corresponding to two alternative sets of edges, where:

- Set V contains two types of nodes:
  - internal nodes $v_0, \dots, v_n$, with v_0 specified as root r,
  - leaves $l_1, \dots, l_c$, each exclusively associated with one class from set $\{1, \dots, c\}$,
- Set E^d, for $d \in \{0, 1\}$, contains all possible edges with weights m^d_{ji} corresponding to the probability of moving from node u_i to $u_j \in V$, as presented in Figure 2a. In the following, the aggregated probabilities of moving from node u_i to other nodes will be called a transition vector and denoted as $m^d_{\cdot i}$.
- If u_i is a leaf, then $m^d_{ji} = \delta_{ji}$ (Kronecker delta), which means that it is impossible to move out from the leaves.
- Each internal node u_i makes binary decisions $d \in \{0, 1\}$ with probabilities σ^d_i of using edges from set E^d.
- $\sigma^0_i + \sigma^1_i = 1$ and G can be transformed to a standard directed graph by combining $m^0_{\cdot i}$ and $m^1_{\cdot i}$ using the following formula for each node u_i: $\sigma^0_i m^0_{\cdot i} + \sigma^1_i m^1_{\cdot i}$. This process is presented in Figure 2b.

Notice that if all transition vectors are binary, then after removing the edges with zero probability, SBDG becomes a binary directed graphs (Platt et al., 1999).

Self-organizing neural graphs Self-Organizing Neural Graph (SONG) is a fully differentiable adaptation of SBDG that can be combined with various deep architectures. SONG is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}^0, \mathcal{E}^1)$, where $\mathcal{V}$, $\mathcal{E}^0$, $\mathcal{E}^1$ implement V, E^0, and E^1 of SBDG, and are obtained for input point x in the following way:

- The probability of decision $d = 1$ in node u_i is obtained as $\sigma^1_i(x) = \sigma(xw_i + b_i)$, where σ is the sigmoid logistic function, w_i is a filter function, and b_i is a bias[2].
- The probability of decision $d = 0$ equals $\sigma^0_i(x) = 1 - \sigma^1_i(x)$.
- The probability of moving from internal nodes is defined by two matrices $\mathrm{M}^d = [m^d_{ji}] \in \mathbb{R}^{(n+c)\times n}$, for $d = \{0, 1\}$, with positive values and columns summing up to 1. In our implementation, we obtain such matrices by applying softmax to each of their columns.

Notice that $\{w_i\}_{i=1,\dots,n}$, $\{b_i\}_{i=1,\dots,n}$, M^0, and M^1 are trainable parameters of the model.

Finally, we define a directed graph $\mathcal{G}_x = (\mathcal{V}, \mathcal{E})$ generated for input x where $\mathcal{E}$ corresponds to the combination of matrices M^0 and M^1:

$$\mathrm{M}_x = \mathbb{1}\sigma_x^T \odot \mathrm{M}^1 + \mathbb{1}(\mathbb{1} - \sigma_x)^T \odot \mathrm{M}^0, \tag{1}$$

where $\sigma_x = [\sigma^1_0(x), \dots, \sigma^1_n(x)]^T$, symbol $\odot$ denotes the Hadamard product, and $\mathbb{1}$ is the all-ones vector of dimension n.

Decision model Matrix M_x contains the probability of moving from internal nodes to all nodes of the graph. However, to apply the theory of the Markov processes, it needs to be extended by columns corresponding to the leaves (as presented on the left side of Figure 3):

$$P_x = \left[\begin{array}{c:c} \mathrm{M}_x & \begin{array}{c} 0 \\ \hdashline \mathrm{I} \end{array} \end{array}\right] \in \mathbb{R}^{(n+c)\times(n+c)}, \tag{2}$$

where $0 \in \mathbb{R}^{n\times c}$ is zero matrix and $\mathrm{I} \in \mathbb{R}^{c\times c}$ is an identity matrix. As a result, we obtain a square stochastic (transition) matrix used to describe the transitions of a Markov chain. While P_x contains the probability of moving from u_i to u_j in one time step, it can be easily used to obtain a similar probability for N steps by calculating the N-th power of P_x. Finally, the resulting matrix can be multiplied by vector $v = [1, 0, ..., 0]^T$ to obtain the probability of moving from the root to any node of the graph, including leaves, whose probability is the output of the model. We present a simple example illustrating this process on the right side of Figure 3. More examples are provided in Figures 12, 13, 14, and 15.

Regularizations Similarly as in Soft Decision Trees (SDT) (Frosst & Hinton, 2017), we observe that our graphs require additional training regularizers. The reasons for that are threefold. First, SONG may get stuck on plateaus in which one or more $\sigma^d_i(x)$ is 0 for all input samples x, and the

[2]In practice, this probability could also be obtained with any NN that ends with a sigmoid function.

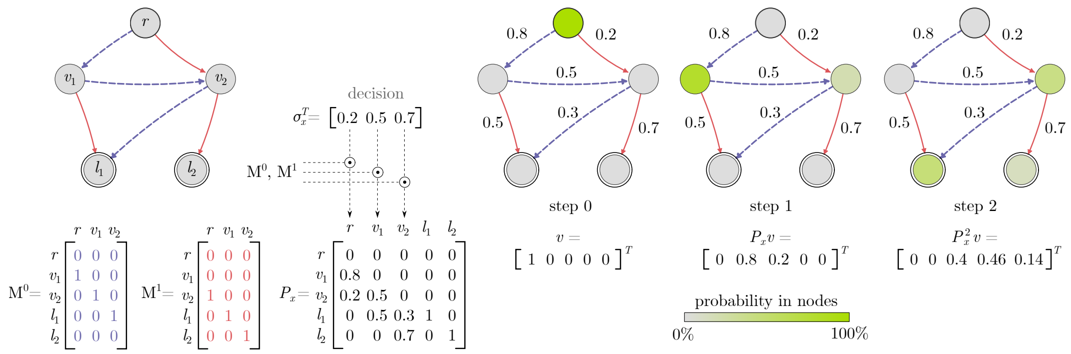

decision
$\sigma_x^T = [0.2\ 0.5\ 0.7]$
M^0, M^1
step 0
step 1
step 2
$v = [1\ 0\ 0\ 0\ 0]^T$
$P_x v = [0\ 0.8\ 0.2\ 0\ 0]^T$
$P_x^2 v = [0\ 0\ 0.4\ 0.46\ 0.14]^T$
probability in nodes
0%
100%

Figure 3: Construction of the transition matrix and successive steps of our Markov process. On the left, a graph with its matrices M^0 and M^1 is presented, followed by an exemplary decision vector σ_x and the resulting matrix P_x. On the right, the flow in a graph is depicted for 3 consecutive steps. At first, the probability is entirely placed in the root. However, in the next steps, the distribution splits between nodes according to the transition probabilities, reaching leaves in step 3. The probabilities in the leaves after all steps are class probabilities inferred by the model (the number of steps is considered as a method hyperparameter).

gradient of the sigmoid logistic function for this decision is always very close to zero. Second, if SONG is uncertain of its predictions, it can safely hold the probabilities in internal nodes instead of moving them to leaves, which results in a small accumulated probability in the latter. Third, SONG tends to binarize what is positive in general, but if this binarization appears too early, the model can get stuck in a local minimum. Therefore, to prevent model degeneration, we introduce three types of regularization. The first one, called node regularization, is based on (Frosst & Hinton, 2017) and encourages each internal node to make equal use of both sets of edges $\mathcal{E}^0$, passing half of the training samples to one direction (using M_0) and the other half to the other direction (with M_1). The second one, called leave regularization, enforces the summary probabilities in leaves to be close to 1. Finally, we apply Gumbel-softmax (Jang et al., 2016) instead of softmax to each column of matrices M_0 and M_1 to explore the trajectories of the graph better. Details are provided in Appendix D.

4 Theoretical analysis

In this section, we show that the graph structure generated by SONG is binarized during training, which increases its accuracy and makes the model easier to interpret and understand, see Figure 6. Due to the page limit, we move all proofs to the Appendix B, while here we only describe the intuition behind our ideas.

To study decision graphs, we use the probabilistic model over trajectories defined by arbitrary SBDG. A trajectory of length N, starting at the root of SBDG G, is defined as $T = (u_{i_t})_{t=1..N}$ with binary decisions $d_t \in \{0, 1\}$, $i_t \in I$, where I denotes the set of node indexes. Thus, our trajectory starts at the root ($i_0 = 0$) and successively passes through nodes $u_{i_{t_1}}, \ldots, u_{i_{t_N}}$. The position of trajectory after time t is defined as $T(t) = u_{i_t}$ and the probability of trajectory T is defined as

$$\mathbf{prob}(T; G) = \prod_{t=1}^{N} (\sigma_{i_{t-1}}^{d_t} \cdot m_{i_t i_{t-1}}^{d_t}).$$

Then the probability of reaching leaf l after N steps with a random trajectory T equals $\mathbf{prob}(T(N) = l \,|\, T \sim G)$, where $T \sim G$ denotes that we sample trajectories with respect to distribution given by $\mathbf{prob}(\cdot; G)$. Given a SONG $\mathcal{G}$, to analyze a decision made on a single data point x, we denote the probability that a random trajectory of length N reaches leaf corresponding to the class y as $\mathbf{prob}(T(N) = y \,|T \sim \mathcal{G}_x)$.

Now we present the main idea why during training SONG tends to binarize connections (for the detailed proofs we refer to the Appendix B). For a fixed $d \in \{0, 1\}$, we denote $G[i, j; d]$ as the graph that makes a decision of moving from u_i to u_j with probability 1. Observe that $G[i, j; d]$ comes from binarization in graph G of the connections from u_i under the choice d. In the following theorem, we show that if G has no cycles, then we can decompose the probability of its trajectory into the mixture of such binarized graphs.

Theorem 4.1. *Let G be a SBDG where the probability of visiting twice an arbitrary internal node by a trajectory of length N is zero. Moreover, u_i be an internal node, fixed $d \in \{0, 1\}$, and an arbitrary trajectory T of length N. Then*

$$\mathbf{prob}(T; G) = \sum_{j=1}^{n} m_{ji}^{d} \mathbf{prob}(T; G[i, j, d]). \tag{3}$$

Roughly speaking, the proof follows from the fact that under the assumptions of the theorem the set of the possible trajectories passing through u_i with given d can be decomposed into n disjoint sets of trajectories which after passing through u_i visit u_j, for $j = 1, \ldots, n$.

Now we proceed to the consequence of the above result for SONG $\mathcal{G}$. The accuracy of $\mathcal{G}$ over set X is defined as the probability of predicting the correct class

$$\mathrm{acc}(\mathcal{G}; X) = \tfrac{1}{K} \sum_{i=1}^{K} \mathbf{prob}(T(N) = y_i \, | T \sim \mathcal{G}_{x_i}).$$

Theorem 4.2. *Let $\mathcal{G}$ be a SONG. We assume that for every $x \in X$ no trajectory in $\mathcal{G}_x$ of length N that visits twice the same internal node with nonzero probability. Let a node index $i \in \{1, \ldots, n\}$ and $d \in \{0, 1\}$ be fixed. Then*

$$\mathrm{acc}(\mathcal{G}; X) = \sum_{j=1}^{n} m_{ji}^{d} \mathrm{acc}(\mathcal{G}[i, j, d]; X). \tag{4}$$

The consequence of the above theorem is profound. Namely, since the convex combination of nonnegative reals is bounded by their maximum, we obtain that

$$\mathrm{acc}(\mathcal{G}; X) \leq \max_{j=1..n} \mathrm{acc}(\mathcal{G}[i, j, d]; X).$$

This means, that if we properly binarize the connections from u_i, i.e. replace the SONG $\mathcal{G}$ by $\mathcal{G}[i, \bar{j}, d]$ (with $\bar{j} = \arg\max_j \mathrm{acc}(\mathcal{G}[i, j, d]; X)$), we obtain a model with at least the same accuracy. By applying the above operation for all nodes of $\mathcal{G}$ and all choices of d we will obtain a binary graph which has accuracy at least that of $\mathcal{G}$. Statistically SONG increases its accuracy if at least one of the inequalities in equation 4 is strong for an arbitrary i and d. Summarizing, binarization of the connections between nodes increases the performance of the model, which is formalized in the following theorem.

Corollary 4.1. *We assume that we are given a SONG $\mathcal{G}$ such that for an arbitrary $x \in X$ no trajectory in $\mathcal{G}_x$ of length N visits twice the same internal node with nonzero probability. Then we can remove some connections from $\mathcal{G}$ to make a binary SONG, in such a way that the accuracy on X is at least that of the original SONG $\mathcal{G}$.*

Observe, that if the training of the SONG is successful and the model obtains accuracy close to one, the probability of arriving in the leaves becomes also close to one, see Figure 4. The condition that we arrive with probability one in leaves after N jumps clearly implies that we cannot visit twice the same internal node (in that case we could return to this node with nonzero probability). The formal zero loss assumption (perfectly trained model) is commonly accepted in deep learning literature – as observed in (Ma et al., 2018) "most of the modern machine learning, especially deep learning, relies on classifiers which are trained to achieve near zero classification and regression losses on the training data". This statement is supported by our experimental analysis, which shows that the loss for training and test sets drop rapidly, achieving near-zero values at the final epochs of training (see Figure 7).

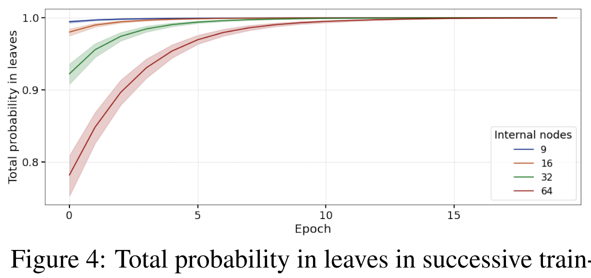


Figure 4: Total probability in leaves in successive training epochs for SONG trained on MNIST dataset. Each color represents a different number of internal nodes (9, 16, 32, and 64), and each line corresponds to mean and standard deviation over multiple training repetitions.

Table 1: Comparison of models with deep architecture in terms of model features and accuracy on MNIST, CIFAR10 (CIF10), CIFAR100 (CIF100), and TinyImageNet (TinyIN). ResNet18 was used to extract the vector representation of input images for DNDF (Kontschieder et al., 2015), DT, NBDT (Wan et al., 2020), RDT (Alaniz et al., 2021), and SONG. For DDN (Murthy et al., 2016), DCDJ (Baek et al., 2017), and ANT-A (Tanno et al., 2019), the backbone models are provided in the brackets. "Ex?" indicates if the method retains interpretable properties such as pure leaves, sequential decisions, and non-ensemble. "SO?" indicates if the model is self-organized (does not require a predefined structure). "EE?" indicates if the structure and weights of model are trained in an end-to-end continuous manner.

Method	Ex?	SO?	EE?	MNIST	CIF10	CIF100	TinyIN
DDN (NiN)	✗	✓	✗	-	90.32	68.35	-
DCDJ (NiN)	✗	✓	✓	-	-	69.00	-
ANT-A* (n/a)	✓	✓	✗	**99.36**	93.28	-	-
ResNet18	✗	✗	✗	98.91	94.93	75.82	63.05
DNDF	✗	✗	✗	97.20	94.32	67.18	44.56
DT	✓	✗	✗	-	93.97	64.45	52.09
NBDT	✓	✗	✗	-	94.82	**77.09**	**64.23**
NBDT w/o hierarchy	✓	✓	✗	-	94.52	74.97	-
RDT	✓	✓	✓	-	93.12	-	-
SONG (ours)	✓	✓	✓	98.81	**95.62**	76.26	61.99

While the above fact demonstrates that introducing binary connections improves the performance of the model, it may be not obvious that SONG binarizes the connections during gradient training. To see that, let us first notice that after calculating the loss function for each input sample, the gradient is propagated from the leaves back to the root. It implies that all computation paths that end in the internal nodes do not influence the transition probabilities in the graph. One may also notice, that paths leading to the correct leaves (corresponding to the correct class for the input sample) are strengthened by applying the chain rule to compute gradients with respect to the transition probabilities. Conversely, paths ending in the incorrect leaves are weakened. Because transition probabilities are softmax outputs limited to the range $[0, 1]$, only the strongest path ending in the correct leaf is iteratively reinforced at the expense of other paths. This aggressive exploitation of the main paths causes SONG to binarize (see Figure 6 and Appendix C).

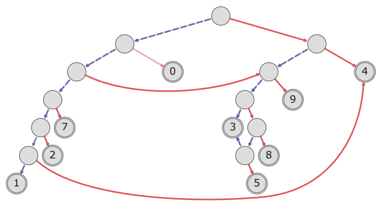
5 Experiments

In this section, we analyze the accuracy of the SONGs trained on Letter (Asuncion & Newman, 2007), Connect4 (Asuncion & Newman, 2007), MNIST (LeCun et al., 2010), CIFAR10 (Krizhevsky et al., 2009), CIFAR100 (Krizhevsky et al., 2009), and TinyImageNet (Le & Yang, 2015) datasets and compare it with the state of the art methods (Wan et al., 2020; Kontschieder et al., 2015; Murthy et al., 2016; Tanno et al., 2019; Alaniz et al., 2021; Baek et al., 2017). We examine how the number of nodes and steps influence the structure of graphs, the number of internal nodes used by the model, the number of back edges, and the distance from the root to leaves. Moreover, we explain how the probability of back and cross edge changes in the successive training steps. Finally, we provide a detailed comparison with SDT (Frosst & Hinton, 2017) and present sample graphs obtained for the MNIST dataset. In all experiments, we use leaves normalization and Gumbel-softmax, and we treat node regularization as a hyperparameter of the model. While this section presents only the most important findings for the sake of clarity, the experimental setup and detailed results can be found in Appendix G.

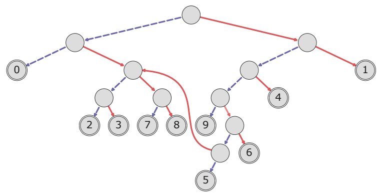
SONG in deep learning setup In the first experiment, we apply SONG at the top of the backbone Convolutional Neural Network (CNN) without the final linear layer. CNN takes the input image and generates the representation, which is passed to the SONG. SONG processes the representation and returns the predictions for each class, which are then used with target labels to calculated Binary Cross-Entropy (BCE) loss. As a backbone network, we use ResNet18 for all datasets except MNIST, for which we employ a smaller network (see Appendix G for details).

Table 2: Comparison of SDT (Frosst & Hinton, 2017) and shallow SONG (SONG-S) on three datasets, where shallow corresponds to direct flattened inputs (no backbone network used). The accuracy of each model is reported along with the number of internal nodes specified in the parentheses. SONG-S-small contains the minimal number of nodes necessary to match the accuracy of SDT. SONG-S-large uses the same number of internal nodes as SDT. Please notice that SONG models are trained without a distillation mechanism, and they always obtain better results than SDT without distillation.

Method	Letter	Connect4	MNIST
SDT w/o distillation (Frosst & Hinton, 2017)	78.00 (511)	78.63 (255)	94.45 (255)
SDT (Frosst & Hinton, 2017)	81.00 (511)	80.60 (255)	**96.76** (255)
SONG-S-large (ours)	**86.25** (511)	**82.82** (255)	95.74 (255)
SONG-S-small (ours)	82.95 (64)	80.27 (8)	94.66 (64)



(a) 16 internal nodes and 8 steps.



(b) 32 internal nodes and 8 steps.

Figure 5: Examples of the graph structures obtained by training SONG on the MNIST dataset. The root is the top-most node in each graph, and double node borders denote the leaves with numbers of the MNIST classes. For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

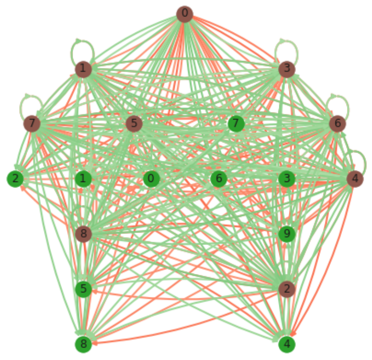
As presented in Table 1, our method matches or outperforms most of the recent state-of-the-art methods. On CIFAR10, SONG accuracy outperforms all baseline by almost 1 percentage point. On MNIST, it is worse than ANT (Tanno et al., 2019) by around 0.5%, and on CIFAR100 and TinyImagNet, NBDT (Wan et al., 2020) achieves better results. However, both ANT and NBDT are not trained in an end-to-end continuous manner. Moreover, NBDT requires a hierarchy provided before training, and without such a hierarchy, it obtains accuracy more than 1% lower than SONG on CIFAR100.

SONG as shallow model Although SONG can be successfully used in a deep learning setup, it can also be treated as a shallow model. In this case, SONG directly processes an input sample and returns the predictions passed with target labels to BCE loss. This setup is similar to the one presented in experiments on SDTs (Frosst & Hinton, 2017). Hence, we compare to SDT on all datasets considered in (Frosst & Hinton, 2017).

Table 2 shows that SONG obtains better results than SDT without distillation on all datasets. Moreover, on Letter and Connect4, SONG outperforms even SDT with distillation. We also observe that SONG requires fewer nodes than SDT and obtains on par results on the Connect4 dataset with 30 times fewer nodes. For Letter and MNIST, similarly good results can be obtained with 30 times fewer nodes. This finding is in line with (Shotton et al., 2016) which shows that decision graphs require dramatically less memory while considerably improving generalization.

SONG structure As a fully differentiable model, SONG strengthens or weakens an edge between any pair of nodes during training to constantly optimize the graph's structure (see Figure 6). Consequently, it can generate any structure that uses all available nodes, or only some of them. In particular, the final structure may be a binary tree or contain back edges. Moreover, the distance from the root to leaves can vary. This variability is visualized in Figure 5, where we present two graphs obtained for MNIST using a different number of internal nodes and steps.

In Figures 18 and 19, we provide statistics on multiple SONGs generated for the MNIST and CIFAR10, respectively. We observe a significant difference in SONG structure depending on the number of internal nodes and steps. First, we note that the number of internal nodes used by the model increases with the increasing number of steps N, and it does not depend on the total number


(a) Initial values of M^0, M^1.

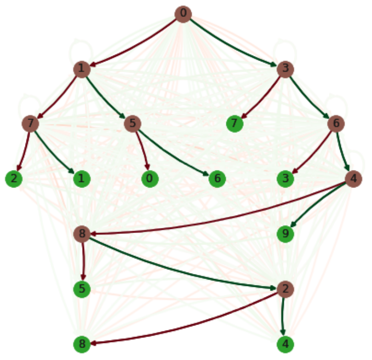

(b) Trained values of M^0, M^1.

Figure 6: Visualisation of the graphs with edges corresponding to the transition matrices M^0 (brown edges) and M^1 (green edges) of the SONG before and after training on the MNIST dataset with 9 internal nodes. One can observe that SONG models binarize the connections during gradient training. Notice that brown and green nodes correspond to internal nodes and leaves, respectively.

of internal nodes n. As a natural consequence, a similar trend is observed for the distance from the root to the leaves. When it comes to back edges, their number is relatively small, and they appear only for a larger number of steps. At the same time, the cross edges are more often and increase with the increased number of internal nodes.

SONG structure during training We analyze the relationship between BCE loss and the probability of back and cross edges in the successive epochs of the training. We present the mean over multiple models and all test samples (as each test sample x has its graph represented by matrix P_x). We observe that the probability of back edges decreases together with decreasing BCE loss, both for simple MNIST (see Figure 9) and more complicated CIFAR100 datasets (see Figure 7).

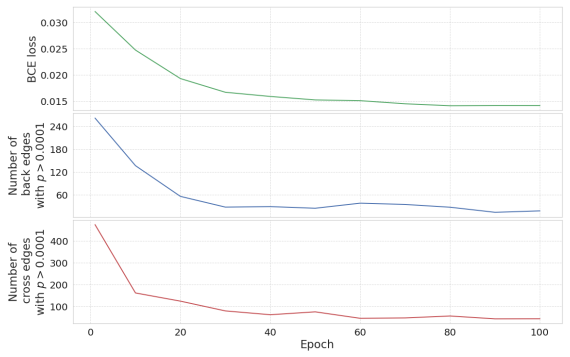


Figure 7: BCE loss as well as the number of back and cross edges in the successive training epochs of SONG with 256 internal nodes and 10 steps trained for CIFAR100. One can observe that number of back edges decrease together with decreasing BCE loss.

Broader impact This work is mostly a theoretical contribution with practical elements, and as such, does not have a direct impact on society. However, our framework is a cornerstone of neural decision graphs, which sheds new light on combining modern neural networks with explainable decision models such as graphs. Hence, due to the high applicability of tree and graph decision models in many domains, our work can bring long-term benefits outside machine learning.

6 Conclusions

In this work, we introduce Self-Organizing Neural Graphs (SONGs), a new type of decision graphs applicable in any deep learning pipeline. They optimize their structure during training by strengthening or weakening graph edges using gradient descent. Thanks to the graph structure, SONG can reuse the decision nodes and obtain state-of-the-art results with a significantly smaller number of nodes than existing methods. Moreover, the introduced general paradigm based on Markov processes allows for efficient training, and SONG converges to the interpretable binary acyclic directed graphs. Hence, we believe that our work opens a plethora of research pathways towards more effective applications of decision graphs in a deep learning setup.

REFERENCES

Karim Ahmed, Mohammad Haris Baig, and Lorenzo Torresani. Network of experts for large-scale image categorization. In *European Conference on Computer Vision*, pp. 516–532. Springer, 2016.

Stephan Alaniz, Diego Marcos, Bernt Schiele, and Zeynep Akata. Learning decision trees recurrently through communication. In *34th IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2021.

Arthur Asuncion and David Newman. Uci machine learning repository, 2007.

Seungryul Baek, Kwang In Kim, and Tae-Kyun Kim. Deep convolutional decision jungle for image classification. *arXiv preprint arXiv:1706.02003*, 2017.

Osbert Bastani, Carolyn Kim, and Hamsa Bastani. Interpreting blackbox models via model extraction. *arXiv preprint arXiv:1705.08504*, 2017.

Leonard A Breslow and David W Aha. Simplifying decision trees: A survey. *The Knowledge Engineering Review*, 12(01):1–40, 1997.

Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: deep learning for interpretable image recognition. *arXiv preprint arXiv:1806.10574*, 2018.

Philip A. Chou. Optimal partitioning for classification and regression trees. *IEEE Computer Architecture Letters*, 13(04):340–354, 1991.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Cem Keskin and Shahram Izadi. Splinenets: Continuous neural decision graphs. *arXiv preprint arXiv:1810.13118*, 2018.

Ron Kohavi and Chia-Hsin Li. Oblivious decision trees, graphs, and top-down pruning. In *IJCAI*, pp. 1071–1079. Citeseer, 1995.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pp. 1467–1475, 2015.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. *CS 231N*, 7:7, 2015.

Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database, 2010.

Wei-Yin Loh. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery*, 1(1):14–23, 2011.

Siyuan Ma, Raef Bassily, and Mikhail Belkin. The power of interpolation: Understanding the effectiveness of sgd in modern over-parametrized learning. In *International Conference on Machine Learning*, pp. 3325–3334. PMLR, 2018.

Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In *International conference on extending database technology*, pp. 18–32. Springer, 1996.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5528–5531. IEEE, 2011.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pp. 246–252. Citeseer, 2005.

Ravi Teja Mullapudi, William R Mark, Noam Shazeer, and Kayvon Fatahalian. Hydranets: Specialized dynamic architectures for efficient inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8080–8089, 2018.

Venkatesh N Murthy, Vivek Singh, Terrence Chen, R Manmatha, and Dorin Comaniciu. Deep decision network for multi-class image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2240–2248, 2016.

Meike Nauta, Ron van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition. *arXiv preprint arXiv:2012.02046*, 2020.

Arlindo L Oliveira and Alberto Sangiovanni-Vincentelli. Using the minimum description length principle to infer reduced ordered decision graphs. *Machine Learning*, 25(1):23–50, 1996.

JJ Oliver. Decision graphs - an extension of decision trees. In *Proc. 4th International Conference on Artificial Intelligence and Statistics, Miami, FL, 1993*, 1993.

Jonathan Oliver. *Decision graphs: an extension of decision trees*. Citeseer, 1992.

John C Platt, Nello Cristianini, John Shawe-Taylor, et al. Large margin dags for multiclass classification. In *nips*, volume 12, pp. 547–553, 1999.

J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

John Shafer, Rakesh Agrawal, and Manish Mehta. Sprint: A scalable parallel classifier for data mining. In *Vldb*, volume 96, pp. 544–555. Citeseer, 1996.

Jamie Shotton, Ross Girshick, Andrew Fitzgibbon, Toby Sharp, Mat Cook, Mark Finocchio, Richard Moore, Pushmeet Kohli, Antonio Criminisi, Alex Kipman, et al. Efficient human pose estimation from single depth images. *IEEE transactions on pattern analysis and machine intelligence*, 35(12): 2821–2840, 2012.

Jamie Shotton, Toby Sharp, Pushmeet Kohli, Sebastian Nowozin, John Winn, and Antonio Criminisi. Decision jungles: Compact and rich models for classification. 2016.

Alberto Suárez and James F Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.

Hiroki Sudo, Koji Nuida, and Kana Shimizu. An efficient private evaluation of a decision graph. In *International Conference on Information Security and Cryptology*, pp. 143–160. Springer, 2018.

Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, pp. 6166–6175. PMLR, 2019.

Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–18, 2018.

Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: neural-backed decision trees. *ICLR 2021*, 2020.

Quanshi Zhang, Yu Yang, Haotian Ma, and Ying Nian Wu. Interpreting cnns via decision trees. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6261–6270, 2019.

A SONGS VISUALIZATION

In this section, we provide the visualization of a SONG trained on MNIST where the nodes are represented by the learned filters and the "average" image passing through those nodes (see Figure 8). Moreover, we provide additional examples of the graph structures obtained by training SONG on the MNIST and CIFAR10 datasets (see Figures 10 and 11) together with the consecutive steps of the Markov process (see Figures 12, 13, 14, and 15).

Finally, we analyze the relationship between BCE loss and the probability of back and cross edges in the successive epochs of the training. We present the mean over multiple models and all test samples (as each test sample x has its graph represented by matrix P_x). The number of back and cross edges is obtained in the following way. We first calculate all paths from the root with a probability higher than a particular threshold 0.0001. Then we create a standard binary directed graph that contains all nodes and edges from those paths. Finally, we run the DFS algorithm for this graph (starting from the root) to obtain backed and cross edges.

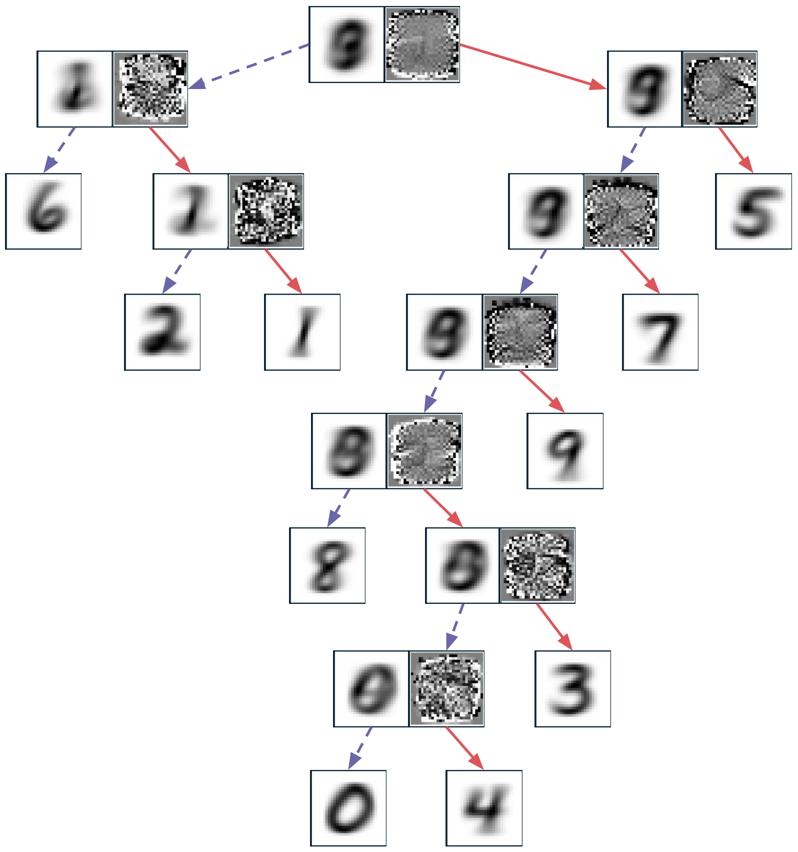


Figure 8: Visualization of a shallow SONG (SONG-S) trained on MNIST where the nodes are represented by the learned filters and the "average" image passing through those nodes (corresponding to the right and left side of each node, respectively). Notice that SONGs contain filters only in the inner nodes, as it is impossible to move out from the leaves.

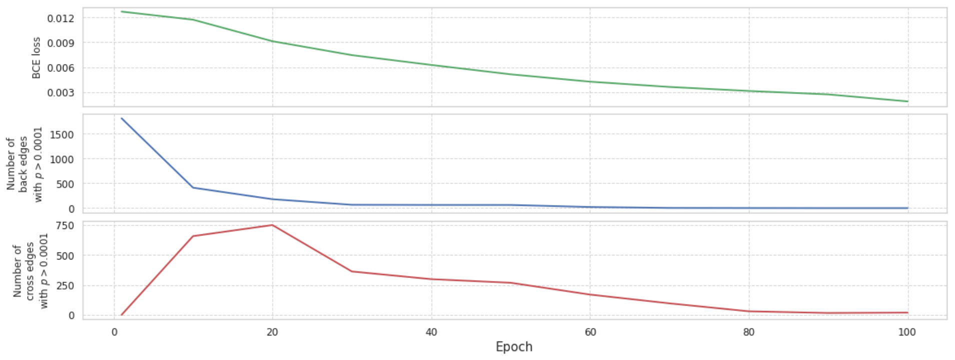


Figure 9: BCE loss as well as the number of back and cross edges in the successive training epochs of SONG with 64 internal nodes and 10 steps trained for MNIST. One can observe that number of back edges decrease together with decreasing BCE loss.

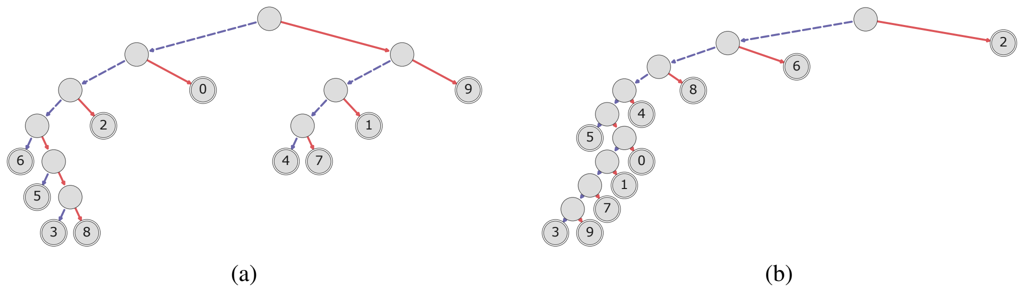


Figure 10: Examples of the graph structures obtained by training SONG on the MNIST dataset. The root is the top-most node in each graph, and the leaves are denoted by double node borders. The numbers on the leaves are the MNIST classes. For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

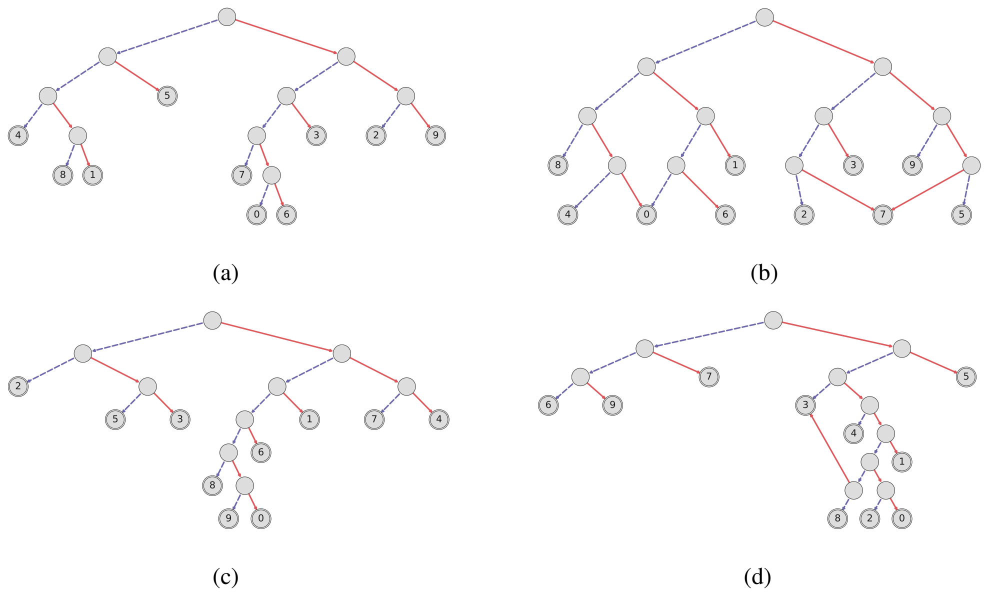


Figure 11: Examples of the graph structures obtained by training SONG on the CIFAR10 dataset. The root is the top-most node in each graph, and the leaves are denoted by double node borders. The numbers on the leaves are the CIFAR10 classes. For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

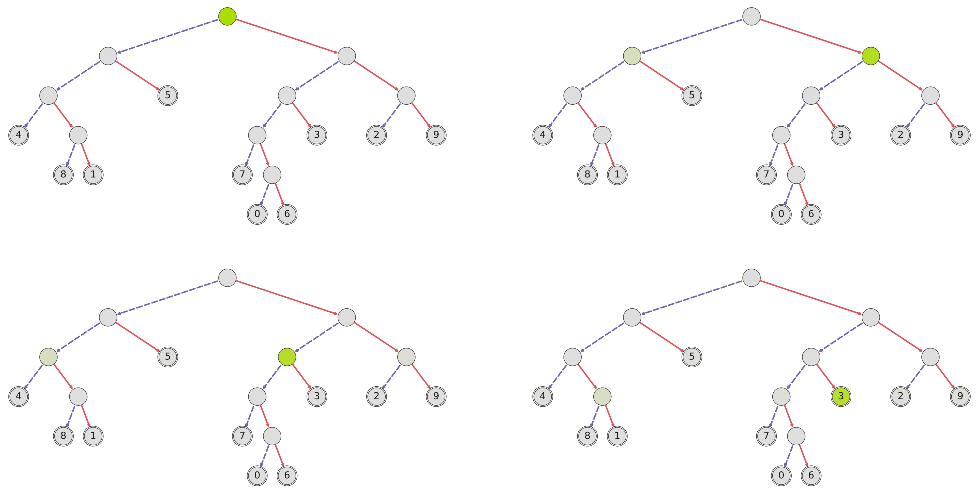


Figure 12: An input image passing through a SONG trained on CIFAR10. High saturation of the green color denotes high probability in the node. Each graph represent a consecutive step of the inference (from left to right, then top to bottom). For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

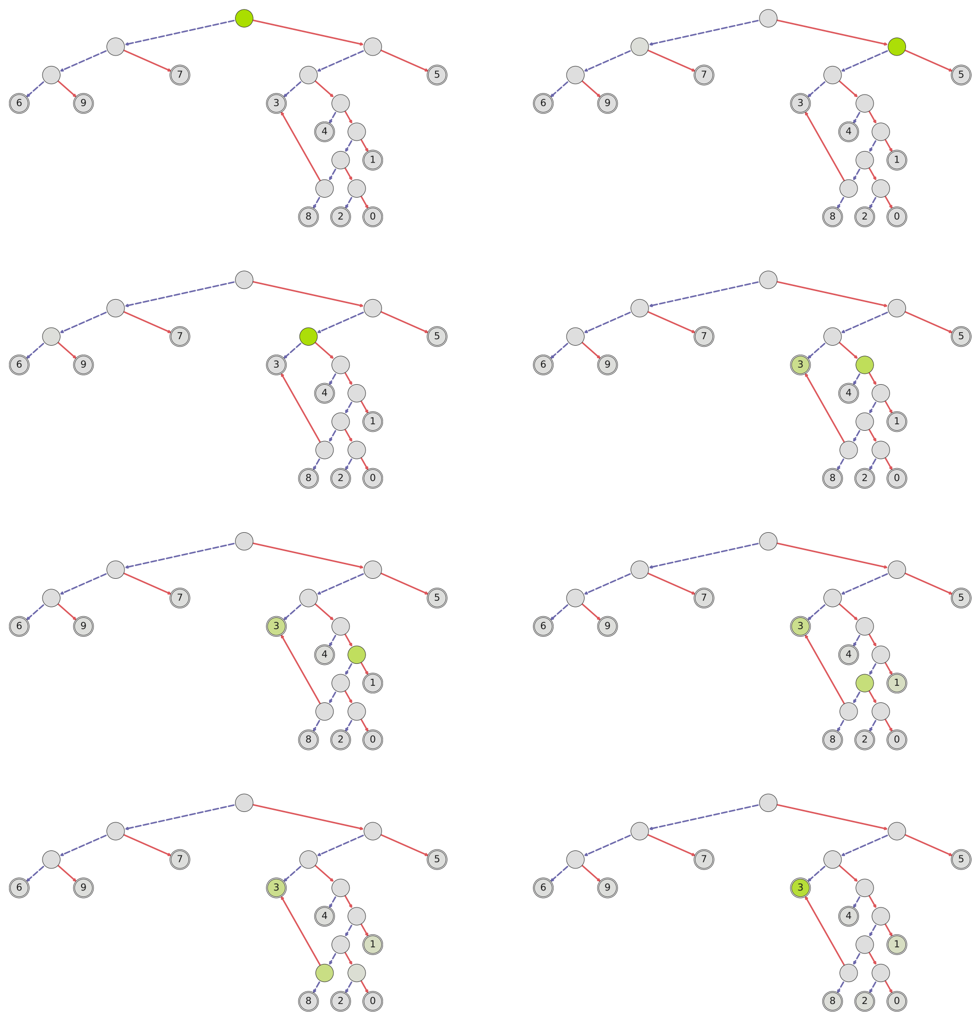


Figure 13: An input image passing through a SONG trained on CIFAR10. High saturation of the green color denotes high probability in the node. Each graph represent a consecutive step of the inference (from left to right, then top to bottom). For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

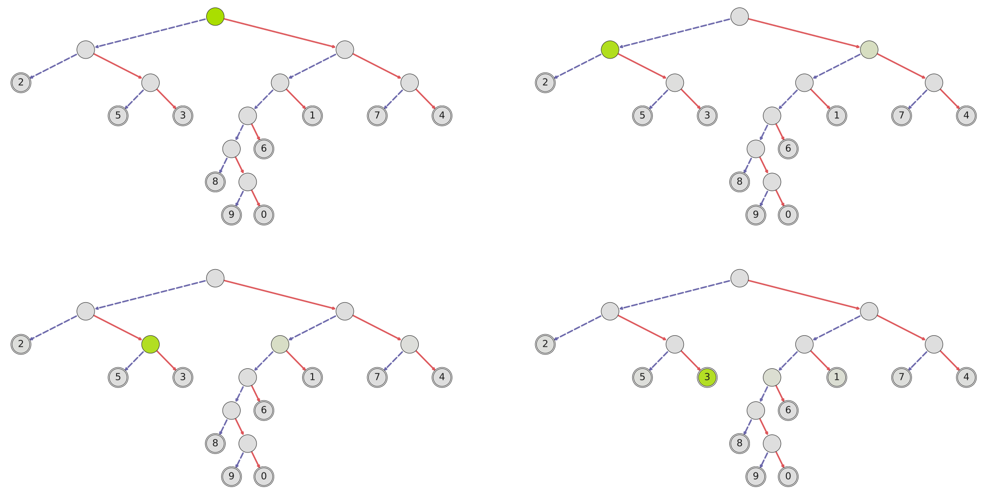


Figure 14: An input image passing through a SONG trained on CIFAR10. High saturation of the green color denotes high probability in the node. Each graph represent a consecutive step of the inference (from left to right, then top to bottom). For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

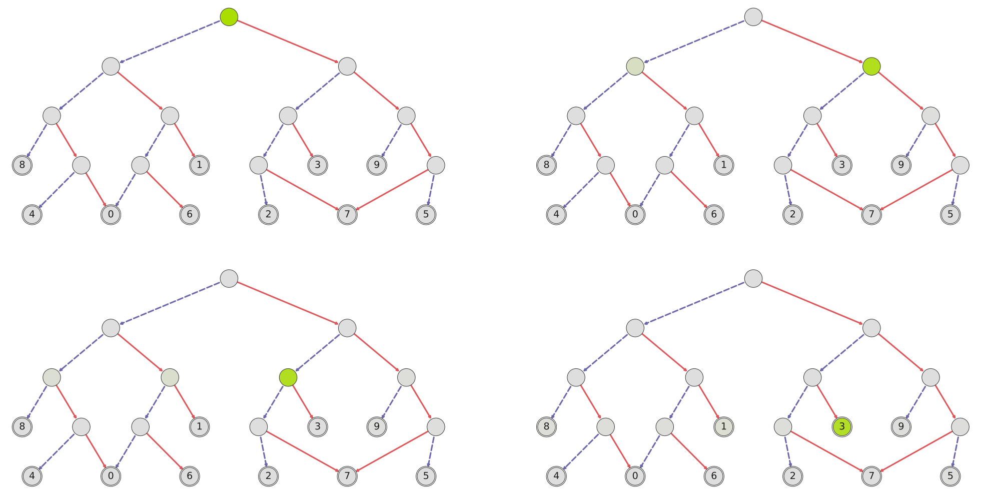


Figure 15: An input image passing through a SONG trained on CIFAR10. High saturation of the green color denotes high probability in the node. Each graph represent a consecutive step of the inference (from left to right, then top to bottom). For each node v_i, we present two edges corresponding to the highest probability from two transition vectors $m^0_{\cdot i}$ and $m^1_{\cdot i}$ (represented as dashed blue and solid red arrows, respectively).

B THEORETICAL ANALYSIS

Let us recall the most important definitions from section '*Theoretical analysis*' from the main paper. In our theoretical results, we consider SONG as the probabilistic model over trajectories. A trajectory of length N, starting at the root of SBDG G, is defined as $T = (u_{i_t})_{t=1..N}$ with binary decision $d_t \in \{0, 1\}, i_t \in I$, where I denotes the set of node indexes. Thus, our trajectory starts at the root ($i_0 = 0$) and successively passes through nodes $u_{i_{t_1}}, \ldots, u_{i_{t_N}}$. The position of trajectory after time t is defined as $T(t) = u_{i_t}$ and the probability of trajectory T is defined as

$$\mathbf{prob}(T; G) = \prod_{t=1}^{N} (\sigma_{i_{t-1}}^{d_t} \cdot m_{i_t i_{t-1}}^{d_t}).$$

Then the probability of reaching leaf l after N steps with a random trajectory T equals $\mathbf{prob}(T(N) = l \,|\, T \sim G)$, where $T \sim G$ denotes that we sample trajectories with respect to distribution given by $\mathbf{prob}(\cdot; G)$.

Next, we introduce a binarized graph G, where we binarize the connections from any pair of nodes. For a fixed $d \in \{0, 1\}$, we denote $G[i, j; d]$ as the graph that makes a decision of moving from u_i to u_j with probability 1.

In the following theorem, we show that if G has no cycles, then we can decompose the probability of its trajectory into the mixture of such binarized graphs.

Theorem B.1. *(Theorem 4.1 in the main paper) Let G be a SBDG where the probability of visiting twice an arbitrary node by a trajectory of length N is zero. Moreover, u_i be an internal node, fixed $d \in \{0, 1\}$, and an arbitrary trajectory T of length N. Then*

$$\mathbf{prob}(T; G) = \sum_{j=1}^{n} m_{ji}^{d} \mathbf{prob}(T; G[i, j, d]). \tag{5}$$

Proof. Let $T = (u_{i_t})_{t=1..N}$ be a given trajectory and let us consider three cases of passing trough node u_i. First case assumes that T does not pass through u_i, i.e. $i \neq i_t$ for $t = 1, \ldots, N$. Then, directly from the definition of the trajectory's probability

$$\mathbf{prob}(T; G) = \mathbf{prob}(T; G[i, j, d]), \text{ for an arbitrary } j.$$

This completes the proof of equation 5 in this case. Hence, let us now consider the cases where the trajectory T passes through node u_i.

Suppose the second case, when T passes through u_i more than once. In this case, we will show that both the left and right sides of equation 5 are zero. Obviously, $\mathbf{prob}(T; G) = 0$ follows directly from the assumption that the probability of visiting twice an arbitrary node by a trajectory of length N in G is zero. Assume, for an indirect proof that there exist j such that $m_{ji}^{d} \mathbf{prob}(T; G[i, j, d]) > 0$. Then $m_{ji}^{d} > 0$. Moreover, if T passes though u_i and makes a decision d, then it has to move to u_j. In consequence,

$$\mathbf{prob}(T; G) = m_{ji}^{d} \mathbf{prob}(T; G[i, j, d]) > 0$$

is a contradiction.

Let us consider the remaining, third case, when T passes through u_i only once, and makes a decision d. In other words, there exists a unique t such that $i_t = i$ and $d_t = d$. Observe that if $j = i_{t+1}$ then we move from u_i to u_j and all the probabilities $\mathbf{prob}(T; G[i, l, d]) = 0$, for $l \neq j$. Moreover, since T visits u_i only once, we get $\mathbf{prob}(T; G) = m_{ji}^{d} \mathbf{prob}(T; G[i, j, d])$, which completes the proof. □

We now show the consequences of the above theorem for the SONG model. For this purpose, we assume that $X = (x_i)_{i=1..K}$ where each x_i is associated with a label y_i. We also consider SONG $\mathcal{G}$ trained on X for trajectories of length N. Thus for each pair (x, y), we define the probability that a random trajectory of length N reaches leaf corresponding to y as $\mathbf{prob}(T(N) = y \,|T \sim \mathcal{G}_x)$.

In the following theorem, we show that if SONG is trained with zero CE or BCE loss, then no trajectory of length N in $\mathcal{G}_x$ visits the same internal node twice with nonzero probability.

Theorem B.2. *Let us consider SONG classifier with N moves and x being a data point with class y, such that* $\operatorname{loss}\big(\mathbf{prob}\boldsymbol{(}T(N) = y \,|T \sim \mathcal{G}_x\boldsymbol{)}, y\big) = 0$.

Then no trajectory of length N in $\mathcal{G}_x$ visits the same internal node twice with nonzero probability.

Proof. First observe, that directly from the fact that both CE and BCE are non-negative, $\operatorname{loss}\big(\mathbf{prob}\boldsymbol{(}T(N) = y \,|T \sim \mathcal{G}_x\boldsymbol{)}, y\big) = 0$ iff

$$\mathbf{prob}\boldsymbol{(}T(N) = y \,|T \sim \mathcal{G}_x\boldsymbol{)} = 1.$$

Now suppose that there exists a trajectory T with nonzero probability, which goes through a given internal node u twice, i.e. $T(t_1) = T(t_2) = v$ for $t_1 < t_2$. Observe that $T(t)$ is not a leaf for $t \in [t_1, t_2]$, since after reaching the leaf, we stay in it. Consider the trajectory $\tilde{T}$ given by

$$\tilde{T}(t) = \begin{cases} T(t) \text{ for } t \leq t_1, \\ T(t_1 + s) \text{ for } t = t_1 + l(t_2 - t_1) + s, l \in \mathbb{N}, s \in \{0, .., t_2 - t_1\}. \end{cases}$$

In other words, this is a trajectory that forms a cycle after reaching u. Thus we does not end in a leaf with nonzero probability, which leads to a contradiction. □

The accuracy of $\mathcal{G}$ over set X is defined as the probability of predicting the correct class

$$\operatorname{acc}(\mathcal{G}; X) = \frac{1}{K} \sum_{i=1}^{K} \mathbf{prob}\boldsymbol{(}T(N) = y_i \,|T \sim \mathcal{G}_{x_i}\boldsymbol{)}.$$

As a direct consequence of Theorem B.1, we formulate the following fact.

Theorem B.3. *(Theorem 4.2 in the main paper) Let $\mathcal{G}$ be a SONG. We assume that for every $x \in X$ no trajectory in $\mathcal{G}_x$ of length N that visits twice the same internal node with nonzero probability. Let a node index $i \in \{1, \ldots, n\}$ and $d \in \{0, 1\}$ be fixed. Then*

$$\operatorname{acc}(\mathcal{G}; X) = \sum_{j=1}^{n} m_{ji}^{d} \operatorname{acc}(\mathcal{G}[i, j, d]; X).$$

Proof. By Theorem B.1, for an arbitrary point $x \in X$ (with class y) and trajectory of length N, we have

$$\mathbf{prob}\boldsymbol{(}T; \mathcal{G}_x\boldsymbol{)} = \sum_{j=1}^{n} m_{ji}^{d} \mathbf{prob}\boldsymbol{(}T; \mathcal{G}_x[i, j, d]\boldsymbol{)}.$$

In consequence,

$$\mathbf{prob}\boldsymbol{(}T(N) = y \,|T \sim \mathcal{G}_x\boldsymbol{)} = \sum_{j=1}^{n} m_{ji}^{d} \mathbf{prob}\boldsymbol{(}T(N) = y \,|T \sim \mathcal{G}_x[i, j, d]\boldsymbol{)}.$$

Averaging the above probability over all points from X and applying the definition of accuracy, we obtain the assertion of the theorem. □

Observe that the above theorem implies that if we discretize connections in the graph by applying formula equation 6 (below), then we do not decrease the accuracy of the model (statistically, we increase it):

Theorem B.4. *Let $\mathcal{G}_x$ be SONG generated for $x \in X$ with CE or BCE loss equals zero. Moreover, let node index $i \in I$ and $d \in \{0, 1\}$ be fixed, and*

$$j = \arg\max_{\tilde{j}} \operatorname{acc}(\mathcal{G}[i, \tilde{j}, d]; X). \tag{6}$$

Then

$$\operatorname{acc}(\mathcal{G}; X) \leq \operatorname{acc}(\mathcal{G}[i, j, d]; X).$$

Proof. From Theorem B.2 we obtain that $\mathcal{G}_x$ is SONG generated for $x \in X$ with no trajectory of length N that visits twice the same point with nonzero probability. Theorem B.3 implies that if we discretize connections in the graph by applying formula equation 6, then we do not decrease the accuracy of the model. □

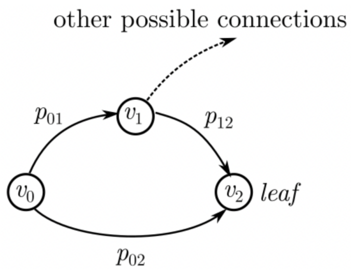


Figure 16: Simplified example of a SONG graph introduced to draw an intuition about graph binarization property.

C Explanation on graph binarization

To outline the idea why the Markov chain in our model gets binarized, let us consider a simplified version of SONG model with only one transition matrix M and no decision functions σ_i in nodes. Let us consider a simple subgraph G (Figure 16) with two non-zero paths:

$$G: \quad v_0 \overset{p_{01}}{\to} v_1 \overset{p_{12}}{\to} v_2, \quad v_0 \overset{p_{02}}{\to} v_2,$$

where v_0, v_1, v_2 are nodes, and $p_{01}, p_{12}, p_{02} \in [0,1]$ are transition probabilities, $l = v_2$ is a leaf node and $r = v_0$ is the root (i.e. the initial probability on nodes is $[v_0, v_1, v_2] = [1, 0, 0]$).

In the following, we will demonstrate why one of these two paths disappears. To show this, let us first define the output of the model for leaf l:

$$l = v_1 p_{12} + v_0 p_{02} = v_0(p_{01}p_{12} + p_{02}),$$

and calculate the gradients of the loss function $\mathcal{L}$ with respect to the transition probabilities:

$$\frac{\partial \mathcal{L}}{\partial p_{01}} = \frac{\partial \mathcal{L}}{\partial l} \cdot \frac{\partial l}{\partial p_{01}} = \frac{\partial \mathcal{L}}{\partial l} v_0 p_{12},$$

$$\frac{\partial \mathcal{L}}{\partial p_{02}} = \frac{\partial \mathcal{L}}{\partial l} \cdot \frac{\partial l}{\partial p_{02}} = \frac{\partial \mathcal{L}}{\partial l} v_0,$$

$$\frac{\partial \mathcal{L}}{\partial p_{12}} = \frac{\partial \mathcal{L}}{\partial l} \cdot \frac{\partial l}{\partial p_{12}} = \frac{\partial \mathcal{L}}{\partial l} v_0 p_{01}.$$

Now, we can consider two cases:

1. $p_{12} < 1$: Then, $\frac{\partial \mathcal{L}}{\partial p_{01}} < \frac{\partial \mathcal{L}}{\partial p_{02}}$ and because $p_{01} + p_{02} = 1$, the bigger gradient will cause p_{02} to increase and p_{01} to decrease. This will hold in the next iterations of training as long as $p_{12} < 1$, and in consequence $p_{02} \to 1$ and $p_{01} \to 0$.
2. $p_{12} = 1$: Then, the subgraph is already binarized from node v_1 onwards, and the gradients with respect to p_{01} and p_{02} are equal (the model converged). While this situation can, in theory, lead to a non-binary graph, it is very rare due to random initialization.

The above reasoning can be easily generalized to the case of multiple nodes and paths:

- If more than two alternative paths exist, both p_{01} and p_{02} can increase, but then at least one of the other edges leaving node v_0 has to decrease ($\sum_i p_{0i} = 1$). This way, the alternative paths will be eliminated one after another when other transition probabilities are already close to 0.
- If paths contain more nodes, the gradients contain the product of all transition probabilities on the path instead of one value.

Thus, the training converges when either all transition probabilities are binarized, or the product of transition probabilities on all alternative paths except for the first transition is exactly the same (the paths are equivalent).

D Regularizations

Here we provide more details about three types of regularization described in the paper.

Node regularization The node regularization is a direct adaptation of the approach proposed by Frosst & Hinton (2017). It is used to avoid getting stuck at poor solutions by encouraging each internal node to make equal use of both left and right subtrees. In our approach, this regularization encourages each internal node to make equal use of both sets of edges $\mathcal{E}^0$ and $\mathcal{E}^1$. I.e., to send half of the training samples to one direction (using M_0) and half of them to the other direction (with M_1). For this purpose, we calculate the cross entropy between the desired average distribution 0.5, 0.5 for those two sets and the actual average distribution $\alpha_{i,s}$, $\beta_{i,s}$ in node v_i at step s

$$L_{nodes} = -\frac{\lambda}{2}\sum_{i=1}^{n} \log(\alpha_{i,s}) + \log(\beta_{i,s}),$$

where

$$\alpha_{i,s} = \frac{\sum_{x\in B}(P_x^s r)_i \cdot (\sigma_i^1(x))^\gamma}{\sum_{x\in B}(P_x^s r)_i},$$

$$\beta_{i,s} = \frac{\sum_{x\in B}(P_x^s r)_i \cdot (\sigma_i^0(x))^\gamma}{\sum_{x\in B}(P_x^s r)_i},$$

B is a batch of samples used in an iteration, $\gamma \in [1, 2]$, and $(P_x^s r)_i$ corresponds to ith coordinate of vector $(P_x^s r)$. One can observe that our node regularizer is calculated per node and step, and it is different from (Frosst & Hinton, 2017), where additional loss is computed once for each node. Moreover, we penalize model for making uncertain decisions ($\sigma_{i,s}(x) \approx 0.5$) using the parameter γ.

Leave regularization The leave regularization, enforcing the summary probabilities in leaves to be close to 1, is defined as

$$L_{leaves} = -\log\left(\sum_{i=n}^{n+c}(P_x^N r)_i\right), \qquad (7)$$

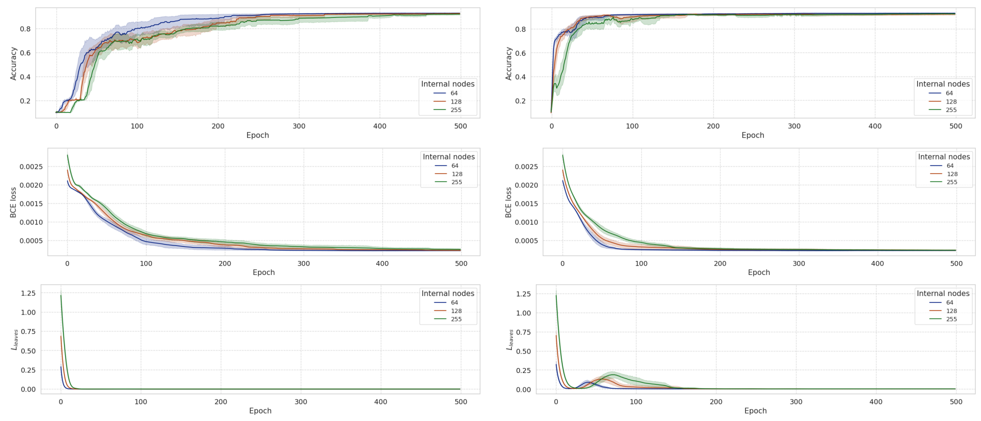
where n is the number of nodes (excluding root indexed with 0), c is the number of leaves (classes), and N is the number of steps.

In Figure 17, we present a comparison between SONG trained on MNIST dataset with (a) and without L_{leaves} regularization (b). The accuracy and BCE loss reported at the final stage of training are similar for both models. However, there are significant differences between their convergence times. Most interestingly, models with regularization hold L_{leaves} close to 0 during the whole training, so the sum of probability in the leaves is close to 1 all the time. On the other hand, the models without regularization have an increased value of L_{leaves} between 50 and 150 epoch, meaning that the leaves are not reached for some of the input samples. Such behavior can be especially detrimental for larger datasets that require more training epochs to converge.

Gumbel-softmax We use Gumbel-softmax (Jang et al., 2016) instead of softmax to each column of matrices M_0 and M_1 to explore the trajectories of the graph better. In other words, Gumbel-softmax introduces randomness, which results in a wider exploration of the graph structure in the optimization process.

E Nodes and edges statistics

Here, we show the nodes and edges statistics calculated for SONGs trained on the MNIST and CIFAR10 dataset (see Figures 18 and 19, respectively). They are discussed in Section 5.



(a) SONG trained with L_{leaves} regularization. (b) SONG trained without L_{leaves} regularization.

Figure 17: Accuracy, BCE loss, and L_{leave} in the successive training epochs of SONG trained on the MNIST dataset. Each color represents a different number of internal nodes (64, 128, 255), and each line corresponds to mean and standard deviation over multiple training repetitions.

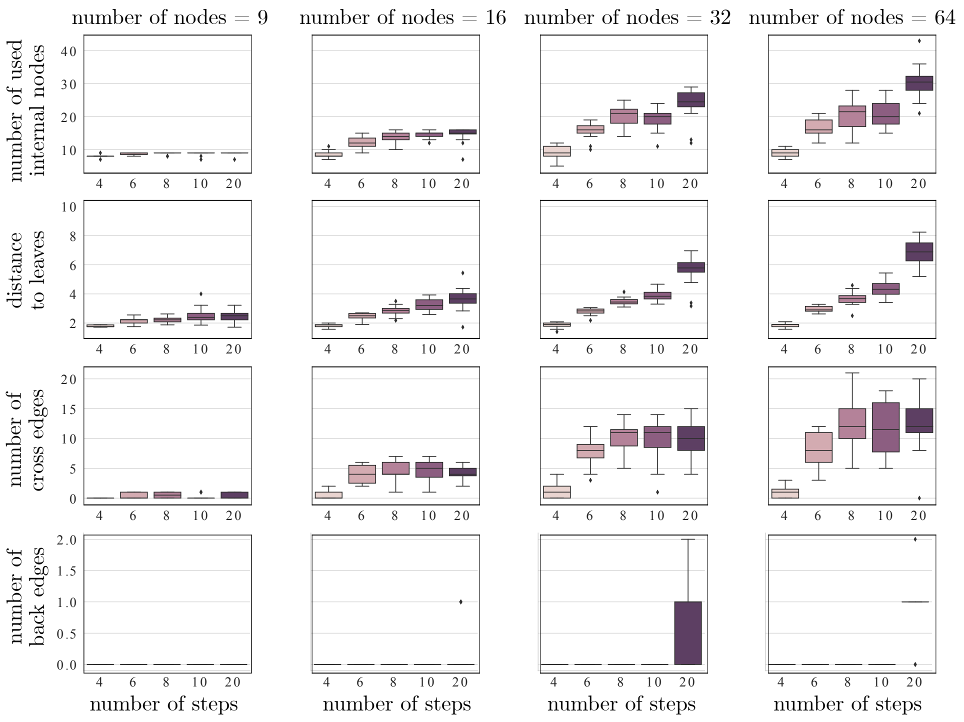


Figure 18: Nodes and edges statistics calculated for SONGs trained on the MNIST dataset. For each combination of the number of internal nodes and steps, 20 graphs are trained and used to plot the distributions of four statistics. One can observe a significant difference in SONG structure depending on those hyperparameters.

F TRANSITION MATRICES

In Figures 20-25, we present sample matrices M^0 and M^1 before and after training. One can observe that at the beginning, there are weak connections between all nodes. However, trained matrices are almost binary and usually contain one value close to 1 in each column, and all other values are close to 0. Moreover, in Figure 26, we present the mean distances between transition matrices (P_x) obtained for samples of the same and different classes. One can observe, among others, that the diagonal is visibly darker than the rest of the matrix, which means that inputs from the same class

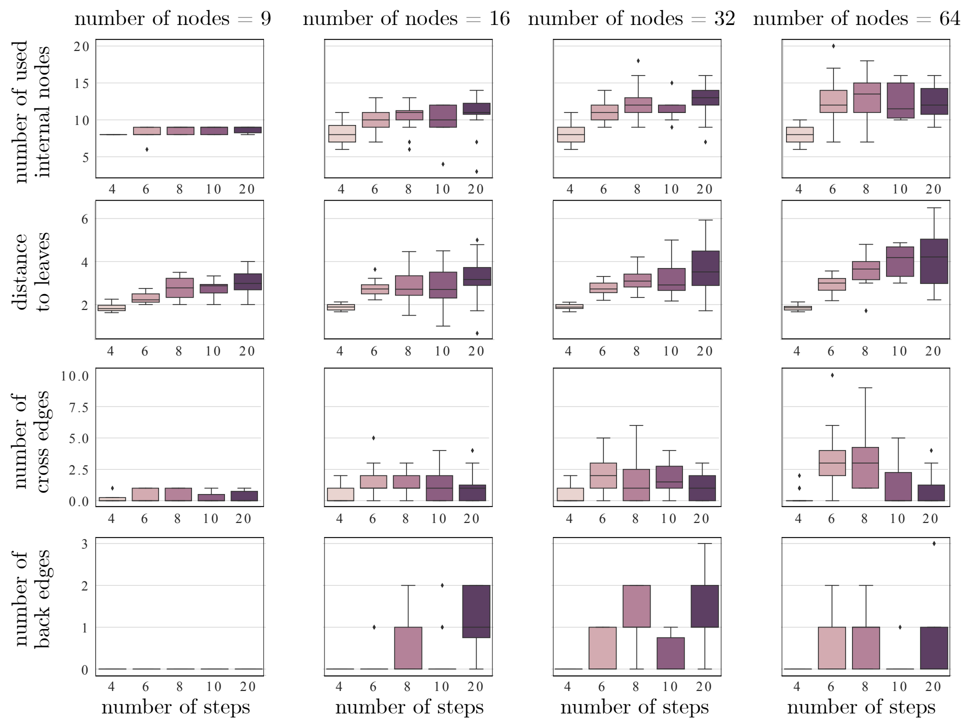
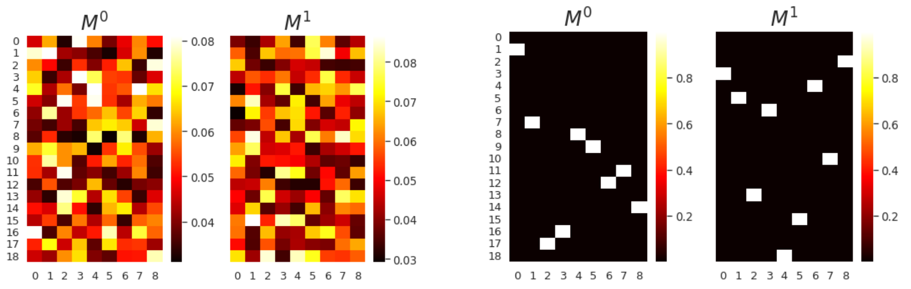


Figure 19: Nodes and edges statistics calculated for SONGs trained on the CIFAR10 dataset. For each combination of the number of internal nodes and steps, 20 graphs are trained and used to plot the distributions of four statistics.



(a) Initial values of M^0, M^1. (b) Trained values of M^0, M^1.

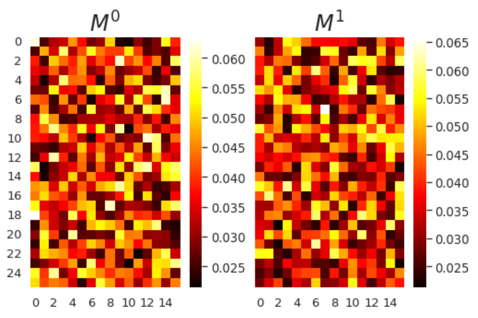
Figure 20: Sample matrices M^0 and M^1 of the SONG before and after training on the MNIST dataset with 9 internal nodes. One can observe that SONG models binarize the connections during gradient training. Graphs corresponding to presented transition matrices are visualized in Figure 6.

have more similar transition matrices. This confirms that we obtain similar transition matrices for similar inputs.

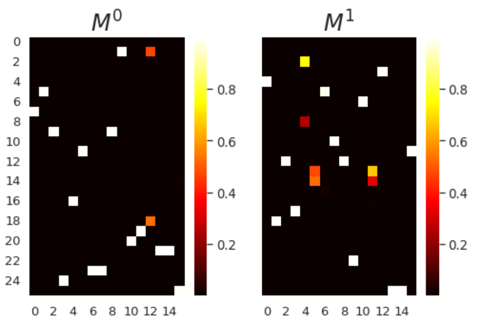
G Experimental setup

We used the following datasets in our experiments:

- Letter (`https://archive.ics.uci.edu/ml/datasets/Letter+Recognition`),
- Connect4 (`http://archive.ics.uci.edu/ml/datasets/connect-4`),
- MNIST (published under CC BY-SA 3.0 license),
- CIFAR 10 & CIFAR 100 (published under MIT license),

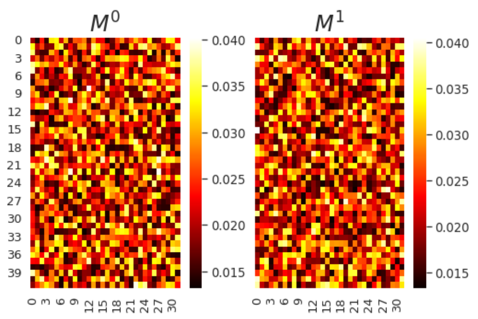


(a) Initial values of M^0, M^1.

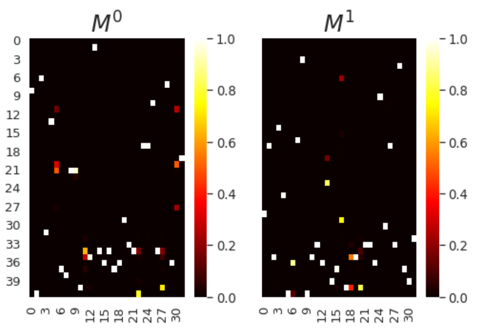


(b) Trained values of M^0, M^1.

Figure 21: Sample matrices M^0 and M^1 of the SONG before and after training on the MNIST dataset with 16 internal nodes.

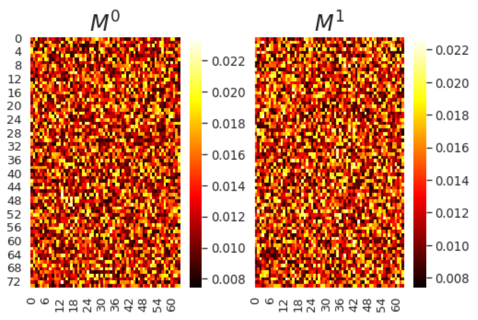


(a) Initial values of M^0, M^1.

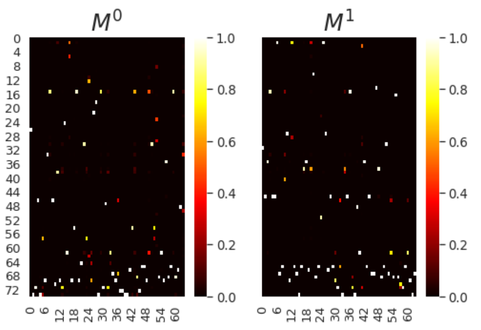


(b) Trained values of M^0, M^1.

Figure 22: Sample matrices M^0 and M^1 of the SONG before and after training on the MNIST dataset with 32 internal nodes.

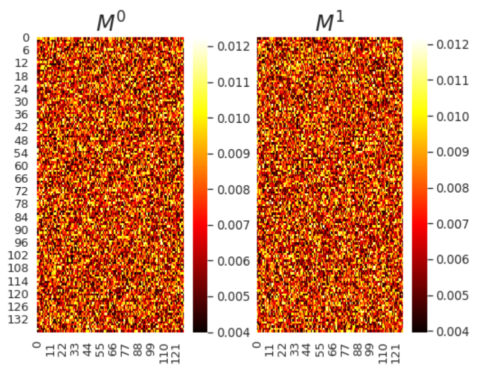


(a) Initial values of M^0, M^1.

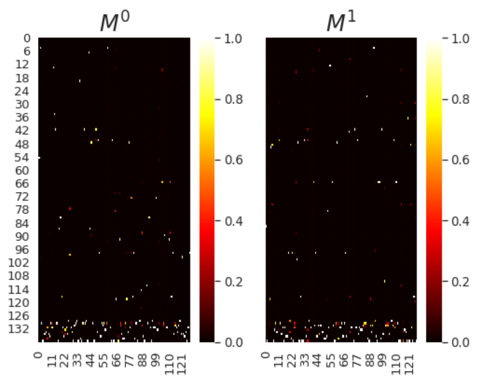


(b) Trained values of M^0, M^1.

Figure 23: Sample matrices M^0 and M^1 of the SONG before and after training on the MNIST dataset with 64 internal nodes.

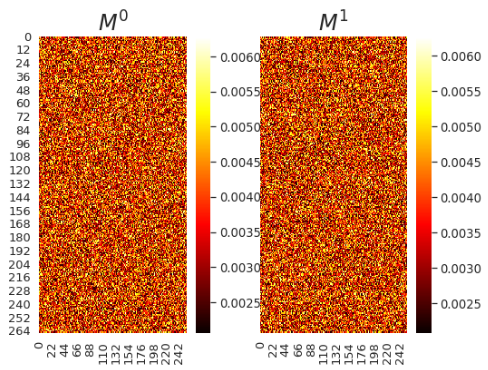


(a) Initial values of M^0, M^1.

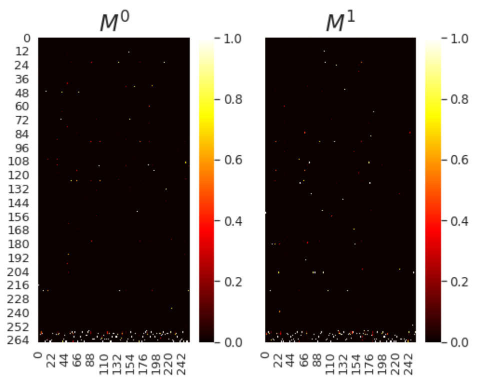


(b) Trained values of M^0, M^1.

Figure 24: Sample matrices M^0 and M^1 of the SONG before and after training on the MNIST dataset with 128 internal nodes.

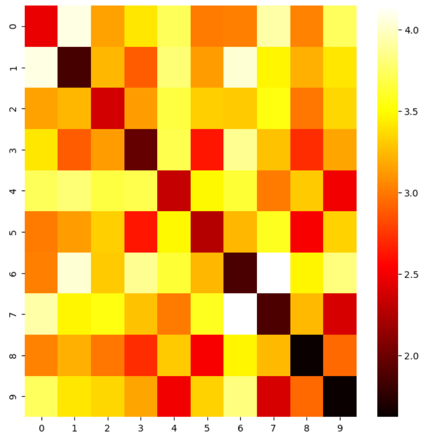


(a) Initial values of M^0, M^1.

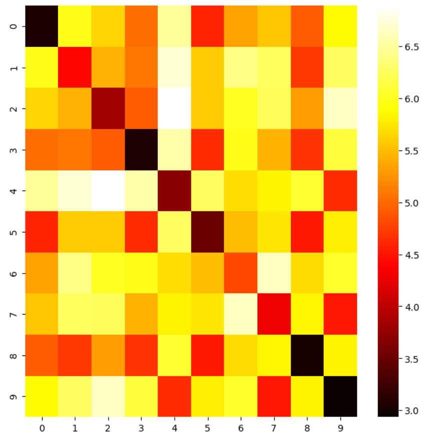


(b) Trained values of M^0, M^1.

Figure 25: Sample matrices M^0 and M^1 of the SONG before and after training on the MNIST dataset with 256 internal nodes.



(a) SONG trained with 16 internal nodes.



(b) SONG trained with 64 internal nodes.

Figure 26: Mean distances between transition matrices P_x for pairs of MNIST input samples represented by a distance matrix (the larger distance, the brighter color). The rows and columns correspond to 0-9 digits.

- TinyImageNet (`https://www.kaggle.com/c/tiny-imagenet/data`).

Moreover, we consider two types of setups, deep (SONG) and shallow (SONG-S). In SONG, we build neural networks that contain two successive parts, CNN and a graph. For the MNIST dataset, the CNN is built from two convolution layers with 8 and 16 filters of size 5×5, each followed by ReLU and 2×2 max pooling. Finally, a linear layer returns representation vectors of dimension 50. For other datasets (CIFAR10, CIFAR100, and TinyImageNet), we use model ResNet18 without the last linear layer. At the same time, for SONG-S, we only flatten the input sample to a one-dimensional vector.

For SONGs, we apply a similar experimental setup as in the state-of-the-art methods (Wan et al., 2020) to have comparable results. More precisely, we take the previously trained ResNet18 network, remove its last layer, and use the remaining part as a CNN part. For the MNIST data, we train the first part directly using Binary Cross Entropy (BCE) loss. For the remaining datasets, we take a model from `github.com/alvinwan/neural-backed-decision-trees` (published under MIT license) trained with Cross Entropy (CE) and finetune it using BCE loss. During training the SONG, weights of CNNs are frozen. Moreover, the following hyper-parameters are considered in the grid-search:

- For MNIST and CIFAR10:
  - the number of nodes: 9, 16, 32, 64,
  - the number of steps: 4, 6, 8, 10, 20.
- For CIFAR100:
  - the number of nodes: 99, 256, 512,
  - the number of steps: 7, 12, 20, 40.
- For TinyImagenet200:
  - the number of nodes: 512,
  - the number of steps: 20, 40.

Additionally, we consider a batch size 64 or 128 and the learning rate 0.001 for all datasets. Finally, when it comes to initialization, M^0, M^1, and biases in nodes are initialized from a uniform distribution on the interval $[0, 1]$, and the remaining parameters (filters in the nodes) use the Kaiming initialization.

For SONG-S, the following hyper-parameters are considered in the grid-search:

- For Letter dataset:
  - the number of nodes: 25, 32, 64, 128, 511,
  - the number of steps: 5, 10, 20, 30, 40, 50.
- For Connect4 dataset:
  - the number of nodes: 2, 8, 16, 32, 255,
  - the number of steps: 2, 5, 10.
- For MNIST dataset:
  - the number of nodes: 9, 16, 32, 64, 128, 256,
  - the number of steps: 4, 6, 8, 10, 20, 30, 40, 50.

The remaining hyper-parameters are similar to the SONG setup.

H Detailed results

In this section, we provide details on the experiments conducted for SONG in a deep learning setup (see Table 3 and 4) and when treating SONG as a shallow model (see Table 5).

Table 3: Results of SONG in a deep learning setup. One can observe that for the MNIST dataset (a), the performance increases with the increasing number of nodes and steps. In contrast to CIFAR10 (b), where the performance is relatively similar for all combinations of the parameters. It can be caused by the smaller dimension of the representation vector in MNIST (50) than in CIFAR10 (512).

(a) MNIST.

nodes \ steps	4	6	8	10	20
9	95.66	97.29	97.25	97.95	97.56
16	97.31	97.83	98.23	98.43	98.56
32	96.82	97.74	98.35	98.65	98.62
64	96.29	98.12	98.12	98.47	98.68

(b) CIFAR10.

nodes \ steps	4	6	8	10	20
9	94.48	94.86	94.92	94.94	94.93
16	94.88	94.95	94.86	94.87	94.89
32	94.99	94.95	94.95	94.90	94.98
64	94.90	94.87	94.88	94.94	94.93

Table 4: Results obtained for selected models from Table 3 ("base") and their finetuned versions. We analyze two types of finetuning, either by using basis weights and finetune all the parameters of the network ("finetune") or by taking the graph structure from the base model, reset other network parameters, and train the network from scratch ("reset"). One can observe that there is no obvious winning strategy, and it should be considered a hyperparameter. Notice also that we bold the performance reported in the main paper.

(a) MNIST.

nodes	steps	base	finetune	reset
9	10	97.95	98.43	98.67
16	8	98.23	**98.81**	98.66
32	8	98.35	98.61	98.81
32	10	98.65	98.52	98.71
64	20	98.68	98.63	98.72

(b) CIFAR10.

nodes	steps	base	finetune	reset
9	10	94.94	94.98	95.26
16	6	94.95	95.09	95.47
32	6	94.95	95.12	**95.62**
64	10	94.94	95.03	95.41

I SOURCE CODE

The training and evaluation code is available in the archive file '`song_source.zip`'. It contains all the files required to reproduce the main results presented in the paper. Furthermore, in the '`README.md`' file, we describe detailed information about the Python environment, the required packages.

J COMPUTATION TIME AND RESOURCES

We have run our experiments on Nvidia V100 32GB GPUs of our internal cluster. For deep setup, we trained 50, 50, 25, and 10 models for MNIST, CIFAR10, CIFAR100, and TinyImageNet, respectively. Each model required around 2, 2, 6, and 10 hours, respectively. For the shallow setup, we trained 60, 30, and 96 models for Letter, Connect4, and MNIST, respectively. In this case, each model required around 5, 2, and 2 hours, respectively.

Table 5: SONG as a shallow model (SONG-S). One can observe that the performance increases with the increasing number of nodes and steps for all datasets. We bold the performance reported in the main paper.

(a) Letter.

nodes \ steps	5	10	20	30	40	50
25	52.65	63.45	62.90	63.85	67.65	68.55
32	53.65	62.65	72.90	73.30	73.20	73.55
64	57.95	74.00	78.70	79.70	82.95	82.95
128	57.00	73.85	79.60	83.05	84.45	85.75
511	48.75	72.35	81.60	82.50	84.05	**86.25**

(b) Connect4.

nodes \ steps	2	5	10
2	77.47	77.40	77.50
8	75.37	79.60	80.27
16	75.47	80.31	81.55
32	75.36	80.45	82.65
255	75.43	80.43	**82.82**

(c) MNIST.

nodes \ steps	4	6	8	10	20	30	40	50
9	87.58	88.68	88.52	88.93	89.36	90.48	90.36	90.40
16	90.74	91.73	93.06	93.09	93.42	92.97	93.39	93.37
32	88.80	91.47	93.22	93.56	94.38	93.67	93.72	93.56
64	86.35	92.77	93.33	93.41	94.66	94.29	94.86	94.55
128	90.10	93.11	93.65	94.15	94.58	94.80	94.99	94.97
255	90.05	93.11	93.80	93.88	94.28	94.75	95.43	**95.74**