SELF-ORGANIZING ARTIFICIAL PERCEPTRON: BACKPROPAGATING THROUGH THE WIDTH AND DEPTH OF NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In machine learning, the hypotheses' space of neural networks is encoded by their architecture, and it cannot change during training in response to the task. As discrete architectural changes are not differentiable, neural networks are fundamentally unable to self-organize their knowledge and computational structure. This work introduces the Self-Organizing Artificial Perceptron (SOAP), a multi-layer perceptron (MLP) that can jointly learn its architecture, namely depth, width, and the associated parameters via standard backpropagation. Crucially, we do not impose upper bounds on depth or width, meaning the network is formally infinite but practically finite, in the style of Bayesian models. The underlying idea is to make the addition and removal of new layers and neurons compatible with the gradient signal. We test the joint learning of width and depth on 6 tabular datasets, showing SOAP maintains or exceeds the performance of MLPs. In addition, we analyze the behavior of depth and width in response to different batch sizes and starting conditions. Our contribution paves the way for new investigations about a new class of model whose inductive bias adapts to the task's complexity.

1 INTRODUCTION

Neural networks are one of the cornerstones of modern artificial intelligence. Their ability to learn complex, high-dimensional input-output relationships has led to remarkable progress across a wide range of domains, including natural language processing (Vaswani et al., 2017), computer vision (He et al., 2016), speech recognition (Graves & Jaitly, 2014), audio synthesis (van den Oord et al., 2016), and multimodal generation (Ramesh et al., 2021). From language models to high-resolution image generators, many of the most successful AI systems today rely on neural architectures and their associated inductive biases.

Training a neural network, however, requires designing its architecture in advance, and this step is far from being a minor implementation detail. The architecture of a neural network defines the computational graph through which information flows, gradients propagate, and representations are progressively transformed. The network's depth determines how many successive transformations can be composed—influencing the degree of abstraction at which information is represented—while its width controls the capacity of the intermediate representations. Together, they impose a strong constraint on the space of hypotheses the network can learn.

Model selection loops, hyperparameter optimization, and neural architecture search can partially automate the choice of architecture. In every scenario, however, the architecture search space is either bounded a priori (Liu et al., 2019a) or is not directly differentiable with respect to the training objective Wistuba et al. (2019). As a result, classic architectures *cannot be learned during backpropagation*, for instance by allocating neurons, removing unnecessary ones, and deepening or simplifying the computation according to the task. This limitation has forced researchers to rely on increasingly sophisticated frameworks for finding architectures that are suitable for the task prior to, rather than during, learning (Zoph & Le, 2017; Mitchell et al., 2024).

Recently, this long-standing assumption has begun to be challenged. Adaptive-depth (Nazaret & Blei, 2022) and adaptive-width models (Errica et al., 2026) have introduced separate but related mechanisms to learn structural properties of neural networks via backpropagation. These approaches

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

enable unbounded, differentiable growth and shrinkage of depth or width by introducing specific architectural mechanisms compatible with gradient-based optimization. In spirit, they recall earlier heuristic-based constructive approaches (Fahlman & Lebiere, 1989; Micheli, 2009; Liu et al., 2019b; Wu et al., 2020), where networks were expanded or pruned according to hand-designed criteria. Their key difference, however, is that architectural adaptation is no longer driven by external heuristics, rather it requires *no upper bounds and it is integrated into the learning process*.

A natural question is therefore what happens when width and depth are no longer treated as independent design choices, but are learned jointly by the same model. The relationship between these two dimensions has long been studied in machine learning. Empirical investigations have explored families of architectures with varying width-depth trade-offs (Nguyen et al., 2021); golden-ratio-inspired design rules have been proposed for convolutional networks (Tan & Le, 2019); and theoretical analyses have examined the role of depth and width through the lens of function compositionality (Mhaskar et al., 2017). Yet these studies typically compare fixed architectures selected *before* training: they do not address the behavior of a single network whose width and depth are both adaptive, nor do they investigate whether joint architectural learning introduces new optimization instabilities, degenerate solutions, or unexpected interactions between the two forms of adaptation.

In this work, we study this problem by introducing the Self-Organizing Artificial Perceptron (SOAP), a new family of multi-layer perceptrons (MLP) capable of jointly learning their width and depth while solving a task. By allowing both neurons and layers to emerge or disappear during training, the Self-Organizing Artificial Perceptron provides a concrete framework for studying the learned interaction between width and depth and the stability of their joint optimization alongside the model's parameters. Across six benchmark datasets, we compare SOAP predictive performance and learned topologies against fixed baselines. We conduct an in-depth stability analysis to understand the convergence behavior of the learned topology with respect to hyper-parameters and initial conditions. Our hope is to take a step toward neural networks that can self-organize in response to the task.

2 Related Work

Determining optimal neural architectures—specifically regarding layers' widths and network's depth—has historically leveraged methods that eventually produce fixed architectures unable to further adapt. These methodologies can be broadly categorized into Neural Architecture Search (NAS), constructive heuristics, and model compression approaches.

Neural Architecture Search (NAS) automates the design of neural networks, aiming to replace manual hyperparameter tuning (Wistuba et al., 2019). Broadly, NAS operates through three main components: a *bounded* search space, a search strategy, and a performance estimation technique. The search space defines the set of all possible operations (e.g., layer types and counts, or connectivity patterns) that the algorithm can combine. The search strategy—which may employ Reinforcement Learning (Zoph & Le, 2017), Evolutionary Algorithms (Real et al., 2019), or differentiable approximations (Liu et al., 2019a)—proposes candidate architectures from this space for evaluation. Finally, performance estimation determines the *fitness* of these candidates, typically by training the proposed models and measuring their accuracy on a validation set. NAS searches and returns an architecture that is inherently static and does not have the ability to continue adapting itself. *Hence, NAS is related in spirit but orthogonal to our work, which learns architectures without imposing upper bounds.*

Constructive methods, instead, modify the network's topology through explicit structural rules, including prototype and local-model allocation (Fritzke, 1994; Schaal & Atkeson, 1998). Classic examples include the cascade correlation algorithm, which adds hidden units to minimize residual error (Fahlman & Lebiere, 1989), and firefly descent, which grows networks based on a specific loss (Wu et al., 2020). The cascade correlation algorithm was also extended to structured domains with the Neural Network for Graphs (NN4G) (Micheli, 2009), which utilizes a constructive feedforward architecture to learn a graph transduction. Other approaches utilize natural gradients (Mitchell et al., 2024) or splitting strategies for lifelong learning (Yoon et al., 2018; Liu et al., 2019b). Typically, most constructive approaches either grow or shrink the network, *but none are able to do both while being purely driven by the task's gradient signal.*

Model compression techniques, such as pruning (Blalock et al., 2020) and Knowledge Distillation (KD) (Hinton et al., 2015), try to reduce the resources needed to perform inference. Pruning reduces

network size by removing connections or neurons (Mishra et al., 2021), using criteria ranging from classical second-order saliency (LeCun et al., 1989; Hassibi & Stork, 1992) to dynamic masks learned during training (He et al., 2018). Knowledge distillation compresses a larger *teacher* model into a smaller *student* by extracting useful fine-grained interclass relations from the teacher's logits, exploiting these soft labels to improve downstream performance compared to relying solely on the hard labels provided by supervised training data. *Crucially, these methods are only subtractive or require pre-trained models, and require heuristics at training time* (Valerio et al., 2022).

3 Self-Organizing Artificial Perceptron (SOAP)

The following sections describe how to build a self-organizing artificial perceptron that jointly learns its width and depth with standard back-propagation. To do so, Section 3.1 first introduces the reader to the concept of unbounded networks, then Section Section 3.2 introduces the associated probabilistic graphical model from which SOAP is derived.

3.1 From Bounded to Unbounded Networks

Traditional multi-layer perceptrons operate by applying successive transformations

$$g_\ell : \mathbb{R}^{D_{\ell-1}} \to \mathbb{R}^{D_\ell} \quad \text{for } \ell = 1, \dots L, \tag{1}$$

from some input state $\mathbf{h}_{\ell-1} \in \mathbb{R}^{D_{\ell-1}}$ to some output state $\mathbf{h}_\ell \in \mathbb{R}^{D_\ell}$, with the input data being $\mathbf{x} = \mathbf{h}_0$. The last hidden state $\mathbf{h}_L$ is eventually fed into a readout function o_L. Here, g_ℓ refers to classic transformations of the form $\sigma(W_\ell \mathbf{h}_\ell + \mathbf{b}_\ell)$, with finite D_ℓ, activation function σ, and weights $W_\ell, \mathbf{b}_\ell$. Therefore, the term D_ℓ stands for the width of layer ℓ, while L represents the depth of the network. In this work, we consider a *multi-readout* MLP, where layer-specific readouts are combined to produce the output $\mathbf{y}$, as illustrated by the schematics in Figure 1b. Formally:

$$\mathbf{y}_\ell = \Omega_\ell(\mathbf{x}) = o_\ell \circ g_\ell \circ g_{\ell-1} \circ \cdots \circ g_2 \circ g_1(\mathbf{x}),\ \ell \in \{1, \dots, L\} \tag{2}$$

$$\mathbf{y} = \Omega(\mathbf{x}) = o\left(\Omega_1(\mathbf{x}), \Omega_2(\mathbf{x}), \dots, \Omega_L(\mathbf{x})\right). \tag{3}$$

Note that intermediate computations are reused by deeper transformations.

In SOAP, we shall deal with $L = \infty$ and $D_\ell = \infty\ \forall \ell$, but we introduce strategies **to learn how to effectively truncate** the model's depth and width to **finite** values, building on results from Nazaret & Blei (2022); Errica et al. (2026). This truncation, as we will see, depends on learnable parameters, $\boldsymbol{\alpha}$ for depth and $\boldsymbol{\nu}$ for width, so that *discrete architectural changes are driven by the same gradient information used in back-propagation*.

As a result, the classic neural network training loop requires minimal changes, as Algorithm 1 illustrates. Section 3.4 shows how to perform inference and update SOAP's structure.

Algorithm 1 High-level overview of SOAP training.

Require: Batch $\mathcal{B} \sim \mathcal{D}$
Require: Initialized SOAP model $\mathcal{M}$ with learnable depth $\boldsymbol{\alpha}$ and width $\boldsymbol{\nu}$ parameters
function TRAINING-STEP($\mathcal{D}$, $\mathcal{B}$)
    $\hat{\mathbf{y}} \leftarrow \mathcal{M}$.forward($\mathcal{B}$)
    $\mathcal{L} \leftarrow$ LOSS($\mathcal{M}, \mathcal{B}, \hat{\mathbf{y}}$)
    $\mathcal{M}, \boldsymbol{\alpha}, \boldsymbol{\nu} \leftarrow$ BACKPROPAGATION($\mathcal{M}, \mathcal{L}$)
    $\mathcal{M} \leftarrow$ UPDATE-STRUCTURE($\mathcal{M}, \boldsymbol{\alpha}, \boldsymbol{\nu}$)
end function
function UPDATE-STRUCTURE($\mathcal{M}, \boldsymbol{\alpha}, \boldsymbol{\nu}$)
    $\mathcal{M} \leftarrow$ UPDATE-DEPTH($\mathcal{M}, \boldsymbol{\alpha}$)
    $\mathcal{M} \leftarrow$ UPDATE-WIDTH($\mathcal{M}, \boldsymbol{\nu}$)
    return $\mathcal{M}$
end function

3.2 Framing Unbounded Networks as a Probabilistic Model

SOAP's architecture, though not very different from a classic multi-readout MLP, emerges from a probabilistic framework for learning under unbounded depth and width. Hence, before moving to the practical implementation, we first formalize the generative model and discuss a variational inference treatment that makes maximum likelihood estimation tractable and optimizable via gradient-ascent.

We define latent variables $\boldsymbol{\lambda} = \{\lambda_\ell\}_{\ell \to \infty},\ \boldsymbol{\lambda}^o = \{\lambda^o_\ell\}_{\ell \to \infty}$ that control the truncated width of the network's hidden layers and those of the layer-specific readouts o_ℓ, respectively. For each neuron n at layer ℓ, $\boldsymbol{\theta}_{\ell n}$ denotes the multivariate latent variable of incoming weights (and biases), and we use $\boldsymbol{\theta}_\ell = \{\boldsymbol{\theta}_{\ell n}\}_{n \to \infty}$ to refer to the set of weights of a specific layer. Analogously, $\boldsymbol{\theta}^o_{\ell n}$ and $\boldsymbol{\theta}^o_\ell$ denote the corresponding quantities for the readouts. The complete collections of hidden-layer and readout

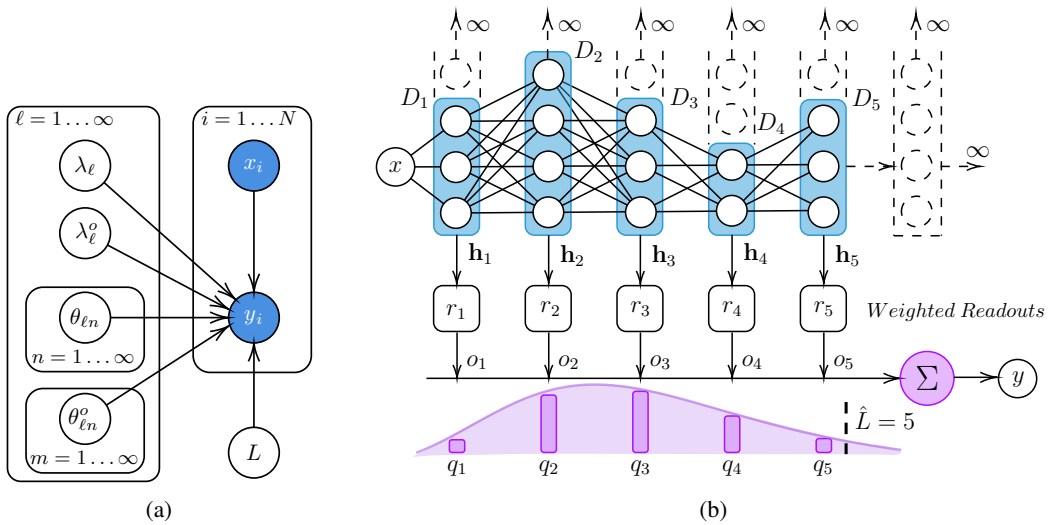


Figure 1: Probabilistic graphical model (a) and high-level diagram for SOAP (b). The latent variable L determines the *finite depth* $\hat{L}$ of the network, and each layer ℓ is governed by its own hidden width latent λ_ℓ and readout width latent λ_ℓ^o with associated weight latents $\boldsymbol{\theta}_\ell$ and $\boldsymbol{\theta}_\ell^o$. The output y is the sum of layer-specific outputs $o_\ell * q_\ell$, where q_ℓ is the learned importance of a layer.

weight variables are therefore $\boldsymbol{\theta} = \{\boldsymbol{\theta}_\ell\}_{\ell \to \infty}$ and $\boldsymbol{\theta}^o = \{\boldsymbol{\theta}_\ell^o\}_{\ell \to \infty}$, respectively. With slight abuse of notation with respect to Section 3.1, a latent variable L governs the truncated depth of the network.

The graphical model of Figure 1a defines the conditional joint distribution over the observed data sample $(\mathbf{x}_i, \mathbf{y}_i)$ and the set of latent variables we have just introduced:

$$p(\mathbf{y}_i, \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L \mid \mathbf{x}_i) = p(\mathbf{y}_i \mid \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L, \mathbf{x}_i) p(\boldsymbol{\theta}) p(\boldsymbol{\theta}^o) p(\boldsymbol{\lambda}) p(\boldsymbol{\lambda}^o) p(L), \tag{4}$$

with $p(\mathbf{y}_i \mid \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L, \mathbf{x}_i)$ being the neural network described in detail in Section 3.4. Without duplicating formulas for the layer-specific readouts, and following Nazaret & Blei (2022):

$$p(\boldsymbol{\theta}) = \prod_{\ell=1}^{\infty} \prod_{n=1}^{\infty} p(\boldsymbol{\theta}_{\ell n}) = \prod_{\ell=1}^{\infty} \prod_{n=1}^{\infty} \mathcal{N}(\boldsymbol{\theta}_{\ell n}; \mathbf{0}, \mathrm{diag}(\boldsymbol{\sigma}_\ell^\theta)) \tag{5}$$

$$p(\boldsymbol{\lambda}) = \prod_{\ell=1}^{\infty} p(\lambda_\ell) = \prod_{\ell=1}^{\infty} \mathcal{N}(\lambda_\ell; \mu_\ell^\lambda, \sigma_\ell^\lambda) \quad p(L) = \mathrm{Poisson}(L; \cdot), \tag{6}$$

where the index n refers to a specific neuron. The choice of Gaussian and Poisson simply follows standard practice in the literature, and the factorizations account for an infinite number of neurons and layers. The hyper-parameters $\mu_\ell^\lambda, \sigma_\ell^\lambda, \sigma_\ell^\theta$ (one set for each hidden layer and one for each readout) control the strength of the prior distributions, but they can also be uninformative.

3.3 Variational Inference and ELBO Optimization

Maximizing the log-likelihood term $p(\mathbf{y} \mid \mathbf{x})$ of this probabilistic model requires marginalizing over all the latent variables, yielding the following intractable integral:

$$p(\mathbf{y} \mid \mathbf{x}) = \sum_{L=1}^{\infty} \int_{\boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o} p(\mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L \mid \mathbf{x}). \tag{7}$$

We therefore employ variational inference to optimize the evidence lower bound (ELBO) instead:

$$\log p(\mathbf{y} \mid \mathbf{x}) \geq \mathbb{E}_q \left[\log \frac{p(\mathbf{y}, \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L \mid \mathbf{x})}{q(\boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L)} \right] = \mathcal{L}_{\text{ELBO}} \tag{8}$$

where the variational distribution $q(\boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L)$ is an arbitrary distribution that we now define. First, the distribution factorizes as

$$q(\boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L) = q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}, L; \boldsymbol{\rho}) q(\boldsymbol{\theta}^o \mid L, \boldsymbol{\lambda}^o; \boldsymbol{\rho}^o) q(\boldsymbol{\lambda} \mid L; \boldsymbol{\nu}) q(\boldsymbol{\lambda}^o \mid L; \boldsymbol{\nu}^o) q(L; \boldsymbol{\alpha}) \tag{9}$$

where $\boldsymbol{\rho} = \{\rho_\ell\}, \boldsymbol{\rho}^o = \{\rho_\ell^o\}, \boldsymbol{\nu} = \{\nu_\ell\}, \boldsymbol{\nu}^o = \{\nu_\ell^o\}\ \forall \ell \in [1, \hat{L}]$, and $\boldsymbol{\alpha}$ are the *learnable variational parameters*, while $\hat{L}$ is the truncated depth of the network (Section 3.4).

Similar to Errica et al. (2026), let us define the discrete width D_ℓ of layer ℓ as a function of the latent variable λ_ℓ (Section 3.4 provides the details). Then, we define the variational terms as follows, once again avoiding duplicating formulas for the readout distributions:

$$q(\boldsymbol{\theta} \mid L, \boldsymbol{\lambda}; \boldsymbol{\rho}) = \prod_{\ell=1}^{L} \prod_{n=1}^{D_\ell} \mathcal{N}(\boldsymbol{\theta}_{\ell n}; \text{diag}(\boldsymbol{\rho}_{\ell n}), \mathbf{I}) \prod_{n'=D_\ell+1}^{\infty} p(\boldsymbol{\theta}_{\ell n'}) \prod_{\ell'=L+1}^{\infty} p(\boldsymbol{\theta}_{\ell'}) \tag{10}$$

$$q(\boldsymbol{\lambda} \mid L; \boldsymbol{\nu}) = \prod_{\ell=1}^{L} \mathcal{N}(\lambda_\ell; \nu_\ell, 1) \prod_{\ell'=L+1}^{\infty} p(\lambda_{\ell'}) \tag{11}$$

$$q(L; \mu_L, \sigma_L, \gamma_L) = \text{DiscreteSkewNormal}(L; \mu_L, \sigma_L, \gamma_L)\ \ (\text{Appendix}D) \tag{12}$$

where $\boldsymbol{\alpha} = (\mu_L, \sigma_L, \gamma_L)$ are learnable parameters of the Skew Normal (SN) in its centered parametrization (Arellano-Valle & Azzalini, 2008). The choice of the centered SN allows modeling asymmetric distributions in contrast to the Poisson used by Nazaret & Blei (2022). The standard SN density with shape parameter α is

$$f_{\text{SN}}(z; \alpha) = 2\phi(z)\Phi(\alpha z), \qquad z \in \mathbb{R},\ \alpha \in \mathbb{R}, \tag{13}$$

where ϕ and Φ denote the standard Normal density and cumulative distribution function. The parameter α controls the asymmetry, with $\alpha = 0$ recovering the Normal distribution.
In SOAP, we use a positive-support, discretized SN parameterized by its mean, standard deviation, and skewness, $(\mu_L, \sigma_L, \gamma_L)$. A complete presentation is given in Section D.

Learning objective. With this factorization in place, we can show the ELBO for SOAP becomes

$$\begin{aligned}\mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(L)} \Bigg[& \underbrace{\mathbb{E}_{q(\boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o \mid L)}[\log p(\mathbf{y}_i \mid \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L, \mathbf{x}_i)]}_{\text{Predictive Objective}} \\ & - \underbrace{\text{KL}(q(\boldsymbol{\theta} \mid L) \parallel p(\boldsymbol{\theta})) - \text{KL}(q(\boldsymbol{\theta}^o \mid L) \parallel p(\boldsymbol{\theta}^o))}_{\text{Weights Regularizer}} \\ & - \underbrace{\text{KL}(q(\boldsymbol{\lambda} \mid L) \parallel p(\boldsymbol{\lambda})) - \text{KL}(q(\boldsymbol{\lambda}^o \mid L) \parallel p(\boldsymbol{\lambda}^o))}_{\text{Width Regularizer}} \Bigg] \quad - \underbrace{\text{KL}(q(L) \parallel p(L))}_{\text{Depth Regularizer}}.\end{aligned} \tag{14}$$

where KL denotes the Kullback-Leibler divergence and we report the full derivation of the ELBO in Section B. The ELBO incorporates three objectives: the predictive term, the weights' regularizer, and model complexity regularizers with respect to depth and layer-wise widths. Note that the regularization terms are completely optional, as we can choose the priors to be uninformative. In any case, the more data samples we have, the stronger the predictive term will dominate the ELBO.

3.4 Network Truncation drives Self-Organization

We formalized SOAP as an unbounded sequence of transformations with unbounded hidden representations. At each forward pass, both dimensions are truncated using the current values of variational parameters optimized by the ELBO. Gradients therefore act on continuous quantities that determine how and when discrete structural changes occur (Nazaret & Blei, 2022; Errica et al., 2026).

Let p be a discrete distribution over $\mathbb{N}^+$ *which we want to truncate to have finite support*, with cumulative mass function $F_p(t) = \sum_{x=1}^{t} p(x)$. For $k \in (0, 1)$, we define the quantity

$$Q_k[p] \coloneqq \min\{t \in \mathbb{N}_0 \mid F_p(t) \geq k\}. \tag{15}$$

where $k > 0.9$ in our experiments, following previous works. The corresponding upper-truncated and normalized distribution is computed as

$$\frac{p(x)\mathbb{1}[x \leq Q_k[p]]}{F_p(Q_k[p])}. \tag{16}$$

This distribution has finite support and omits at most $1-k$ probability mass of the original distribution. In SOAP, it determines both the finite layer widths D_ℓ and network depth $\widehat{L}$, as we now show.

Width truncation and soft ordering of neurons. Each layer ℓ associates its neurons with a monotonically decreasing importance distribution $f_k^\ell(j;\nu_\ell)$ parametrized by ν_ℓ, where $j \in \mathbb{N}^+$ is the neuron's index. Let us define $F_{\nu_\ell}(j) = 1 - e^{-\nu_\ell j}$ as the CDF of an exponential distribution. In this paper, we discretize the exponential distribution similarly to Roy (2003):

$$p_\ell(j;\nu_\ell) = F_{\nu_\ell}(j) - F_{\nu_\ell}(j-1) = (1 - e^{-\nu_\ell j}) - (1 - e^{-\nu_\ell (j-1)}) = (e^{\nu_\ell} - 1)e^{-\nu_\ell j}. \tag{17}$$

By applying Equation (16) to truncate the distribution, we obtain

$$f^\ell(j;\nu_\ell) = \frac{e^{-\nu_\ell j}}{\sum_{r=1}^{D_\ell} e^{-\nu_\ell r}}, \qquad j \in [1, ..D_\ell], \tag{18}$$

which turns out to be a softmax assigning greater importance to earlier neurons. The width D_ℓ can be readily computed with Equation 15, often in closed form as in the case of the exponential.

The same probabilities that determine D_ℓ also modulate the hidden activations:

$$h_j^\ell = \sigma\left(\sum_{r=1}^{D_{\ell-1}} w_{jr}^\ell h_r^{\ell-1} + b_j^\ell\right) f^\ell(j;\nu_\ell), \qquad j \in [1, ..D_\ell]. \tag{19}$$

Hence, gradients with respect to ν_ℓ flow through f^ℓ, whereas a quantile transition triggers the discrete reshaping in Algorithm 1. We can apply the same width adaptation to each readout o_ℓ as well; implementation details on creation and deletion of neurons are discussed in detail in Section C.

Depth truncation. The variational posterior $q(L;\boldsymbol{\alpha})$ is analogously defined as:

$$\widehat{L} = Q_k[q(\cdot;\boldsymbol{\alpha})] \quad \text{and} \quad q(L;\boldsymbol{\alpha}) = \frac{\text{DiscreteSkewNormal}(L;\boldsymbol{\alpha})}{\sum_{r=1}^{\widehat{L}} \text{DiscreteSkewNormal}(r;\boldsymbol{\alpha})}, \qquad L \in [1, ..\hat{L}]. \tag{20}$$

Therefore only transformations and readouts up to $\widehat{L}$ are active. The quantile updates the discrete structure, while $\boldsymbol{\alpha}$ remains learnable through $q(L;\boldsymbol{\alpha})$ in the ELBO (Equation (14)).

Prediction. The prediction of the neural network is computed after applying a first-order approximation (Section A) as in Nazaret & Blei (2022), resulting in

$$p(\mathbf{y} \mid \mathbf{x}) \approx \mathbb{E}_{q(L;\boldsymbol{\alpha})}\left[p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\rho}, \boldsymbol{\rho}^o, \boldsymbol{\nu}, \boldsymbol{\nu}^o, L)\right] = \sum_{\ell=0}^{\widehat{L}} q(\ell;\boldsymbol{\alpha}) \underbrace{p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\rho}, \boldsymbol{\rho}^o, \boldsymbol{\nu}, \boldsymbol{\nu}^o, L = \ell)}_{\text{output of the } \ell\text{-th readout}}. \tag{21}$$

Thus, SOAP extends the multi-readout architecture of Section 3.1 by weighting local predictions with q. For fixed support, Equations (19) and (21) provide gradients for $\boldsymbol{\alpha}, \boldsymbol{\nu}, \boldsymbol{\nu}^o$; crossing a quantile boundary triggers the next UPDATE-STRUCTURE of Algorithm 1.

3.5 Limitations and Future Work

SOAP relies on specific distributional and architectural choices that follow in part previous work. The choice works well empirically but would require a more solid theoretical motivation, for instance by extending the work of Caron et al. (2025) on asymmetric activations' rescalings. Moreover, each candidate depth requires a separate linear or one-hidden-layer MLP readout, which we cannot model as SOAP recursively. While an MLP readout is universal, we also investigate the impact of using a linear one instead. Finally, Equation (21) evaluates all active readouts, even when q assigns negligible mass. Pruning such heads at inference and renormalizing the remaining weights could greatly reduce total computation and parameters (Errica et al., 2025), but we leave this trade-off to future work.

4 Experiments

Our objective is **not** to establish a new state of the art performance, rather demonstrating that the adaptive mechanisms of SOAP can match the predictive performance of established baselines while allowing the network to jointly learn its width and depth. We therefore evaluate SOAP empirically

on a set of tabular classification benchmarks, including synthetic and real-world datasets. Specifically, we report evaluation metrics for the following datasets: Spiral, SpiralHard (Errica et al., 2026), DoubleMoon, POL (OpenML Contributors, 2026; Vanschoren et al., 2014), CreditCards (Yeh & Lien, 2009), and MiniBooNE (Roe et al., 2005). Our primary goal is to assess whether the introduction of two complementary adaptation dimensions can retain the predictive performance of fixed-topology baselines. In doing so, we also evaluate the capacity allocation of the discovered networks to see if the model can successfully adapt to the complexity of the task. In addition, we study the learned topologies under different initializations to characterize the adaptation dynamics of the model.
We compare SOAP against two baselines. The first is a conventional *Fixed* multilayer perceptron whose depth and width are selected via a grid search and fixed before training. The second baseline is AWN (Errica et al., 2026), which only adapts the widths of its hidden layers while keeping the network depth fixed. Results for the baselines and data splits are taken from Errica et al. (2026).
For the predictive experiments, we trained the model using a fixed batch size of 128 and a maximum of 5000 epochs via the Adam optimizer with default beta parameters. Training is terminated early when the validation accuracy does not improve for 500 epochs. Every tested configuration used a minimum depth of one hidden layer and a minimum hidden width of two neurons to avoid degenerate cases. The width and depth quantile thresholds are set to 0.9 and 0.95, respectively; the remaining hyperparameters are reported in Table 4, which defines the space for our model selection.
For every candidate SOAP configuration, we perform two independent model-selection runs using different random seeds, with configurations being selected using the internal validation set. The selection process prioritizes predictive accuracy and subsequently considers cross-entropy, depth, and total width as tie-breakers. Finally, the selected configuration is then retrained ten times from independent random initializations for the final risk assessment phase, which uses the outer training and validation partitions. The outer test set remains untouched until the final evaluation. In Section 5, we report the mean and standard deviation of the predictive accuracy over the ten risk-assessment runs, together with the corresponding converged architectural metrics.

5 RESULTS

We summarize here the results of the empirical analysis described earlier, starting with the quantitative results, followed by a discussion of the topologies explored by the model. Finally, we extend our analysis to the stability properties of SOAP, by showing how the network behaves as we vary its initial conditions for both width and depth.

Predictive Performance. Table 1 reports the accuracy achieved by the model, averaged across all ten final runs. In particular, the reported metrics show that SOAP matches or exceeds the predictive performance of both baselines across all datasets. On the synthetic datasets (DoubleMoon, Spiral, and SpiralHard), SOAP achieves perfect accuracy. On the real-world datasets, SOAP statistically matches the performance of the AWN baseline and, on MiniBooNE, outperforms both the Fixed and AWN architectures.
Table 1: Mean and std performance comparison of the Fixed baseline, AWN, and SOAP models. Bold results are statistically significant improvements.

Dataset	Fixed	AWN	SOAP
	Accuracy Mean (Std)	Accuracy Mean (Std)	Accuracy Mean (Std)
DoubleMoon	100.0 (0.0)	100.0 (0.0)	100.0 (0.0)
Spiral	99.5 (0.5)	99.8 (0.1)	100.0 (0.0)
SpiralHard	98.0 (2.0)	100.0 (0.0)	100.0 (0.0)
pol	99.3 (0.2)	99.2 (0.1)	99.0 (0.3)
CreditCards	81.6 (0.1)	81.8 (0.1)	81.7 (0.1)
MiniBooNE	92.9 (0.3)	93.2 (0.1)	**93.8** (0.1)

Structural Trends. More interestingly, we can observe how SOAP allocates its representational capacity depending on the task. Table 2 reports the converged architectures learned by SOAP, alongside the selected fixed architectures and the width of AWN. Across the three synthetic tasks, SOAP discovers increasingly complex topologies as task difficulty increases, with the same trend also observed for the total number of readout neurons. These results suggest that SOAP tends to allocate

additional representational capacity through both increased depth and width as the complexity of the task grows, without a corresponding loss in predictive performance.

Table 2: Architectural metrics for the Fixed baseline, AWN, and SOAP. Results report the mean and standard deviation (in parentheses) of the converged topologies across ten random seeds.

Dataset	Fixed	AWN		SOAP			
	Width	Depth	Width	Depth	Width	Mean Width per Layer	Readout Width
DoubleMoon	8	1	8 (3)	1 (0)	23 (0)	23 (0)	46 (0)
Spiral	16	1	66 (9)	2 (0)	375 (38)	188 (19)	54 (22)
SpiralHard	32	1	227 (32)	4 (1)	942 (360)	224 (58)	160 (76)
pol	48	1	84 (11)	2 (1)	266 (192)	166 (72)	602 (747)
CreditCards	16	1	51 (12)	1 (0)	8 (6)	8 (6)	30 (6)
MiniBooNE	32	4	53 (11)	3 (0)	246 (59)	84 (17)	53 (22)

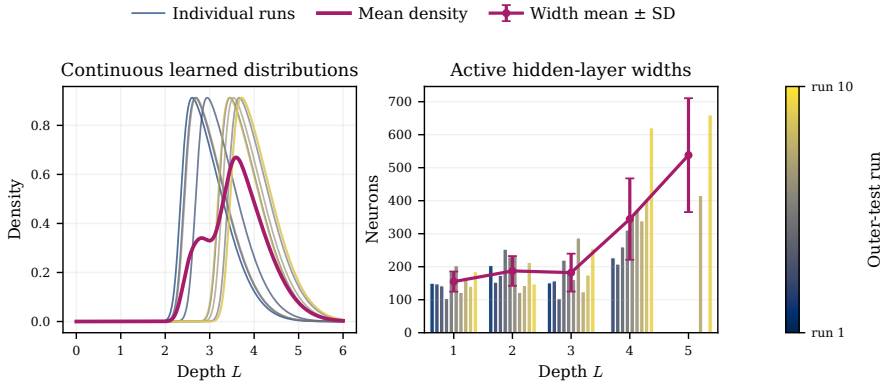


Figure 2: Final learned depth distributions and hidden-layer widths for the selected SOAP configuration on SpiralHard, averaged over ten outer test runs. The left panel shows the continuous depth distributions, while the right panel shows the hidden-layer widths by layer. Note that since depth is adaptive, only the width of two configurations is reported for layer $L = 5$.

The topology of the final outer test runs for the selected SOAP configuration on SpiralHard is depicted in Figure 2. The figure shows the learned continuous Skew Normal distributions (left) and the per-layer width allocation (right). The learned distributions across random initializations mainly differ in the location of their peaks, while the spread and skewness parameters remain relatively stable. The learned SN distribution always exhibits a right skew, with most of the probability mass concentrated around layers 3–5. The corresponding neuron allocation is predominantly tail-loaded, suggesting that, for this particular task, the model favors allocating representational capacity towards the last layers. The capacity values for SOAP are predominantly larger than those of AWN. This is partly because model selection favored configurations without weight regularization, which allow larger topologies. These results should therefore be interpreted as evidence of adaptive capacity allocation, rather than as a direct comparison of parameter efficiency.

Topological Convergence. To further investigate the empirical results reported in Tables 1 and 2 and assess the stability of SOAP, we analyze the evolution of its learned topology over training epochs across a range of model hyperparameters for the SpiralHard dataset. Here we only report the results of the analysis after fixing the batch size to 128, and focus on regularized configurations with $\sigma_\theta = 10$; The complete set of configurations considered in the analysis is detailed in Table 5.

As shown in Figure 3, SOAP exhibits width-convergence properties similar to those reported by Errica et al. (2025), consistently converging to a comparable total number of neurons. This suggests that introducing an additional axis of topological adaptivity does not substantially alter the stability properties of the underlying width-adaptation mechanism. More interestingly, the figure reveals a distinct convergence behavior along the orthogonal axis of learnable depth. SOAP exhibits remarkably stable convergence in terms of the number of layers, only slightly dependent of its initial topology. In particular, although the models are initialized with different exponential rates and values of μ, corresponding to substantially different numbers of hidden neurons and different initial depths, the

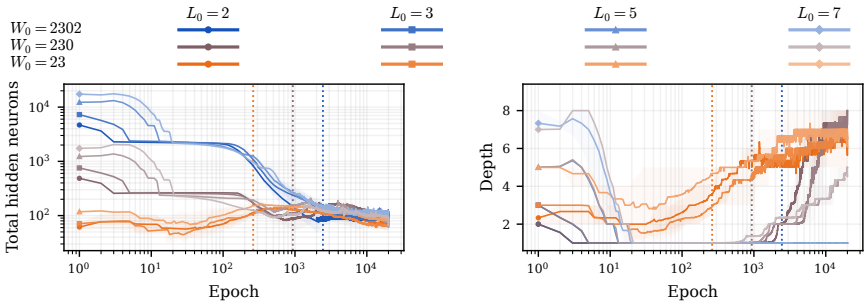


Figure 3: Width and depth trajectories for stability configurations with batch size 128 and $\sigma_\theta = 10$. Starting total width and depth are denoted by W_0 and L_0, respectively. Vertical dotted lines mark 99% validation accuracy is achieved, on average, for each configuration grouped by rate ν.

different configurations tend to converge to comparable number of layers while exhibiting similar accuracy trajectories. The only notable exception observed in our experiments occurs for $\nu = 0.001$,

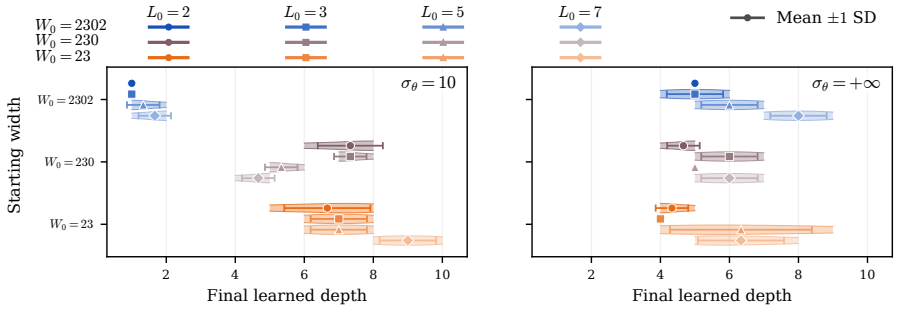


Figure 4: Final learned depth for stability configurations (batch size 128). The panels show $\sigma_\theta = 10$ (left) and no θ prior (right). Starting total width and depth are denoted by W_0 and L_0, respectively.

corresponding to approximately 2300 neurons per layer. In this regime, the model consistently retains a single hidden layer while still successfully adapting its width to discover the number of neurons required to solve the task. This behavior highlights the flexibility of the proposed approach: rather than converging to a fixed architectural regime, SOAP can adapt its topology along different axes depending on the initial conditions and the requirements of the task. The violin plots in Figure 4 illustrate this behavior more clearly: when moderate regularization is introduced ($\sigma_\theta = 10$) the final learned depth is only weakly influenced by the initial number of layers, while being more strongly determined by the initial hidden neurons population. Conversely, when no weight regularization is applied, the starting number of layers has a stronger impact on the final learned depth.
Last but not least, in contrast to Errica et al. (2025) where larger batch sizes promote better topological convergence, SOAP does not exhibit the same dependency on batch size, achieving a stable convergence trajectory even for batches as small as 32 training samples.

6 Conclusion

This work introduced, for the first time, a variational formalism for the *joint* learning of unbounded width and depth in MLPs called SOAP. In this approach, hard architectural choices ultimately depend on the task to be solved via backpropagation and not on researchers, who become responsible for higher-level design choices. We showed SOAP recovers similar or better performance than fixed MLPs, and we observed stable convergence trends in both width and depth with respect to different starting configurations. Our contribution paves the way for new studies on the differentiable relation between width and depth and the consequent representational power of neural networks, building upon previous results for fixed-topology models.

AI USE STATEMENT

Generative AI tools were used only in a supporting role. The authors developed the mathematical derivations, core implementation, and experiments independently. AI tools were used to check notation and mathematical conventions, and, after the experiments had been completed, to assist with limited post hoc processing and presentation of the resulting data. This included generating or editing code for extracting and aggregating existing results and producing scientific figures.

AI tools were also used to search and summarize software documentation. They also helped improve the clarity and organization of parts of the manuscript. They did not influence the methods, hypotheses, experimental design, hyper-parameter choices, and were not used to generate or clean datasets, formulate mathematical claims, write proofs, or draw the paper's sections.

The authors reviewed all AI-assisted text, mathematics, code, and figures. Results and plots were checked against the underlying experiment outputs, and the authors take responsibility for the final content of this work, including all claims and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The authors put substantial effort into making the experiments reproducible with minimal user intervention. Section E presents the experimental details, including the fixed hyperparameters, those searched during model selection, and descriptions of the datasets used. Section C describes implementation details concerning the structural adaptivity of SOAP, including initialization and network reshaping. Section D contains details of the depth distribution used throughout the paper and its related mathematical derivations. Finally, the accompanying supplementary material provides instructions for reproducing the experiments.

REFERENCES

Reinaldo B. Arellano-Valle and Adelchi Azzalini. The centred parametrization for the multivariate skew-normal distribution. *Journal of Multivariate Analysis*, 99:1362–1382, 2008. ISSN 0047259X. doi: 10.1016/j.jmva.2008.01.020.

A. Azzalini. A class of distributions which includes the normal ones. *Scandinavian Journal of Statistics*, 12:171–178, 1985. ISSN 0303-6898.

Adelchi Azzalini. The skew-normal distribution and related multivariate families. *Scandinavian Journal of Statistics*, 32:159–188, 2005. ISSN 0303-6898.

Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of Machine Learning and Systems*, 2:129–146, 2020.

Francois Caron, Fadhel Ayed, Paul Jung, Hoil Lee, Juho Lee, and Hongseok Yang. Over-parameterised shallow neural networks with asymmetrical node scaling: Global convergence guarantees and feature learning. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856.

Federico Errica, Henrik Christiansen, Viktor Zaverkin, Takashi Maruyama, Mathias Niepert, and Francesco Alesiani. Adaptive message passing: A general framework to mitigate oversmoothing, oversquashing, and underreaching. In *Proceddings of the 43rd International Conference on Machine Learning (ICML)*, 2025.

Federico Errica, Henrik Christiansen, Viktor Zaverkin, Mathias Niepert, and Francesco Alesiani. Adaptive width neural networks. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*, 2026.

Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

Cédric Flecher, Denis Allard, and Philippe Naveau. Truncated skew-normal distributions: Moments, estimation by weighted moments and application to climatic data. *METRON*, 68:331–345, 2010. ISSN 0026-1424, 2281-695X. doi: 10.1007/BF03263543.

Bernd Fritzke. A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7, 1994.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

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.

Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 2018.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. doi: 10.48550/arXiv.1503.02531.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019a.

Qiang Liu, Lemeng Wu, and Dilin Wang. Splitting steepest descent for growing neural architectures. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS)*, 2019b.

Hrushikesh Mhaskar, Qianli Liao, and Tomaso Poggio. When and why are deep networks better than shallow ones? In *Proceedings of the AAAI conference on Artificial Intelligence (AAAI)*, 2017.

Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021. doi: 10.48550/arXiv.2104.08378.

Rupert Mitchell, Robin Menzenbach, Kristian Kersting, and Martin Mundt. Self-expanding neural networks. *arXiv preprint arXiv:2307.04526*, 2024.

Achille Nazaret and David Blei. Variational inference for infinitely deep neural networks. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, 2022.

Thao Nguyen, Maithra Raghu, and Simon Kornblith. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021.

OpenML Contributors. pol, 2026. OpenML Dataset ID: 43991 (Variant of ID: 722).

Donald B. Owen. Tables for computing bivariate normal probabilities. *The Annals of Mathematical Statistics*, 27:1075–1090, 1956. ISSN 0003-4851, 2168-8990. doi: 10.1214/aoms/1177728074.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pp. 8821–8831, 2021.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

Byron P. Roe, Hai-Jun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543:577–584, 2005. ISSN 0168-9002. doi: 10.1016/j.nima.2004.12.018.

Dilip Roy. The discrete normal distribution. *Communications in Statistics - Theory and Methods*, 32: 1871–1883, 2003. ISSN 0361-0926. doi: 10.1081/STA-120023256.

Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, November 1998. ISSN 0899-7667. doi: 10.1162/089976698300016963.

Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pp. 6105–6114. PMLR, 2019.

Lorenzo Valerio, Franco Maria Nardini, Andrea Passarella, and Raffaele Perego. Dynamic hard pruning of neural networks at the edge of the internet. *Journal of Network and Computer Applications*, 200:103330, 2022.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. In *9th ISCA Workshop on Speech Synthesis Workshop (SSW)*, 2016.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15:49–60, 2014. ISSN 1931-0145, 1931-0153. doi: 10.1145/2641190.2641198.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

Martin Wistuba, Ambrish Rawat, and Tejaswini Pedapati. A survey on neural architecture search. *arXiv preprint arXiv:1905.01392*, 2019. doi: 10.48550/arXiv.1905.01392.

Lemeng Wu, Bo Liu, Peter Stone, and Qiang Liu. Firefly neural architecture descent: a general approach for growing neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 22373–22383. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/fdbe012e2e11314b96402b32c0df26b7-Paper.pdf.

I-Cheng Yeh and Che-hui Lien. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications*, 36: 2473–2480, 2009. ISSN 0957-4174. doi: 10.1016/j.eswa.2007.12.020.

Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. In *6th International Conference on Learning Representations (ICLR)*, 2018.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2017. doi: 10.48550/arXiv.1611.01578.

A First Order Approximation

Computing expectations over $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}, L; \boldsymbol{\rho})$ and $q(\boldsymbol{\lambda} \mid L; \boldsymbol{\nu})$ requires integrating over the continuous space of the latent variables, which is intractable. Following Nazaret & Blei (2022), we therefore employ the following first order Taylor approximation to compute expectations:

$$\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\lambda},L;\boldsymbol{\rho})}\left[f(\boldsymbol{\theta})\right] \approx f(\boldsymbol{\rho}), \tag{22}$$

and likewise for $\boldsymbol{\lambda}$:

$$\mathbb{E}_{q(\boldsymbol{\lambda}|L;\boldsymbol{\nu})}\left[f(\boldsymbol{\lambda})\right] \approx f(\boldsymbol{\nu}). \tag{23}$$

To see why the approximation holds, consider a random vector $X \in \mathbb{R}^d$ with probability density p and let $\mathbf{x}$ denote a realization of X. Moreover, let $f(\mathbf{x})$ be a sufficiently smooth function that takes as input realizations of X, meaning that its output is also randomly distributed. Recall the first-order Taylor expansion of $f(\mathbf{x})$ at expansion point $\boldsymbol{\mu}_x = \mathbb{E}_p[X]$ is:

$$f(\mathbf{x}) = f(\boldsymbol{\mu}_x) + \nabla f(\boldsymbol{\mu}_x)^T(\mathbf{x} - \boldsymbol{\mu}_x) + R_{\geq 2}(\mathbf{x}). \tag{24}$$

Here, $R_{\geq 2}(\mathbf{x})$ collects all Taylor terms beyond first order. Substituting it at the first-order into the definition of $\mathbb{E}_p[f(X)]$ gives

$$\begin{aligned}
\mathbb{E}_p[f(X)] &= \int_{\mathbb{R}^d} f(\mathbf{x})p(\mathbf{x})d\mathbf{x} && (25)\\
&\approx \int_{\mathbb{R}^d} p(\mathbf{x})\left(f(\boldsymbol{\mu}_x) + \nabla f(\boldsymbol{\mu}_x)^T(\mathbf{x} - \boldsymbol{\mu}_x)\right)d\mathbf{x} && (26)\\
&= \int_{\mathbb{R}^d} p(\mathbf{x})f(\boldsymbol{\mu}_x)d\mathbf{x} + \int_{\mathbb{R}^d} p(\mathbf{x})\nabla f(\boldsymbol{\mu}_x)^T(\mathbf{x} - \boldsymbol{\mu}_x)d\mathbf{x} && (27)\\
&= f(\boldsymbol{\mu}_x) + \int_{\mathbb{R}^d} p(\mathbf{x}) \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}(\boldsymbol{\mu}_x)(x_i - \boldsymbol{\mu}_{x,i})d\mathbf{x} && (28)\\
&= f(\boldsymbol{\mu}_x) + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}(\boldsymbol{\mu}_x) \int_{\mathbb{R}^d} p(\mathbf{x})(x_i - \boldsymbol{\mu}_{x,i})d\mathbf{x} && (29)\\
&= f(\boldsymbol{\mu}_x) + \sum_{i=1}^{d} \frac{\partial f}{\partial x_i}(\boldsymbol{\mu}_x) \underbrace{\mathbb{E}_p[X_i - \boldsymbol{\mu}_{x,i}]}_{=0} && (30)\\
&= f(\boldsymbol{\mu}_x) && (31)
\end{aligned}$$

B Full ELBO Derivation

Consider a single pair of i.i.d. training signals $\mathbf{x}_i, \mathbf{y}_i$. Building upon the methodology established by UDN (Nazaret & Blei, 2022), the ELBO formulation for SOAP incorporates the additional latent variables for width ($\boldsymbol{\lambda}, \boldsymbol{\lambda}^o$) and readout weights ($\boldsymbol{\theta}^o$):

$$\log p(\mathbf{y}_i \mid \mathbf{x}_i) \geq \mathbb{E}_{q(\boldsymbol{\theta},\boldsymbol{\theta}^o,\boldsymbol{\lambda},\boldsymbol{\lambda}^o,L)} \log \frac{p(\mathbf{y}_i, \boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L \mid \mathbf{x}_i)}{q(\boldsymbol{\theta}, \boldsymbol{\theta}^o, \boldsymbol{\lambda}, \boldsymbol{\lambda}^o, L)}, \tag{32}$$

where we dropped the variational parameters $\boldsymbol{\rho}, \boldsymbol{\rho}^o, \boldsymbol{\nu}, \boldsymbol{\nu}^o, \mu, \sigma, \gamma$ of $q(\cdot)$ for convenience of notation.

Exploiting the conditional independence assumptions from the graphical model of Figure 1a, the joint distribution factorizes as defined in Equation (4). Applying the nested structure of the variational

family q and separating the expectation over the depth variable L yields:

$$\begin{aligned}\log p(\mathbf{y}_i \mid \mathbf{x}_i) \geq \mathbb{E}_{q(L)}\Bigg[&\mathbb{E}_{q(\boldsymbol{\theta},\boldsymbol{\theta}^o,\boldsymbol{\lambda},\boldsymbol{\lambda}^o|L)}\left[\log p(\mathbf{y}_i \mid \boldsymbol{\theta},\boldsymbol{\theta}^o,\boldsymbol{\lambda},\boldsymbol{\lambda}^o,L,\mathbf{x}_i)\right]\\ &+\mathbb{E}_{q(\boldsymbol{\theta},\boldsymbol{\theta}^o,\boldsymbol{\lambda},\boldsymbol{\lambda}^o|L)}\left[\log\frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta}\mid L,\boldsymbol{\lambda})}+\log\frac{p(\boldsymbol{\theta}^o)}{q(\boldsymbol{\theta}^o\mid L,\boldsymbol{\lambda}^o)}\right]\\ &+\mathbb{E}_{q(\boldsymbol{\theta},\boldsymbol{\theta}^o,\boldsymbol{\lambda},\boldsymbol{\lambda}^o|L)}\left[\log\frac{p(\boldsymbol{\lambda})}{q(\boldsymbol{\lambda}\mid L)}+\log\frac{p(\boldsymbol{\lambda}^o)}{q(\boldsymbol{\lambda}^o\mid L)}\right]\\ &+\log\frac{p(L)}{q(L)}\Bigg].\end{aligned} \tag{33}$$

By applying the factorizations defined in Equations (5), (6), (10) and (11), we can expand the log ratios — recall that D_ℓ is defined in Equation (18) to be the width truncation point for layer ℓ at the current stage of optimization:

$$\log\frac{p(\boldsymbol{\theta})}{q(\boldsymbol{\theta}\mid L,\boldsymbol{\lambda})}=\log\frac{\prod_{\ell=1}^{\infty}\prod_{n=1}^{\infty}p(\boldsymbol{\theta}_{\ell n})}{\prod_{\ell=1}^{L}\left[\prod_{n=1}^{D_\ell}q(\boldsymbol{\theta}_{\ell n})\prod_{n'=D_\ell+1}^{\infty}p(\boldsymbol{\theta}_{\ell n'})\right]\prod_{\ell'=L+1}^{\infty}p(\boldsymbol{\theta}_{\ell'})} \tag{34}$$

$$=\log\frac{\prod_{\ell=1}^{L}\prod_{n=1}^{D_\ell}p(\boldsymbol{\theta}_{\ell n})}{\prod_{\ell=1}^{L}\prod_{n=1}^{D_\ell}q(\boldsymbol{\theta}_{\ell n})}, \tag{35}$$

$$\log\frac{p(\boldsymbol{\lambda})}{q(\boldsymbol{\lambda}\mid L)}=\log\frac{\prod_{\ell=1}^{\infty}p(\lambda_\ell)}{\prod_{\ell=1}^{L}q(\lambda_\ell)\prod_{\ell'=L+1}^{\infty}p(\lambda_{\ell'})} \tag{36}$$

$$=\log\frac{\prod_{\ell=1}^{L}p(\lambda_\ell)}{\prod_{\ell=1}^{L}q(\lambda_\ell)}. \tag{37}$$

Analogous factorizations can be derived for $\boldsymbol{\theta}^o,\boldsymbol{\lambda}^o$. Moreover, applying the first-order approximation discussed in Section A, we obtain:

$$\begin{aligned}\log p(\mathbf{y}_i \mid \mathbf{x}_i) \geq \mathbb{E}_{q(L)}\Bigg[&\log p(\mathbf{y}_i \mid \boldsymbol{\rho},\boldsymbol{\rho}^o,\boldsymbol{\nu},\boldsymbol{\nu}^o,L,\mathbf{x}_i)+\log\frac{p(L)}{q(L)}\\ &+\log\frac{\prod_{k=1}^{L}\prod_{n=1}^{D_k}p(\boldsymbol{\rho}_{kn})}{\prod_{k=1}^{L}\prod_{n=1}^{D_k}q(\boldsymbol{\rho}_{kn})}+\log\frac{\prod_{k=0}^{L}\prod_{m=1}^{D_k^o}p(\boldsymbol{\rho}^o_{km})}{\prod_{k=0}^{L}\prod_{m=1}^{D_k^o}q(\boldsymbol{\rho}^o_{km})}\\ &+\log\frac{\prod_{k=1}^{L}p(\boldsymbol{\nu}_k)}{\prod_{k=1}^{L}q(\boldsymbol{\nu}_k)}+\log\frac{\prod_{k=0}^{L}p(\boldsymbol{\nu}^o_k)}{\prod_{k=0}^{L}q(\boldsymbol{\nu}^o_k)}\Bigg]\end{aligned} \tag{38}$$

$$\begin{aligned}=\mathbb{E}_{q(L)}\Bigg[&\log p(\mathbf{y}_i \mid \boldsymbol{\rho},\boldsymbol{\rho}^o,\boldsymbol{\nu},\boldsymbol{\nu}^o,L,\mathbf{x}_i)+\log\frac{p(L)}{q(L)}\\ &+\sum_{k=1}^{L}\sum_{n=1}^{D_k}\log\frac{p(\boldsymbol{\rho}_{kn})}{q(\boldsymbol{\rho}_{kn})}+\sum_{k=0}^{L}\sum_{m=1}^{D_k^o}\log\frac{p(\boldsymbol{\rho}^o_{km})}{q(\boldsymbol{\rho}^o_{km})}\\ &+\sum_{k=1}^{L}\log\frac{p(\boldsymbol{\nu}_k)}{q(\boldsymbol{\nu}_k)}+\sum_{k=0}^{L}\log\frac{p(\boldsymbol{\nu}^o_k)}{q(\boldsymbol{\nu}^o_k)}\Bigg]\end{aligned} \tag{39}$$

where the infinite products corresponding to prior distributions deeper than L cancel out against their identical conditional prior counterparts in the denominator, directly resulting from the nested structure of both the width and depth variational families.

To simplify the notation, we group the conditional log-likelihood and the variational regularization terms into a single conditional objective $\mathcal{L}_\ell$ for a specific depth ℓ:

$$\begin{aligned}\mathcal{L}_\ell = \log p(\mathbf{y}_i \mid \boldsymbol{\rho}, \boldsymbol{\rho}^o, \boldsymbol{\nu}, \boldsymbol{\nu}^o, \ell, \mathbf{x}_i) \\ + \sum_{k=1}^{\ell} \sum_{n=1}^{D_k} \log \frac{p(\boldsymbol{\rho}_{kn})}{q(\boldsymbol{\rho}_{kn}; \boldsymbol{\rho}_{kn})} + \sum_{k=0}^{\ell} \sum_{m=1}^{D_k^o} \log \frac{p(\boldsymbol{\rho}_{km}^o)}{q(\boldsymbol{\rho}_{km}^o; \boldsymbol{\rho}_{km}^o)} \\ + \sum_{k=1}^{\ell} \log \frac{p(\boldsymbol{\nu}_k)}{q(\boldsymbol{\nu}_k; \boldsymbol{\nu}_k)} + \sum_{k=0}^{\ell} \log \frac{p(\boldsymbol{\nu}_k^o)}{q(\boldsymbol{\nu}_k^o; \boldsymbol{\nu}_k^o)}.\end{aligned} \tag{40}$$

Using Equation (15), we define $\widehat{L} = Q_k[q(\cdot; \mu_L, \sigma_L, \gamma_L)]$ to be the upper endpoint of the bounded, connected support of $q(L; \mu_L, \sigma_L, \gamma_L)$. The expectation over L can then be tractably evaluated, yielding:

$$\log p(\mathbf{y}_i \mid \mathbf{x}_i) \geq \sum_{\ell=0}^{\widehat{L}} q(\ell; \mu_L, \sigma_L, \gamma_L) \mathcal{L}_\ell - \mathrm{KL}(q(L; \alpha) \parallel p(L)), \tag{41}$$

where we isolated the depth regularization term as a Kullback-Leibler divergence.

C Dynamic Allocation and Eviction

The quantile computations described in Section 3.4 are performed before a forward pass. If D_ℓ increases, we append the corresponding rows to the weight matrix of layer ℓ and columns to that of layer $\ell + 1$; new weights are sampled from a standard Normal distribution. If D_ℓ decreases, the trailing neurons induced by the soft ordering are physically deleted. The same operations are applied to the layer-specific readouts.

When $\widehat{L}$ increases, a new hidden transformation and its readout are also appended to the network and initialized using the Kaiming+ initialization as described in Errica et al. (2026). When it decreases, layers above the active truncation are retained in memory but excluded from the forward and backward passes. If a later update restores the previous depth, the retained parameters are reactivated instead of reinitialized. Note that this retention and reactivation strategy is an implementation choice rather than a requirement of SOAP, and other policies can be considered.

All allocation and eviction operations are discrete and remain outside the computational graph; gradients only act on the active parameters and on the variational quantities described in Section 3.4.

D Improving Modeling of Depth

Previous work in unbounded networks (Nazaret & Blei, 2022) used a Poisson distribution to model the variational depth-distribution $q(L)$. Errica et al. (2025) further extended the framework by considering a discretized Folded Normal distribution. In this work, we instead adopt a centered Skew Normal, appropriately truncated and discretized to limit its support to the positive natural numbers. The Skew Normal was chosen because it adds explicit control over the distribution skewness, allowing the model to arbitrarily adapt to different depth preferences. In Section D.1 we briefly provide a formal presentation of the distribution, while describing its truncation procedure and centered parameterization in Sections D.2 and D.3, respectively. Furthermore, in Section D.4 we highlight a problematic structural collapsing behavior of the Folded Normal caused by a vanishing gradient near the degenerate point $\mu = 0$.

D.1 Skew Normal

The Skew Normal (SN) distribution was introduced by Azzalini (1985) and is defined as follows for the univariate case: Let $z \in \mathbb{R}$ and $\alpha \in \mathbb{R}$ be a parameter of the distribution, then

$$f_{\mathrm{SN}}(z; \alpha) = 2\phi(z)\Phi(\alpha z) \tag{42}$$

is the probability density function of a Skew Normal distribution with parameter α, where $\phi(z)$ is the PDF of the standard Normal and Φ is its cumulative density function. For $\alpha = 0$ we obtain the Standard Normal distribution.

The cumulative density function (CDF) of the standard SN is defined as

$$F_{\mathrm{SN}}(z;\alpha) = \Phi(z) - 2T(z,\alpha) \tag{43}$$

where

$$T(x,a) = \frac{1}{2\pi}\int_0^a \frac{\exp\left(-x^2(1+t^2)/2\right)}{1+t^2}dt \tag{44}$$

is Owen's T function Owen (1956).

Critically, for $Z \sim \mathrm{SN}(\alpha)$, $\xi \in \mathbb{R}$ and $\omega \in \mathbb{R}^+$, $Y = \xi + \omega Z$ is also SN-distributed, and we shall write for simplicity (with a slight abuse of notation) $Y \sim \mathrm{SN}(\xi,\omega,\alpha)$. The parameters ξ, ω and α are called *location*, *scale* and *shape* of the distribution, respectively.

By applying the transformation $y = g(z) = \xi + \omega z$ one can write the PDF of a general $\mathrm{SN}(\xi,\omega,\alpha)$ as:

$$f_{\mathrm{SN}}(y;\xi,\omega,\alpha) = \left|\frac{d}{dz}g^{-1}\right| f_{\mathrm{SN}}\left(g^{-1}(y);\alpha\right) = 2\omega^{-1}\phi\left(\frac{y-\xi}{\omega}\right)\Phi\left(\alpha\frac{y-\xi}{\omega}\right) \tag{45}$$

from which also follows that

$$F_{\mathrm{SN}}(y;\xi,\omega,\alpha) = F_{\mathrm{SN}}\left(\frac{y-\xi}{\omega};\alpha\right) \tag{46}$$

D.2 Truncated Skew Normal

In order to limit the support of the distribution to only positive numbers, we condition the SN on the positive quadrant and renormalize its density, akin to Flecher et al. (2010). We thus define the Truncated Skew Normal as

$$f^+(z;\xi,\omega,\alpha) = \begin{cases} \dfrac{f_{\mathrm{SN}}(z;\xi,\omega,\alpha)}{1-F_{\mathrm{SN}}(0;\xi,\omega,\alpha)} & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

It immediately follows that, for any $z > 0$, the CDF of the truncated distribution is

$$F^+(z;\xi,\omega,\alpha) = \int_0^z \frac{f_{\mathrm{SN}}(t;\xi,\omega,\alpha)}{1-F_{\mathrm{SN}}(0;\xi,\omega,\alpha)}\,\mathrm{d}t \tag{48}$$

$$= \frac{1}{1-F_{\mathrm{SN}}(0;\xi,\omega,\alpha)}\int_0^z f_{\mathrm{SN}}(t;\xi,\omega,\alpha)\,\mathrm{d}t \tag{49}$$

$$= \frac{F_{\mathrm{SN}}(z;\xi,\omega,\alpha) - F_{\mathrm{SN}}(0;\xi,\omega,\alpha)}{1-F_{\mathrm{SN}}(0;\xi,\omega,\alpha)}. \tag{50}$$

Figure 5 shows the described renormalization for different values of α.

Similarly to the Folded Normal, we apply the discretization procedure described by Roy (2003), resulting in the following discrete probability mass function:

$$p_{\mathrm{DSN}}(x;\xi,\omega,\alpha) = \begin{cases} F^+(1;\xi,\omega,\alpha) & \text{if } x = 0 \\ F^+(x+1;\xi,\omega,\alpha) - F^+(x;\xi,\omega,\alpha) & \forall x \in \mathbb{N}^+ \end{cases} \tag{51}$$

Here, x denotes the discrete depth, with the $x = 0$ mass collecting the continuous interval $[0,1)$ and each positive integer collecting the interval $[x, x+1)$.

D.3 Centered Parameterization

The original formulation of the Skew Normal has been shown to be problematic for likelihood-based optimization because of instability around $\alpha = 0$, due to the Fisher Information matrix being singular

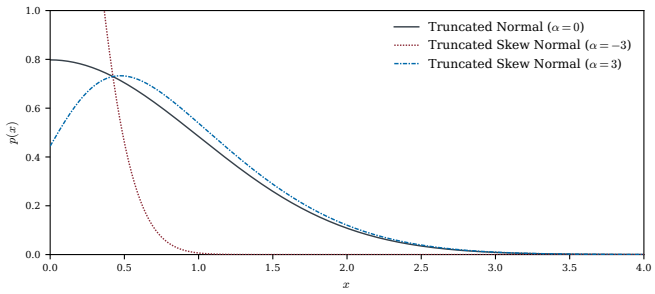


Figure 5: Probability density function of the Truncated Skew Normal distribution. The density is renormalized over the positive support $\mathbb{R}^+$ while preserving the asymmetric properties of the underlying Skew Normal.

at that point (Arellano-Valle & Azzalini, 2008). This can lead to poorly conditioned gradients and difficulty learning the correct underlying data distribution even for simple estimation tasks. Moreover, the Direct Parameterization (DP)—so called because the parameters ξ, ω, α can be read *directly* from Equation (45)—does not allow the optimizer to independently tune the mean, variance or skewness of the distribution because they are each inherently coupled with more than one direct parameter. To see this, consider a model of the Skew Normal holding the learnable parameters (ξ, ω, α): an optimization step changing just one of those parameters would cause a change in one or more of the statistical moments of the distribution. If the model wanted to change the mean μ of the distribution by a small value, the optimizer would need to adjust all three parameters ξ, ω, α at once.

To overcome these limitations, Arellano-Valle & Azzalini (2008) propose a reparametrization more suitable for estimation. The core idea of the Centered Parameterization (CP) is to replace the direct parameters with the actual statistical moments of the distribution: the mean $\mu = \mathbb{E}[Y]$, the variance $\sigma^2 = \mathrm{Var}\,[Y]$, and the third standardized moment (or *skewness coefficient*) γ_1.

Consider a standard Skew Normal random variable $Z \sim \mathrm{SN}(0, 1, \alpha)$ and the affine transformation $Y = \xi + \omega Z$. To establish the mapping between the two parameterizations, we first define the following intermediate variables based on the shape parameter α:

$$\delta = \frac{\alpha}{\sqrt{1+\alpha^2}}, \qquad b = \sqrt{\frac{2}{\pi}}. \tag{52}$$

The mean and variance of the standard variable Z are then given by:

$$\mu_Z = \mathbb{E}[Z] = b\delta, \qquad \sigma_Z^2 = \mathrm{Var}\,[Z] = 1 - \mu_Z^2. \tag{53}$$

Using these properties, we can readily express the moments of Y to define the CP parameters μ, σ, and γ_1:

$$\mu = \mathbb{E}[Y] = \xi + \omega\mu_Z, \tag{54}$$

$$\sigma^2 = \mathrm{Var}\,[Y] = \omega^2\sigma_Z^2 = \omega^2(1 - \mu_Z^2), \tag{55}$$

$$\gamma_1 = \frac{4-\pi}{2}\left(\frac{\mu_Z}{\sqrt{1-\mu_Z^2}}\right)^3. \tag{56}$$

In a machine learning setting, the model directly holds the learnable CP parameters (μ, σ, γ_1), allowing a standard optimizer to independently tune one of those parameters without inadvertently altering the other two. However, to evaluate the probability density function during optimization, we must map them back to the Direct Parameterization (ξ, ω, α). This process involves simple algebraic manipulations to invert the equations above.

First, we must recover μ_Z from the target skewness γ_1. By defining the intermediate quantity $c = \left(\frac{2\gamma_1}{4-\pi}\right)^{1/3}$, we can compute:

$$\mu_Z = \frac{c}{\sqrt{1+c^2}}, \qquad \text{and} \qquad \delta = \frac{\mu_Z}{b}. \tag{57}$$

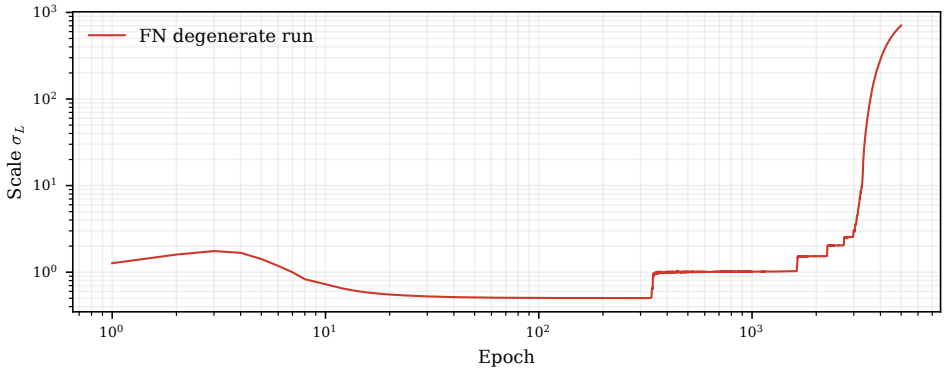
Finally, the direct parameters are obtained as:

$$\alpha = \frac{\delta}{\sqrt{1-\delta^2}}, \qquad \omega = \frac{\sigma}{\sqrt{1-\mu_Z^2}}, \qquad \xi = \mu - \omega\mu_Z. \tag{58}$$

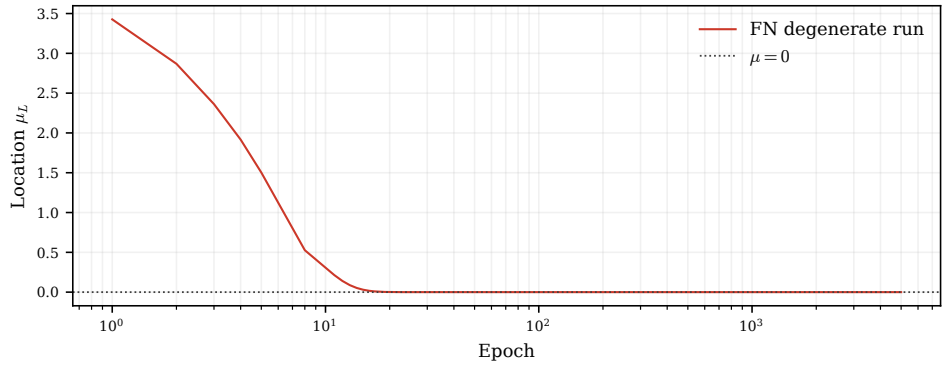
Note that since the theoretical range of γ_1 is approximately $[-0.9953, 0.9953]$ (Azzalini, 2005), we can safely limit the active range of the learnable parameter governing γ_1 by applying the $\tanh$ operator, possibly scaled by a threshold γ_{max}. This ensures that the predicted skewness remains strictly within the valid mathematical bounds.

D.4 Folded Normal: Gradient Analysis

When the Folded Normal was used to model depth, we repeatedly observed a structural failure mode in which the network continued to add layers throughout training. This phenomenon is illustrated in Figure 6, which depicts the FN parameters μ_L and σ_L over training epochs for one of the affected configurations trained on the SpiralHard dataset. The structural collapse of the Folded Normal is similar to the known optimization issues one can observe when using the Direct Parameterization (DP) of the Skew Normal (Arellano-Valle & Azzalini, 2008).



(a)



(b)

Figure 6: The figure reports learned scale (a) and location (b) parameters over epochs for a single, degenerate run using the FN distribution. The SpiralHard dataset was used to train the model and the configuration used the following hyperparameters: $\sigma_\theta = 10$, $\mu_\lambda = 0.1$, $\mu_{L_0} = 4.0$. The location parameter reached a minimum of approximately 10^-4.

In contrast, we empirically observed the Centered Parameterization of the Skew Normal (CP-SN) not to exhibit such unstable behavior on its μ_L parameter, as reported by Figure 7 for an equivalent configuration on the SpiralHard dataset. Furthermore, the CP-SN displayed stable depth adaptivity throughout all tested configurations.

To formalize the above observation, we show that, when adopting the FN, the gradient of the loss with respect to μ vanishes at the limiting location $\mu = 0$, effectively stalling the optimizer. Consider the

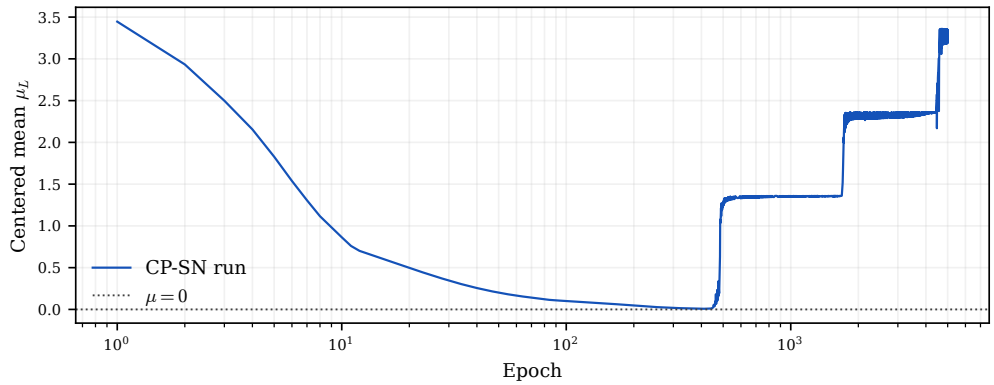


Figure 7: The figure reports learned mean (μ_L) parameters over epochs for a single run using the SN distribution on the SpiralHard dataset, matching the hyper-parameters of the FN configuration described in Figure 6. The μ_L parameter reached a minimumof 7.89×10^{-3}.

ELBO objective defined in Equation (14). The parameter μ influences the loss exclusively through the discrete variational probability distribution $q(L)$. By applying the chain rule, the gradient of the ELBO with respect to μ can be expressed as:

$$\frac{\partial \mathcal{L}}{\partial \mu} = \sum_{\hat{\ell}=0}^{M} \frac{\partial q(\hat{\ell})}{\partial \mu} \frac{\partial \mathcal{L}}{\partial q(\hat{\ell})}. \tag{59}$$

where M is the current finite truncation of the discrete depth distribution. Crucially, the entire gradient for μ depends on the derivative of $q(\hat{\ell})$. In Errica et al. (2025), the Folded Normal distribution is discretized as:

$$q(L) = p_{\text{FN}}(L; \mu, \sigma) \begin{cases} F_{\text{FN}}(1; \mu, \sigma) & \text{if } L = 0 \\ F_{\text{FN}}(L+1; \mu, \sigma) - F_{\text{FN}}(L; \mu, \sigma) & \forall L \in \mathbb{N}^+. \end{cases} \tag{60}$$

For any $L > 0$, the gradient of this discrete probability function is the integral of the derivative of the continuous density function f_{FN} over the interval $[L, L+1]$:

$$\frac{\partial}{\partial \mu} q(L) = \int_0^{L+1} \frac{\partial}{\partial \mu} f_{\text{FN}}(t; \mu, \sigma) dt - \int_0^{L} \frac{\partial}{\partial \mu} f_{\text{FN}}(t; \mu, \sigma) dt \tag{61}$$

$$= \int_L^{L+1} \frac{\partial}{\partial \mu} f_{\text{FN}}(t; \mu, \sigma) dt. \tag{62}$$

Substituting the definition of the Folded Normal density yields:

$$\frac{\partial}{\partial \mu} f_{\text{FN}}(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \frac{\partial}{\partial \mu} \left[\exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) + \exp\left(-\frac{(x+\mu)^2}{2\sigma^2} \right) \right] \tag{63}$$

$$= \frac{1}{\sqrt{2\pi\sigma^3}} \left[(x-\mu) \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) - (x+\mu) \exp\left(-\frac{(x+\mu)^2}{2\sigma^2} \right) \right]. \tag{64}$$

When evaluated exactly at $\mu = 0$, this gradient simplifies to:

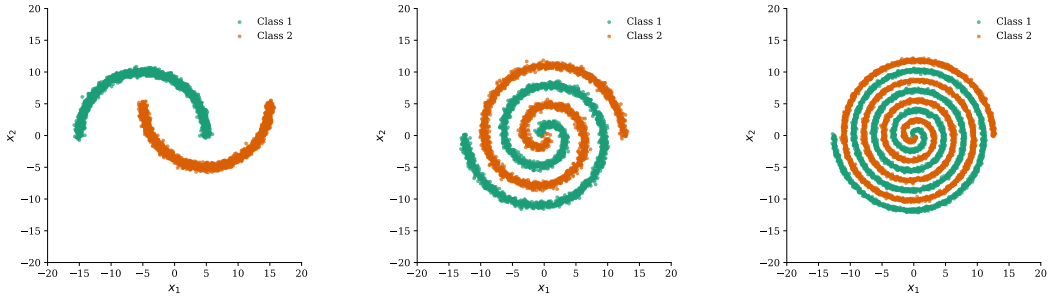
$$\frac{\partial}{\partial \mu} f_{\text{FN}}(x; \mu, \sigma) \Big|_{\mu=0} = \frac{1}{\sqrt{2\pi\sigma^3}} \left[x \exp\left(-\frac{x^2}{2\sigma^2} \right) - x \exp\left(-\frac{x^2}{2\sigma^2} \right) \right] = 0, \tag{65}$$

for all x.

Because the derivative of f_{FN} vanishes everywhere, the integral over any interval $[L, L+1]$ is zero (the case for $L = 0$ follows symmetrically). Consequently, $\frac{\partial q(\hat{\ell})}{\partial \mu} = 0$, which nullifies the ELBO gradient evaluated at $\mu = 0$. This theoretical result further corroborates our experimental observations: lacking a gradient for the location parameter at this limiting point, the optimizer can only increase the scale σ to reach deeper representations, triggering the observed scale explosion. The Centered Parameterization of the Skew Normal effectively bypasses this issue by decoupling the statistical moments, allowing the model to smoothly shift its depth without inadvertently flattening the probability distribution.

E EXPERIMENTAL DETAILS

E.1 DATASETS AND DATA PREPARATION



(a) (b) (c)

Figure 8: Scatter plots for the three synthetic datasets: DoubleMoon (a), Spiral (b), SpiralHard (c). Different colors represent different classes.

The experiments consider six binary classification datasets: three synthetic datasets and three real-world datasets. The synthetic datasets are generated by the dataset classes used in the implementation. *DoubleMoon* and *Spiral* contain 2,500 examples per class, whereas *SpiralHard* contains 5,000 examples per class and uses two spiral repetitions. See Figure 8 for a visual representation. The real-world datasets are loaded from OpenML or the UCI repository as indicated in Table 3.

For *MiniBooNE* and *CreditCards*, the implementation standardizes each feature using the mean and standard deviation of the loaded dataset, with 10^{-3} added to the denominator. The *POL* loader does not apply an additional feature transformation. The synthetic datasets are used in their generated numerical form.

For each dataset, two stratified partition schemes are generated. The inner split contains training and validation partitions and is used for model selection, whereas the outer split contains training, validation, and test partitions and is used for the final training and risk assessment. The partitions are generated using seed 42, and shuffling. The configured validation ratios are 0.1 for the inner split and 0.1 for the outer split, while the configured test ratio is 0.2. During model selection, each candidate configuration is trained and evaluated using the inner training and validation partitions. After selecting the best configuration, it is retrained from scratch using the outer training partition. The outer validation partition is used to monitor training and select the best checkpoint, while the outer test partition is reserved for final evaluation. Risk assessment is performed by repeating this final training procedure for the selected configuration across the configured number of independent runs, and evaluating each run on the outer test partition.

Table 3: Summary of the datasets used in the experiments. The number of samples is the total number of available data points before splitting.

Dataset	Type	# Features	# Samples	# Classes	Feature Type
DoubleMoon	Synthetic	2	5000	2	Numerical
Spiral	Synthetic	2	5000	2	Numerical
SpiralHard	Synthetic	2	10000	2	Numerical
POL	Real-world	48	15000	2	Numerical
CreditCards	Real-world	23	30000	2	Mixed
MiniBooNE	Real-world	50	130064	2	Numerical

E.2 PREDICTIVE EXPERIMENTS

Table 4 reports the full set of hyperparameters used and defines the search space for the model selection phase carried out by the experiment described in Section 4.

Table 4: Configuration space for the predictive experiments. Values listed in braces are the alternatives considered during model selection.

Category	Parameter	Value(s)
Architecture	Activation	ReLU6
	Minimum depth	1
	Minimum hidden width	2
	Width quantile	0.9
	Depth quantile	0.95
	Depth truncation	$1 \dots 20$
	Width boundary	0.0005
	Initial width rate	$\{0.1, 0.01\}$
	Depth distribution	{Skew Normal (SN), Folded Normal (FN)}
	SN initial mean (μ_L)	$\{1, 4\}$
	SN initial std (σ_L)	1.0
	SN initial skewness (γ_L)	$\{-0.5, 0.5\}$
	SN skew range	$-0.9 \dots 0.9$
	FN Location	$\{1.0, 4.0\}$
	FN Scale	1.0
	SN/FN Minimum σ	0.5
	Weight-prior placement	Full model
	Weight-prior scale (σ_θ)	$\{1, 10, \text{null}\}$
	Width-prior mean (μ_λ)	$\{0.1, 0.01, \text{null}\}$
	Width-prior scale (σ_λ)	0.5
	Depth prior	$\text{Poisson}(1.0)$
Optimization	Optimizer	Adam
	Learning rate	$\{0.001, 0.01\}$
	Weight decay	0.0
	Batch size	128
	Maximum epochs	5,000
	Early-stopping patience	500
	Model-selection runs	2
	Risk-assessment runs	10

E.3 STABILITY EXPERIMENTS

The stability experiments are a separate protocol from the predictive evaluation. They use only *SpiralHard* and study the trajectories of the learned topology as a function of initialization, batch size, readout type, and weight-prior placement. The stability grid varies the batch size, the initial Exponential width rate, the initial depth mean, the readout type, and whether the weight prior is applied to the readout, the full model, or the backbone. These runs use a centered Skew Normal depth distribution with zero initial skewness.

Stability runs use Adam with learning rate 0.01, no weight decay, and a maximum of 20,000 epochs, with no early stopping criterion.

The stability grid uses three model-selection runs. The complete configuration space is given in Table 5, which comprises 1080 possible configurations.

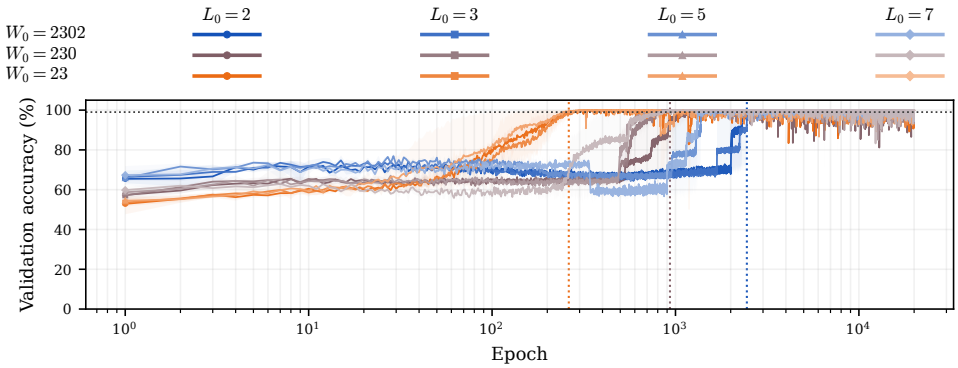
Table 5: Configuration space for the stability experiments on *SpiralHard*. This table is separate from the predictive-experiment configuration in Table 4.

Category	Parameter	Value(s)
Architecture	Activation	ReLU6
	Minimum depth	1
	Minimum hidden width	2
	Width quantile	0.9
	Depth quantile	0.95
	Depth truncation	$1 \dots 20$
	Width boundary	0.0005
	Initial width rate	$\{0.1, 0.01, 0.001\}$
	SN initial mean (μ_L)	$\{1, 2, 4, 6\}$
	SN Initial std (σ_L)	0.0
	SN Initial skewness (γ_L)	0.0
	SN skew range	$-0.9 \dots 0.9$
	Weight-prior placement	{Full model, readout only, backbone}
	Weight-prior scale (σ_θ)	$\{1, 10, \text{null}\}$
	Width-prior mean (μ_λ)	null
	Width-prior scale (σ_λ)	null
	Readout type	{MLP, Adaptive linear}
Optimization	Batch size	$\{32, 128, 256, 512, 1024\}$
	Optimizer	Adam
	Learning rate	0.01
	Weight decay	0.0
	Maximum epochs	20,000
	Model-selection runs	3
	Risk-assessment runs	1

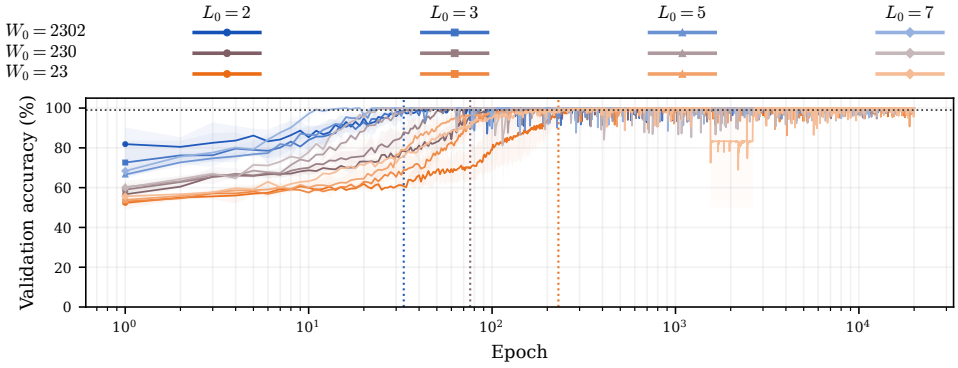
F TOPOLOGICAL STABILITY

In this section we report additional experimental results obtained through the extensive stability analysis described in Section E.3. Unless otherwise stated, the plots use the three model-selection runs for each SpiralHard configuration, of which we retain only the ones that reached at least 90% validation accuracy. Furthermore, we only consider runs with a batch size of 128 and for which the weight regularization is applied to the whole network.

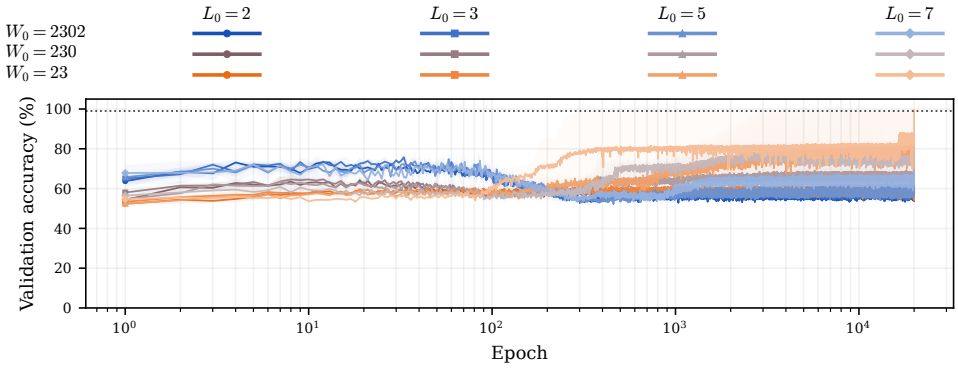
Effects of Regularization on Predictive Performance We report convergence results for the retained stability configurations over all the tested regularization regimes. Figure 9 shows SOAP's validation accuracy convergence over training epochs for various initial configurations of the model, grouped by values of σ_θ. Interestingly — when weight regularization is applied with $\sigma_\theta = 10$ — convergence is seemingly delayed for configurations with a higher initial hidden neuron count, as shown in Figure 9a. Moreover, the number of epochs required to reach 99% validation accuracy seems to reduce as the number of the initially allocated hidden layers increases, suggesting that for this particular task and regularization setting, deeper and narrower networks might exhibit a better optimization behavior—as we expected.



(a) $\sigma_\theta = 10$



(b) $\sigma_\theta = +\infty$



(c) $\sigma_\theta = 1$

Figure 9: The plots depict the average validation accuracy obtained by SOAP configurations on the SpiralHard dataset for various regularization settings, namely $\sigma_\theta = 1$ (c), $\sigma_\theta = 10$ (a) and $\sigma_\theta = +\infty$ (b) respectively. Runs were filtered to only include MLP readouts and weight regularization applied to the whole network. Batch size is fixed to 128. Configurations are visually grouped by their initial total width W_0, with lighter colors in the same group indicating a higher value of initial depth L_0. Vertical dotted bars indicate the first epoch at which the average validation accuracy for a group reached 99%. The horizontal dotted line marks the 99% accuracy threshold on the y-axis.

By contrast, despite generally converging more quickly, when no weight regularization is applied (i.e. $\sigma_\theta = +\infty$), SOAP shows the opposite behavior. Initially wider layers correspond to longer optimization trajectories towards convergence, as depicted in Figure 9b. Nonetheless, the same relation with initial depth seems to hold also for this regularization regime, showing that deeper networks tend to converge faster compared to initially shallow ones.

Finally, Figure 9c also shows the validation accuracy curves obtained by configurations in the strongly regularized regime (i.e., corresponding to $\sigma_\theta = 1$). All tested configurations except one failed to reach the 99% validation accuracy threshold, possibly suggesting that the regularization effect dominates the other terms of the loss function. However, despite runs failing to converge to good local minima, a similar pattern to the $\sigma_\theta = 10$ case can be seen, confirming that, in regularized regimes, smaller SOAP networks might be easier to optimize.

Table 6: Predictive and architectural metrics for SOAP across weight-prior scales on SpiralHard. Results pool all batch-128 MLP configurations across four initial depth means and three initial width rates. Each group contains 36 runs (12 configurations with three runs each); values report the mean and standard deviation (in parentheses).

σ_θ	Accuracy Mean	Depth	Width	Readout Width
1	65.1 (12.5)	1 (1)	37 (55)	68 (115)
10	98.5 (8.3)	5 (3)	87 (22)	92 (63)
$+\infty$	100.0 (0.1)	6 (1)	2567 (2954)	1240 (1253)

Effects of Regularization on Topology. Removing the weight prior accelerates convergence but produces disproportionate capacity growth compared to the regularized regime, while improving final validation accuracy by only 1.52 percentage points—as reported in Table 6.

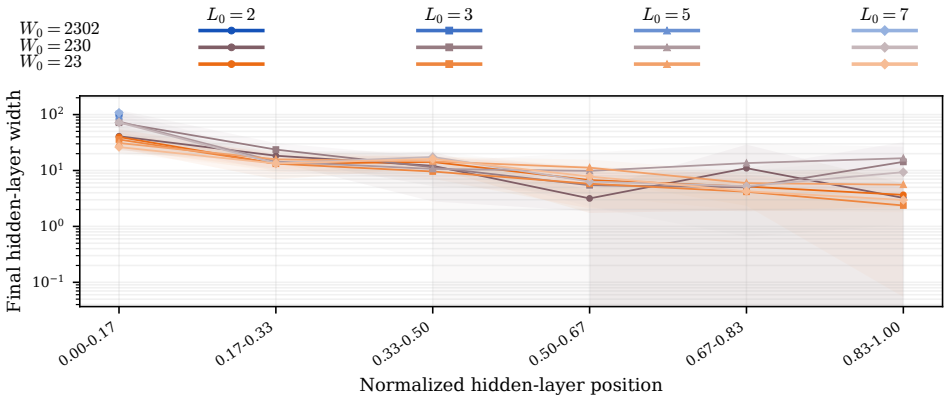


Figure 10: The plotted lines show how neurons are allocated for each hidden layer of the learned SOAP models for the retained configurations, with $\sigma_\theta = 10$. Final learned widths are plotted against normalized ranges of layer-indices, so that runs with different final depths can be compared. Solid lines represent individual configurations, which are visually grouped by their initial total width W_0, with lighter colors within a group indicating a higher initial depth L_0.

To characterize the role of the regularization prior, we therefore study its effects on the final width and depth distributions when $\sigma_\theta = 10$. Figure 10 reports the distribution of neurons over normalzed layer positions, and shows a general width-decreasing trend over hidden layers across configurations. The result is the model dynamically allocating more capacity toward the input layer of the network during training. This is in direct contrast with the results reported by Figure 2, which instead show a larger proportion of neurons allocated deeper into the network. Compared to the unregularized networks produced by model selection, the stronger weight prior might influence the final representational capacity of the model, effectively introducing a soft complexity-budget. This effect results in SOAP favoring a particular allocation strategy depending on the total hidden width it can afford to create.

Finally, Figure 11 depicts a complementary view of the width distribution analysis by dividing the range of possible hidden widths into discrete bins and reporting the fraction of hidden layers whose final width fall into those sub-intervals. Interestingly, as we grow the number of starting neurons, the

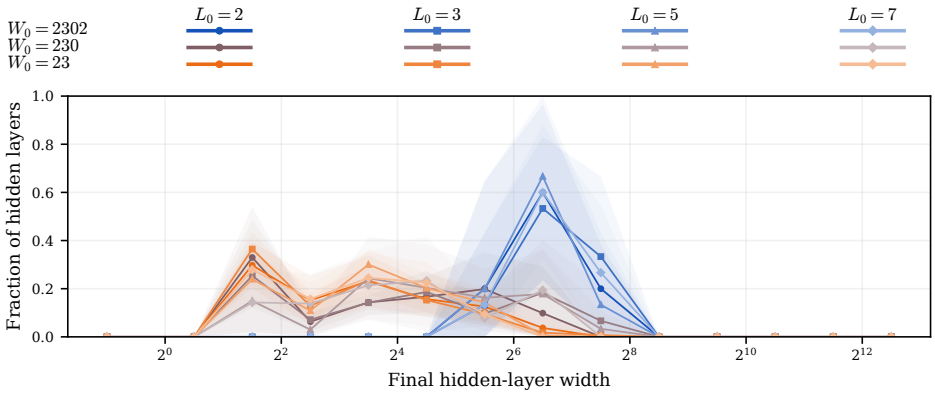


Figure 11: Solid lines represent the fraction of hidden layers falling into logarithmically spaced final-width bins. Starting total width and depth are denoted by W_0 and L_0, respectively.

center of mass of the depicted distribution shifts to the right, eventually resulting in a single major peak around a single layer.

Depth Distribution Variations. By analizing the learned depth-related parameters μ_L, σ_L and γ_L over different configurations, we can further assess the adaptivity behavior of SOAP. Figure 12 plots the difference for each of the learned statistical moments of $q(L)$ compared to their initial settings, over different batch sizes and grouped by initial exponential rate ν. While marginally so, larger batch sizes seem to result in a final topology closer to its initial setting, possibly suggesting that strong structural updates need more iterations to settle, as the batch size grows. Conversely, the standard deviation of $q(L)$ always collapses to its minimum allowed value of 0.5 across every tested scenario. The skewness parameter instead seems to be the most flexible, systematically oscillates over the whole allowed range of $(-0.9, 0.9)$ before settling, with clear trends depending on the initial configuration of the model. In particular, the average skewness goes from strongly positive (right-skew) to strongly negative (left-skew) as the number of initially allocated neurons grows.

Role of Batch Size. The effect of the batch size on the adaptive behavior of SOAP can be seen by examining Figure 13 (Left): the final total width of the model grows as the batch size increases and shows less deviation across configurations. Althought the actual mechanism behind this phenomenon is uncertain, it is somewhat in line with the convergence behavior of Figure 3, in which a common width-decreasing trend emerges as training iterations go on. When the learning rate is fixed, in fact, a larger batch size implies a fewer number of structural updates in SOAP, which might explain the overall increasing number of hidden units for larger batches.

The relation between batch size and depth— which is reported in Figure 13 (Right)— is less evident and no clear pattern can be extracted. Configurations with a high number of initial neurons ($\nu = 0.001$) show a flat curve around $L = 1$ and are not influenced by the increasing batch size, with the other two groups of runs showing opposite trends to each other.

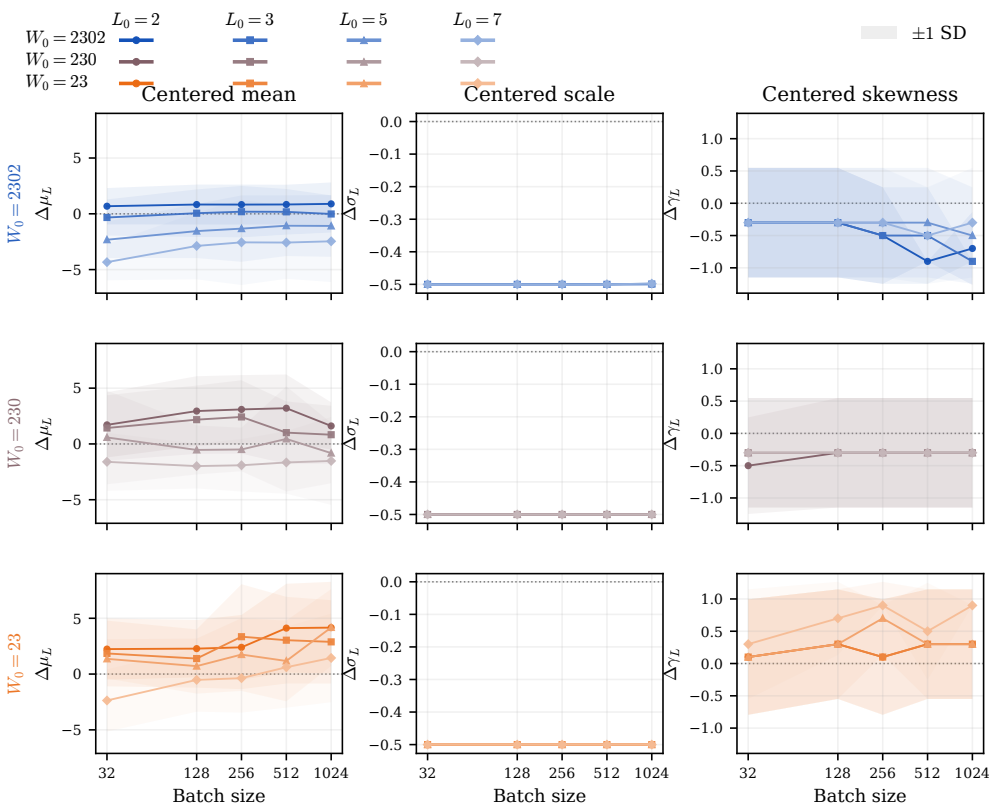


Figure 12: The three depicted panels show changes in the learned centered Skew Normal parameters relative to their initial values: $\Delta\mu_L$, $\Delta\sigma_L$, and $\Delta\gamma_L$, across all tested batch sizes and values of σ_θ. Starting total width and depth are denoted by W_0 and L_0, respectively.

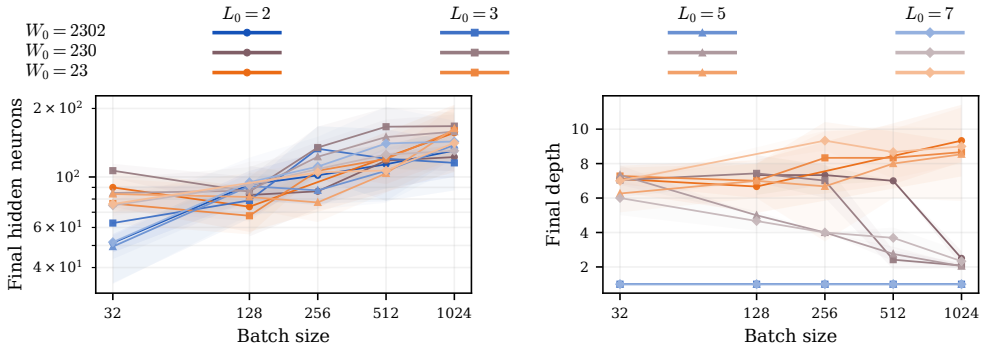


Figure 13: The two panels show final hidden-neuron count (Left) and final depth (Right) against different batch sizes. Starting total width and depth are denoted by W_0 and L_0, respectively.