Variational Inference for Infinitely Deep Neural Networks

Achille Nazaret [1] **David Blei** [1 2]

Abstract

We introduce the *unbounded depth neural network* (UDN), an infinitely deep probabilistic model that adapts its complexity to the training data. The UDN contains an infinite sequence of hidden layers and places an unbounded prior on a truncation ℓ, the layer from which it produces its data. Given a dataset of observations, the posterior UDN provides a conditional distribution of both the parameters of the infinite neural network and its truncation. We develop a novel variational inference algorithm to approximate this posterior, optimizing a distribution of the neural network weights and of the truncation depth ℓ, and without any upper limit on ℓ. To this end, the variational family has a special structure: it models neural network weights of arbitrary depth, and it dynamically creates or removes free variational parameters as its distribution of the truncation is optimized. (Unlike heuristic approaches to model search, it is solely through gradient-based optimization that this algorithm explores the space of truncations.) We study the UDN on real and synthetic data. We find that the UDN adapts its posterior depth to the dataset complexity; it outperforms standard neural networks of similar computational complexity; and it outperforms other approaches to infinite-depth neural networks.

1. Introduction

Deep neural networks have propelled research progress in many domains (Goodfellow et al., 2016). However, selecting an appropriate complexity for the architecture of a deep neural network remains an important question: a network that is too shallow can hurt the predictive performance, and a network that is too deep can lead to overfitting or to unnecessary complexity.

In this paper, we introduce the *unbounded depth neural network* (UDN), a deep probabilistic model whose size adapts to the data, and without an upper limit. With this model, a practitioner need not be concerned with explicitly selecting the complexity for her neural network.

A UDN involves an infinitely deep neural network and a latent truncation level ℓ, which is drawn from a prior. The infinite neural network can be of arbitrary architecture and interleave different types of layers. For each datapoint, it generates an infinite sequence of hidden states $(h_k)_{k\geq 1}$ from the input x, and then uses the hidden state at the truncation h_ℓ to produce the response y. Given a dataset, the posterior UDN provides a conditional distribution over the neural network's weights and over the truncation depth. A UDN thus has access to the flexibility of an infinite neural network and, through its posterior, can select a distribution of truncations that best describes the data. The model is designed to ensure that ℓ has no upper limit.

We approximate the posterior UDN with a novel method for variational inference. The method uses a variational family that employs an infinite number of parameters to cover the whole posterior space. However, this family has a special property: Though it covers the infinite space of all possible truncations, each member provides support over a finite subset. With this family, and thanks to its special property, the variational objective can be efficiently calculated and optimized. The result is a gradient-based algorithm that approximates the posterior UDN, dynamically exploring its infinite space of truncations and weights.

We study the UDN on real and synthetic data. We find that (i) on synthetic data, the UDN achieves higher accuracy than finite neural networks of similar architecture (ii) on real data, the UDN outperforms finite neural networks and other models of infinite neural networks (iii) for both types of data, the inference adapts the UDN posterior to the data complexity, by exploring distinct sets of truncations.

In summary, the contributions of this paper are as follows:

- We introduce the unbounded depth neural network: an infinitely deep neural network which can produce data from any of its hidden layers. In its posterior, it adapts its truncation to fit the observations.

[1]Department of Computer Science, Columbia University, New York, USA [2]Department of Statistics, Columbia University, New York, USA. Correspondence to: Achille Nazaret <achille.nazaret@columbia.edu>.

Proceedings of the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, PMLR 162, 2022.

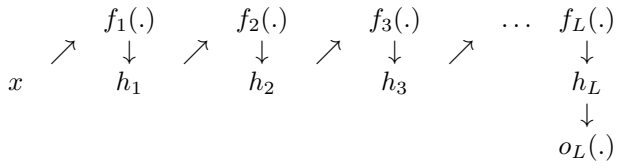
- We propose a variational inference method with a novel variational family. It maintains a finite but evolving set of variational parameters to explore the unbounded posterior space of the UDN parameters.
- We empirically study the UDN on real and synthetic data. It successfully adapts its complexity to the data at hand. In predictive performance, it outperforms other finite and infinite models.

This work contributes to research in architecture search, infinite neural networks, and unbounded variational families. Section 5 provides a detailed discussion of related work.

2. Unbounded depth neural networks

We begin by introducing the family of unbounded-depth neural networks. Let $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ be a dataset of n labeled observations.

Classical neural networks. A classical neural network of depth L chains L successive functions $(f_\ell)_{1\leq\ell\leq L}$, eventually followed by an output function o_L.



Each h_ℓ is called a *hidden state* and each f_ℓ is called a *layer*. Each layer is usually composed of a linear function, an activation function, and sometimes other differentiable transformations like *batch-normalization* (Ioffe & Szegedy, 2015). In deep architectures, f_ℓ can refer to a *block* of layers, such as a succession of 3 convolutions in a Resnet (He et al., 2016) or a dense layer followed by attention in transformers (Vaswani et al., 2017).

We fix a *layer generator* f which returns the layer f_ℓ for each integer ℓ. The layer generator can return layers of different shapes or types for different ℓ as long as two consecutive layers can be chained by composition. Similarly, we fix an *output generator* o_L which transforms the last hidden state h_L into a parameter suitable for generating a response variable. We write θ_ℓ for the parameters of f_ℓ and incorporate those of o_L into θ_L. With this notation, a finite neural network of depth L, generated by (f, o), is written as

$$\Omega_L = o_L \circ f_L \circ f_{L-1} \circ ... \circ f_2 \circ f_1$$

and has parameters $(\theta_1, ..., \theta_L)$.

Finally, we fix a distribution p parametrized by the output of the neural network $\Omega_L(x_i; \theta_{1:L})$, and use it to model the responses conditional on the input:

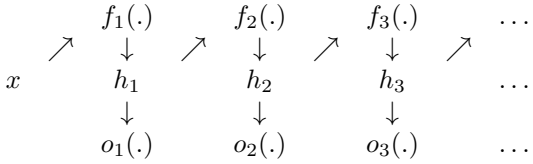
$$\forall i, \quad y_i|x_i, \theta_{1:L} \sim p(y_i; \Omega_L(x_i; \theta_{1:L})).$$

Given a dataset $\{x_i, y_i\}_{i=1}^N$ and Gaussian priors on $\theta_{1:L}$, MAP estimation of the neural network parameters corresponds to classical methods for fitting a neural network with weight decay (Neal, 1996). In this paradigm, the form of p is related to the loss function (Murphy, 2012).

This classical model requires that the layers be set in advance. Its flexibility and ability to capture the data depend crucially on the selection of L. Too large and the model is too flexible, and can overfit. Too small and the model is not flexible enough. It is appealing to consider a model that can adapt its depth to the data at hand.

Unbounded depth neural networks. We extend the finite construction above to formulate an unbounded depth neural network (UDN). We consider an infinite sequence of hidden states $(h_k)_{k\geq 1}$ generated by $(f_\ell)_{\ell\geq 1}$ and parametrized by an infinite set of weights $\theta \triangleq (\theta_\ell)_{\ell\geq 1}$.

A challenge to conceptualizing an infinite-depth neural network is where to hang the observation. What we do is posit a possible output layer o_ℓ after each layer of hidden units.



We then add an additional parameter, a truncation level ℓ, to determine which o_ℓ will generate the response.

The complete UDN models the truncation level ℓ as an unobserved random variable with a prior μ. Along with a prior ρ on the weights, the UDN defines a generative model with an infinite-depth neural network:

$$\begin{aligned} \theta &\sim \rho(\theta) && \triangleright \text{ network weights} \\ \ell &\sim \mu(\ell) && \triangleright \text{ truncation} \\ y_i|x_i, \theta, \ell &\sim p(y_i; \Omega_\ell(x_i; \theta)) && \triangleright \text{ response} \end{aligned}$$

This generative process is represented in figure 1. If the truncation prior μ puts a point mass at L then the model is equivalent to the classical finite model of depth L. But with a general prior over all integers, the posterior UDN has access to all depths of neural networks.

The independence of the priors is important. The model does not put a prior on ℓ and then samples the weights conditional on it. Rather, it first samples a complete infinite neural network, and then samples the finite truncation to produce its data. What this generative structure implies is that different truncations will share the same weights on their shared layers. As we will see in section 4, this property leads to efficient calculations for approximate posterior inference.

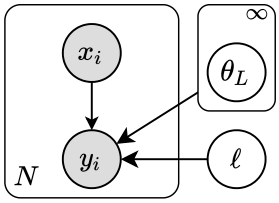


Figure 1. Graphical model for the unbounded depth neural network.

3. Variational Inference for the UDN

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, the goal of Bayesian inference is to compute the posterior UDN $p(\theta, \ell \mid \mathcal{D})$. The exact posterior is intractable, and so we appeal to variational inference (Jordan et al., 1999; Wainwright & Jordan, 2008; Blei et al., 2017). In traditional variational inference, we posit a family of approximate distributions over the latent variables and then try to find the member of that family which is closest to the exact posterior.

The unbounded neural network, however, presents a significant challenge to this approach—the depth ℓ is unbounded and the number of latent variables θ is infinite. To overcome this challenge, we will develop an unbounded variational family $q(\theta, \ell)$ that is still amenable to variational optimization. With the algorithm we develop, the "search" for a good distribution of truncations is a natural consequence of gradient-based optimization of the variational objective.

3.1. Structure of the variational family

We define a joint variational family that factorizes as $q(\theta, \ell) = q(\ell)q(\theta|\ell)$, and note that the factor for the neural network weights depends on the truncation. We introduce the parameters λ, ν and detail the structure of the families $\{q(\ell; \lambda)\}$ and $\{q(\theta|\ell; \nu)\}$.

The unbounded variational family with connected and bounded members $q(\ell; \lambda)$. For a truly unbounded procedure, we require that the variational family over ℓ should be able to explore the full space of truncations $\mathbb{N}^*$. Simultaneously, since the procedure must run in practice, each distribution $q(\ell; \lambda)$ should be tractable, that is $\mathbb{E}_{q(\ell)}[g(\ell)]$ can be computed efficiently for any g.

A sufficient condition for tractable expectations is that $q(\ell; \lambda)$ has finite support; the expectation becomes the finite sum $\sum_{i \in \text{support}(q)} q(i; \lambda)g(i)$. However, to be able to explore the unbounded posterior space $\mathbb{N}^*$, the variational family $\{q(\ell; \lambda)\}$ itself cannot have finite support. It should contain distributions covering all possible truncations ℓ. Moreover, it should be able to navigate continuously between these distributions.

We articulate these conditions in the following definition:

Definition 3.1. A variational family $\mathcal{Q} = \{q(\lambda)\}$ over $\mathbb{N}^*$ is *unbounded with connected and bounded members* if

(1) $\forall q \in \mathcal{Q}$, support$(q)$ is bounded

(2) $\forall L \in \mathbb{N}^*$, $\exists q \in \mathcal{Q}$, $L \in \text{argmax}(q)$

(3) The parameter λ is a continuous variable.

Echoing the discussion above, there are several consequences to this definition:

- By (1), each q has a finite support. We write the maximal value $m(q) := \max\{\ell | q(\ell) > 0\}$.
- Thanks to (2), the approximate posterior can place its main mass around any ℓ. That is, $\mathcal{Q}$ covers the space of all possible truncations: $\bigcup_{q \in \mathcal{Q}} \text{support}(q) = \mathbb{N}^*$.
- Condition (3) ensures that $\mathcal{Q}$ not only contains members with mass on any ℓ, but it can continuously navigate between them. This condition is important for optimization.

The nested family $q(\theta|\ell; \nu)$. In the UDN model, conditional on ℓ, the response y depends only on the first ℓ layers and not the subsequent ones. Thus the exact posterior $p(\theta|\ell)$ only contains information from the data for θ_i up to $i \leq \ell$; the posterior of the θ_i with $i > \ell$ must match the prior.

We mirror this structure in the variational approximation,

$$q(\theta|\ell; \nu) = q(\theta_{1:\ell}; \nu_{1:\ell}) \prod_{k=\ell+1}^{\infty} p(\theta_k). \tag{4}$$

Kurihara et al. (2007b) also introduce a family with structure as in (4), which they call a *nested variational family*.

The evidence lower bound. The full variational distribution combines the *unbounded variational family* of Definition 3.1 with the *nested family* of (4), $q(\ell, \theta) = q(\ell; \lambda)q(\theta|\ell, \nu)$. With this distribution, we can now derive the optimization objective.

Variational inference seeks to minimize the KL divergence between the variational posterior q and the exact posterior p. This is equivalent to maximizing the variational objective (Bishop, 2006), which is commonly known as the Evidence Lower BOund (ELBO). Because of the factored structure of the variational family, we organize the terms of the ELBO with iterated expectations,

$$\mathcal{L}(q) = \mathbb{E}_{q(\ell,\theta)}[\log p(Y, \ell, \theta|X) - \log q(\ell, \theta)] \tag{5}$$

$$= \mathbb{E}_{q(\ell)}\left[\log \tfrac{p(\ell)}{q(\ell)} + \mathbb{E}_{q(\theta|\ell)}\left[\log \tfrac{p(\theta)}{q(\theta|\ell;\nu)} + \textstyle\sum_{i=1}^n \log p(y_i \mid \ell, \theta, x_i)\right]\right]. \tag{6}$$

Further, using the special structure of this variational family, the ELBO can be simplified:

- The factor $q(\theta|\ell)$ satisfies the nested structure condition (4) so $\frac{p(\theta)}{q(\theta|\ell;\nu)} = \frac{p(\theta_{1:\ell})}{q(\theta_{1:\ell};\nu_{1:\ell})}$. This quantity only involves a finite number ℓ of parameters and variables even if the prior and posterior were initially over all the variables.
- The factor $q(\ell)$ satisfies (1). The outer expectation of $\mathcal{L}(q)$ can be explicitly computed.

With these two observations, we rewrite the ELBO:

$$\mathcal{L}(q) = \sum_{\ell=1}^{m(q)} q(\ell;\lambda)\Bigg[\log\tfrac{p(\ell)}{q(\ell;\lambda)} + \mathbb{E}_{q(\theta|\ell;\nu)}\Bigg[\sum_{k=1}^{\ell}\log\tfrac{p(\theta_k)}{q(\theta_k;\nu_k)} + \sum_{i=1}^{n}\log p(y_i;\Omega_\ell(x_i;\theta))\Bigg]\Bigg]. \quad (7)$$

Notice this equation expresses the ELBO using only a finite set of parameters $\lambda, \nu_{1:m(q(\lambda))}$, which we call *active* parameters. Thus we can compute the gradient of the ELBO with respect to $(\lambda, \nu_{1:\infty})$, since only the coordinates corresponding to the active parameters $\lambda, \nu_{1:m(q)}$ can be nonzero. This fact allows us to optimize the variational distribution.

Dynamic variational inference. In variational inference we optimize the ELBO of equation (7). We use gradient methods to iteratively update the variational parameters $(\lambda, \nu_{1:\infty})$. Equation (7) just showed how to take one efficient gradient step, by only updating the active parameters, those with nonzero gradients. From there, a succession of gradient updates becomes possible and still involves only a finite set of parameters. Indeed, even if the special property (2) of the variational family guarantees that $q(\ell;\lambda)$ can place mass on any ℓ, and by doing so, can activate any parameter ν_ℓ during the optimization, successive updates of finitely many parameters will still only affect finitely many parameters.

For instance, the inference can start with $m(q(\lambda)) = 5$ active layers, and increase to $m(q(\lambda)) = 6$ after an update to λ that favors a deeper truncation. The next gradient update of (λ, ν) will then affect ν_6, which was not activated earlier. At any iteration, the ELBO involves a finite subset of the parameters, but this set of active parameters naturally grows or shrinks as needed to approach the exact posterior.

Because the subset of active variational parameters evolves during the optimization, we refer to the method of combining the *unbounded variational family* of Definition 3.1 with the *nested family* of (4), as *dynamic variational inference*. We detail in section 4 how to run efficient computations when we do not know in advance which variational parameters will be activated during the optimization.

3.2. One explicit choice of variational family and prior

We end the inference section by proposing priors for (ℓ, θ) and an explicit variational family satisfying the unbounded family conditions (1, 2, 3) and the nested structure (4).

Choice of prior. We use a standard Gaussian prior over all the weights θ. We set a Poisson(α) prior for ℓ. More precisely, we have $\ell - 1 \sim \text{Poisson}(\alpha)$ because $\ell > 0$. The mean α is detailed in the experiment details in the appendix.

Choice of family $q(\ell;\lambda)$. To obtain the unbounded family from definition 3.1, we adapt the Poisson family $\mathcal{P} = \{\text{Poisson}(\lambda) \mid \lambda > 0\}$ by truncating each individual distribution $q(*;\lambda) = \text{Poisson}(\lambda)$ to its δ-quantile.

$$q^\delta(\ell;\lambda) \propto q(\ell;\lambda)\mathbb{1}[\ell \le \delta\text{-quantile}(q(*;\lambda))]$$

This forms the Truncated Poisson family $\mathcal{TP}(\delta) = \{q^\delta \mid q \in \mathcal{P}\}$ and the following holds:

Theorem 3.2. *For any $\delta \in [0.5, 1[$, $\mathcal{TP}(\delta)$ is unbounded with connected bounded members. For $\delta = 0.95$, we have*

- $\lambda - \ln 2 \le m(q^{0.95}(\lambda)) \le 1.3\lambda + 5$ (8)
- $\forall n \in \mathbb{N}^*,\ \ n \in \text{argmax}(q^{0.95}(n + 0.5))$ (9)
- $\lambda > 0$ *is a continuous parameter.* (10)

Inequalities (8) are shown in appendix using Poisson tail bounds from Short (2013) and bounds on the Poisson median from Choi (1994). It shows that $q^{0.95}(\lambda)$ satisfies the bounded support condition (1) with a support growing linearly in λ. The result (9) offers explicit distributions in $\mathcal{TP}(0.95)$ that satisfy the unbounded family condition (2). Finally, (10) ensures the continuity condition (3). During inference, δ is set to 0.95 and λ is a variational parameter.

Choice of family $q(\theta \mid \ell;\nu)$. The variational posterior on θ controls the neural network weights. In the nested structure (4), we model $q(\theta_{1:\ell};\nu_{1:\ell})$ with a mean-field Gaussian.[1]. With the prior defined earlier, $q(\theta|\ell;\nu)$ becomes

$$q(\theta|\ell;\nu) = \mathcal{N}(\nu_{1:\ell}, I_\ell)[\theta_{1:\ell}] \prod_{k=\ell+1}^{\infty}\mathcal{N}(0,1)[\theta_k].$$

We approximate $\mathbb{E}_{q(\theta|\ell)}[g(\theta_{1:\ell})]$ at the first order with $g(\mathbb{E}_{q(\theta|\ell)}[\theta_{1:\ell}]) = g(\nu_{1:\ell})$ for any g.

Predictions. The UDN can predict the labels of future data, such as held-out data for testing. It uses the learned variational posterior $q(\ell, \theta; \lambda, \nu)$, to approximate the predictive distribution of the label y' of new data x' as

$$p(y' \mid x', \mathcal{D}) \approx \mathbb{E}_{q(\ell,\theta;\lambda,\nu)}[p(y';\Omega_\ell(x';\theta_{1:\ell}))] \approx \sum_{\ell=1}^{m(q)} q(\ell;\lambda)\cdot p(y';\Omega_\ell(x';\theta_{1:\ell})). \quad (11)$$

[1] Some literature uses a point mass for the variational distribution of global parameters like neural network weights (Kingma & Welling, 2014; Author, 2021) Here, however, this choice would cause problems because it creates a mixed discrete-continuous family that leads to discontinuities in the objective.

The predictive distribution forms an ensemble of different truncations, that is discovered during the process of variational inference. This is related to Antoran et al. (2020).

4. Efficient algorithmic implementation

We review the computational aspects of both the model and the associated dynamic variational inference.

Linear complexity in $m(q)$. Evaluating the ELBO (7) or evaluating the predictive distribution (11) requires to compute the output of $m(q)$ different neural networks, Ω_1 to $\Omega_{m(q)}$. However, most of the computations can be shared (Antoran et al., 2020). We calculate the hidden layers sequentially up to $h_{m(q)}$, as they are needed to compute $\Omega_{m(q)}(x)$. We then apply the output layer o_ℓ to each hidden layer h_ℓ and obtain the collection $\{\Omega_\ell(x)\}_{\ell=1}^{m(q)}$. Hence, computing $\Omega_{m(q)}(x)$ alone or the whole collection $\{\Omega_\ell(x)\}_{\ell=1}^{m(q)}$ has the same complexity in $m(q)$.

Lazy initialization of the variational parameters. To compute gradients of the ELBO (7), we leverage modern libraries of the Python language for automatic differentiation. As discussed in section 3, the gradient of the ELBO only involves a finite set of active parameters, yet, this set can potentially reach any size during the optimization. Hence, the ELBO can depend on every possible variational parameters $(\lambda, \nu_{1:\infty})$, not all of which can be instantiated.

In libraries like *Tensorflow 1.0* (Abadi et al., 2015), the computational graph is defined and compiled in advance. This prevents the dynamic creation of variational parameters. In contrast, a library like PyTorch (Paszke et al., 2019) uses a dynamic graph. With this capability, new parameters ν_ℓ and layers f_ℓ can be created only when needed and the computational graph be updated accordingly. Before each ELBO evaluation, we compute the support of the current $q(\ell; \lambda)$ and adjust the variational parameters. The full dynamic variational inference procedure is presented in Algorithm 1.

Algorithm 1 Dynamic variational inference for the UDN

Input: data X, Y; architecture generators f, o;
Initialize: λ
$hidden\_layers, output\_layers = [], []$
for $epoch = 1$ **to** T **do**
  Compute $m(q(\lambda))$
  while $L := |hidden\_layers| < m(q(\lambda))$ **do**
    Add new layer $f(L+1)$ to $hidden\_layers$
    Add new layer $o(L+1)$ to $output\_layers$
    Initialize ν_{L+1}
  end while
  Compute $\mathcal{L}(q)$ in a single forward pass
  Compute gradients $\nabla_{\lambda, \nu_{1:m(q(\lambda))}} \mathcal{L}(q)$
  Update $\lambda, \nu_{1:m(q(\lambda))}$
end for

5. Related work

Neural architecture search Selecting a neural network architecture is an important question. Several methods have been proposed to answer it.

Bayesian optimization (Bergstra et al., 2013; Mendoza et al., 2016) and reinforcement learning (Zoph & Le, 2017) can tune the architecture as a hyperparameter. These algorithms propose successive architectures, which are then evaluated and compared. These methods decouple the architecture search and the model training, which increases the computational complexity. Reusing parameters across runs (Pham et al., 2018; Luo et al., 2018) can speed up the search but cannot avoid multiple trainings.

Dikov & Bayer (2019); Ghosh et al. (2019) jointly learn the network weights and the architecture as a single model. This is similar to what the UDN can do. However, the architecture they learn can only be reduced from an initial candidate by masking some of its parts. This relates to network pruning methods, which remove connections of small weight (LeCun et al., 1990; Hassibi & Stork, 1992).

Closely related to our work is Antoran et al. (2020), which combines predictions from different depths of the same model into a deep ensemble (Lakshminarayanan et al., 2017) to improve its uncertainty calibration. Here, a maximal depth is specified by the user, and only networks of smaller depths are used. Setting a very high value may lead to unnecessary computational complexity. In contrast, the UDN can grow during training, only if necessary.

Unbounded variational inference and Bayesian nonparametrics. An important class of models with an infinite number of latent variables is the class of Bayesian nonparametric (BNP) models (Hjort et al., 2010). BNP models adapt to the data complexity by growing the number of latent variables as necessary during posterior inference.

In a BNP model, variational inference does not usually operate in the unbounded latent space. Instead, proposed methods truncate either the model itself or the posterior variational family to a finite complexity (Blei & Jordan, 2006; Kurihara et al., 2007a; Doshi et al., 2009; Ranganath & Blei, 2018; Moran et al., 2021).

To relax this restriction, some methods propose split-merge heuristics to create or remove latent variables during training (Kurihara et al., 2007b; Zhai & Boyd-Graber, 2013; Hughes & Sudderth, 2013). In contrast, the dynamic variational inference proposed in the paper is a variational approach in which the creation or removal of variational parameters comes naturally from the variational family structure and gradient-based optimization.

Infinite neural networks. An infinite neural network does not have a definitive definition. We can distinguish two groups of concepts: infinite width and infinite depth.

Neal (1996) shows how a single-layer Bayesian neural network with infinite width and a judicious choice of weight prior is equivalent to a Gaussian process. The neural tangent kernel (Jacot et al., 2018; Novak et al., 2020) describes how such a layer evolves during training. De Matthews et al. (2018); Lee et al. (2018) extend these infinite-width results in multiple layers of deep networks. However, in recent work, Pleiss & Cunningham (2021) indicate that a large width in a deep model can be detrimental.

Implicit models take a different approach, to the depth. Deep equilibrium (Bai et al., 2019; 2020) is obtained by constraining an infinite depth neural network to repeat the same layer function until reaching a fixed point. Neural ODEs (Chen et al., 2018; Xu et al., 2021; Luo et al., 2022) define a model with continuous depth – hence infinite – by extending the residual connection of a ResNet into differential equations.

However, these approaches all constrain the weights of the infinite neural network. Some place specific priors on the weights for infinite-width while others set constraint equations on the weights for infinite-depth. The resulting functions can be difficult to relate to classical neural networks. In contrast, the UDN offers flexibility on how to construct the network, while remaining similar to a classic neural network. The practitioner can use arbitrary architectures in a UDN and inspect the posterior like an ensemble of finite networks.

6. Experiments

We study the performances of the UDN on synthetic and real classification tasks. We measure its predictive accuracy on held-out data and analyze how the posterior inference explores the space of truncations.

We find that:

- The dynamic variational inference effectively explores the space of truncations. The UDN adapts its depth to the complexity of the data, from a few layers to almost a hundred layers on image classification.
- The UDN posterior predictive accuracy outperforms all the finite network models Ω_L. It also outperforms ensembles of finite networks (Antoran et al., 2020) and implicit models (Dupont et al., 2019; Bai et al., 2020).

The experiments are implemented in Python with the library *PyTorch* (Paszke et al., 2019). To scale the inference to large datasets, we use stochastic optimization on the ELBO (Robbins & Monro, 1951; Hoffman et al., 2013). This requires unbiased gradients of the ELBO (7), which we compute by subsampling a batch of 256 observations (x_i, y_i) at each iteration. The code is available on GitHub[2].

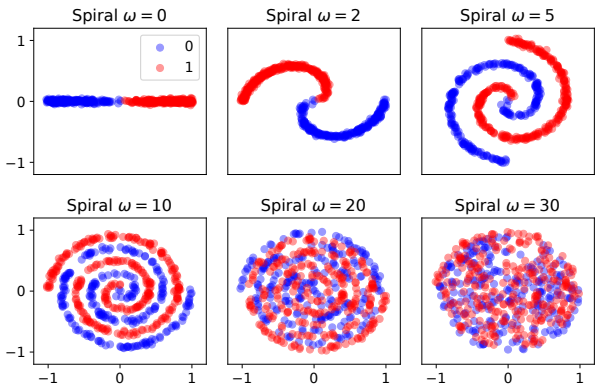


Figure 2. Spiral datasets $\mathcal{D}(\omega)$ for different rotation speed ω. As ω increases, the two branches of the spiral become harder to distinguish. When ω reaches 30, the data exhibits almost no pattern.

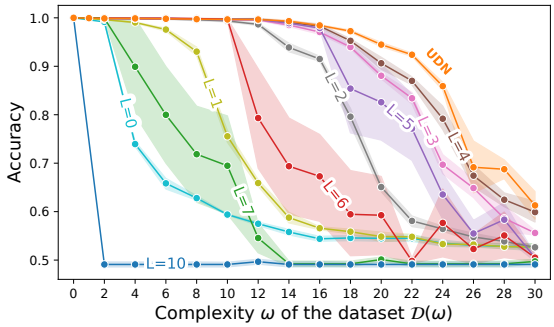
6.1. Spiral classification with fully connected networks

Dataset. To understand the role of depth, we highlight a natural question about neural network architecture: why do we need more depth when only a single wide enough hidden layer can approximate any smooth function (Hornik et al., 1989; Barron, 1994)? A half-answer would be that the same universal approximation theorem also holds with a deep enough neural network of fixed-width (Lu et al., 2017). A more complete answer lies in the quantification of the "wide enough" condition. Theoretical work on approximation theory has used small 2-hidden-layers neural networks to construct oscillating functions that cannot be approximated by any 1-hidden-layer network that uses a subexponential number of hidden units (Eldan & Shamir, 2016).

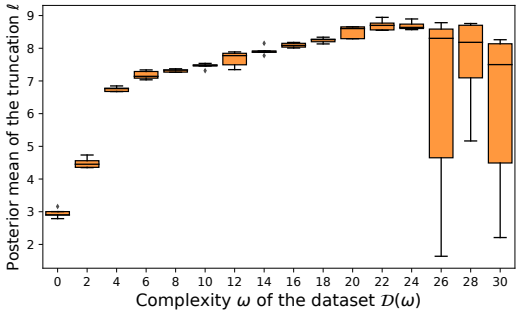
We adapt the construction of this paper and design labeled datasets $\mathcal{D}(\omega)$, each consisting of a spiral with two branches at rotation speed ω. The binary labels correspond to each branch. When ω increases, the branches of the spiral get more interleaved and become harder to distinguish. Figure 2 shows examples and intuition. At $\omega = 30$, the organization of labels appears to be almost random. The mathematical details for the generation of the datasets are in appendix.

Experiment. We follow the notations of section 2. The architecture generator f is defined to return layers with 32-hidden units each. So $f_1 : \mathbb{R}^2 \to \mathbb{R}^{32}$ and $f_\ell : \mathbb{R}^{32} \to \mathbb{R}^{32}$ for $\ell \geq 2$. Each f_l is a linear function followed by a ReLU activation. The output layers o_ℓ are linear layers

[2] https://github.com/ANazaret/unbounded-depth-neural-networks



(a) Classification accuracy for a range of models and datasets $\mathcal{D}(\omega)$. The UDN (orange) achieves the best accuracy for every dataset.



(b) Mean depth ℓ of the UDN posterior, fitted for different $\mathcal{D}(\omega)$. The UDN becomes deeper when the complexity increases.

Figure 3. (a) For all models, the accuracy decreases when ω increases. The accuracy for all models drops at large ω, which corresponds to when the datasets start to lose their patterns; see Figure 2. (b) The UDN favors larger depths as ω increases. It adapts to the complexity of the data. Near $\omega \sim 30$, independent runs of the inference yield different posterior expected depths. It happens when the accuracy drops in (a), and has the same explanation: when the dataset contains almost random label patterns, the UDN does not have an optimal depth to explain the data. – The error bands and the boxes with whiskers are computed with 5 independent runs for each dataset and model.

transforming a hidden state of $\mathbb{R}^{32}$ to a vector in $\mathbb{R}^2$, whose *softmax* parametrizes the probabilities of the two labels.

We train finite neural networks Ω_L for a large range of depths L, using the architecture returned by f. Each Ω_L and the UDN are trained independently. We also train Depth Uncertainty Networks (Antoran et al., 2020) with an ensemble of depths 1 to 10 – referred to as DUN-10.

For each ω, we generate a dataset $\mathcal{D}(\omega)$ on which we train the models for 4000 epochs. We then select the best epoch using a validation set and report the accuracy on a test set.

Results. Figure 3a reports the test accuracy of the UDN and the finite models for $\omega \in [0, 30]$. The UDN achieves the best accuracy for every dataset complexity.

Figure 3b offers reasons for the success of the UDN. The posterior UDN places mass on increasingly deeper ℓ when ω increases. Interestingly, at $\omega \geq 26$, the posterior of the UDN does not select a precise depth across independent runs. This behavior is coherent with the drop in accuracy in figure 3a since the dataset contains label patterns that are almost random at large ω, and the UDN does not have an optimal depth to explain the data.

Table 1 reports the average accuracy (across ω) for the best models and DUN-10. The UDN outperforms all of the other models. DUN-10 also outperforms the individual Ω_L, suggesting that combining several depths of the same network, like the posterior predictive of the UDN does, improves the representation capability of neural networks.

Model	Average accuracy (%)
Standard network Ω_5	84.8 ± 3.6
Standard network Ω_3	87.4 ± 0.6
Standard network Ω_4	89.3 ± 0.9
DUN-10 (Antoran et al., 2020)	90.5 ± 0.7
UDN	$\mathbf{91.7 \pm 1.1}$

Table 1. Test accuracy averaged over the different dataset $\mathcal{D}(\omega)$. The UDN outperforms all the other models. Standard deviations are calculated on 5 runs.

6.2. Image classification on CIFAR-10 with CNN

Dataset. Image classification is a domain where deeper networks have pushed the state-of-the-art. We study the performance of the UDN on the CIFAR-10 dataset (Krizhevsky et al., 2009). We use a layer architecture[3] adapted from He et al. (2016). ResNet building blocks are a succession of three convolution layers with a residual shortcut from the input to the output of the block. Furthermore, Batchnorm (Ioffe & Szegedy, 2015) and ReLU follow each convolution. The exact dimensions of the convolutions are in appendix.

Experiment. Using the notations of section 2, we define the architecture generator f to return layers f_ℓ that are ResNet blocks. Each block contains 3 convolutions, so each network Ω_L corresponds to a ResNet-$3L$. The output layers linearly map hidden states to vectors of $\mathbb{R}^{10}$, whose *softmax* parametrize the probabilities of the ten image classes.

[3]We are interested in the inference of the UDN and not in improving the state-of-the-art of image classification. Hence, we did not tune the best architecture possible.

Model	Accuracy
ResNet-15, Ω_5	$91.7_{\pm 0.2}$
ResNet-18 (Bai et al., 2020)	$92.9_{\pm 0.2}$
ResNet-24, Ω_8	$93.6_{\pm 0.4}$
ResNet-30, Ω_{10}	$94.0_{\pm 0.2}$
ResNet-45, Ω_{15}	$94.0_{\pm 0.2}$
ResNet-60, Ω_{20}	$93.9_{\pm 0.1}$
ResNet-90, Ω_{30}	$93.9_{\pm 0.1}$
NODE (Dupont et al., 2019)	$53.7_{\pm 0.2}$
ANODE (Dupont et al., 2019)	$60.6_{\pm 0.4}$
MDEQ (Bai et al., 2020)	$93.8_{\pm 0.3}$
UDN with ResNet	$\mathbf{94.4_{\pm 0.2}}$

Table 2. Test accuracy on CIFAR-10. The UDN outperforms all the other models. With the architecture that we use, the best finite models seem to have L between 10 and 20 (ResNet-30 and ResNet-60). – Standard deviations are calculated on 3 runs for our experiments, 5 runs for the reported ones.

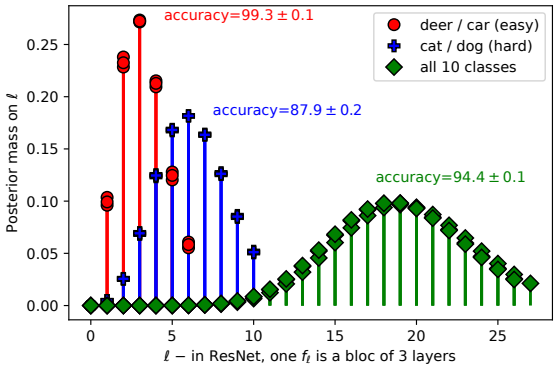


Figure 4. The UDN posterior on the depth ℓ adapts to the complexity of the dataset. It uses deeper truncations for a harder classification task like *cat vs dog* than for an easier one like *deer vs car*. When classifying the full CIFAR-10, the posterior UDN covers an even deeper set of truncations, which matches the depths of the best finite models from table 2. The maximal truncation considered by the green (diamond) posteriors corresponds to a ResNet-81. – For each dataset, 3 independent runs are represented.

We evaluate the UDN and various Ω_L. We report the performance of the ResNet-18 trained in Bai et al. (2020), and the performance of implicit methods: Neural ODEs (NODE), Augmented Neural ODEs (ANODE) and Multiscale Deep Equilibrium Models (MDEQ) (Chen et al., 2018; Dupont et al., 2019; Bai et al., 2020) which are competitive approaches for infinitely deep neural networks.

To evaluate the UDN and the Ω_L, we use the default train-test split of CIFAR-10, and report the test accuracy at the end of training. We use the same hyperparameters and architectures for the UDN and the individual Ω_L. Following He et al. (2016), we use the SGD optimizer and a specific learning rate schedule, detailed in appendix. Table 2 reports the results. The UDN outperforms all the other models.

Exploration of the posterior space. Finally, we aim to analyze how the posterior depth adapts to the complexity of the data. For this purpose, we create two subsamples of CIFAR-10: an *easy* subsample containing only *deer* and *car* images, and a *hard* subsample containing only *cat* and *dog* images. These categories were selected from an independently generated CIFAR-10 confusion matrix, in which (deer,car) were found to be the least confused labels, whereas (cat,dog) were the most. We suspect that the *hard* dataset will require more layers than the *easy* dataset, and will cause the posterior of the UDN to adapt accordingly.

We infer the posterior of the UDN on each of these two subsampled datasets, in addition to the full CIFAR-10. Figure 4 reports the probability mass functions of the posteriors. For the easiest pair of labels (deer, car), the UDN only uses a couple of layers to reach 99% accuracy, whereas it uses more layers for the harder pair (cat,dog) but reaches only 88%. For the full dataset, the posterior puts mass on much more layers and the UDN achieves 94% accuracy. The posteriors effectively adapt to the complexity of the datasets.

7. Discussion and future work

We proposed the unbounded depth neural network, a Bayesian deep neural network that uses as many layers as needed by the data, without any upper limit. We demonstrated empirically that the model adapts to different data complexities and competes with finite and infinite models.

To perform approximate inference of the UDN, we designed a novel variational inference algorithm capable of managing an infinite set of latent variables. We showed with experiments on real data that the algorithm successfully explores different regions of the posterior space.

The UDN and the dynamic variational inference offer several avenues for further research. First, the unbounded neural network could be applied to transformers, where very deep models have shown successful results (Liu et al., 2020). Another interesting direction is to use the unbounded variational family for variational inference of other infinite models, such as Dirichlet processes mixtures. Reciprocally, the UDN could be extended into a deep nonparametric model, where the truncation ℓ changes across observations in the same dataset. Finally, future studies of the UDN could examine how the form of the neural network's weight priors impacts the corresponding posteriors.

Acknowledgments

We are thankful to Gemma Moran, Elham Azizi, Clayton Sanford and anonymous reviewers for helpful comments and discussions. This work is funded by NSF IIS 2127869, ONR N00014-17-1-2131, ONR N00014-15-1-2209, the Simons Foundation, the Sloan Foundation, and Open Philanthropy.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Antoran, J., Allingham, J., and Hernández-Lobato, J. M. Depth uncertainty in neural networks. In *Neural Information Processing Systems*, 2020.

Author, N. N. Suppressed for anonymity, 2021.

Bai, S., Kolter, J. Z., and Koltun, V. Deep equilibrium models. In *Neural Information Processing Systems*, 2019.

Bai, S., Koltun, V., and Kolter, J. Z. Multiscale deep equilibrium models. In *Neural Information Processing Systems*, 2020.

Barron, A. R. Approximation and estimation bounds for artificial neural networks. *Machine Learning*, 14:115–133, 1994.

Bergstra, J., Yamins, D., and Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, 2013.

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Blei, D. and Jordan, M. I. Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, 1:121–143, 2006.

Blei, D., Kucukelbir, A., and McAuliffe, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112:859–877, 2017.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. In *Neural Information Processing Systems*, 2018.

Choi, K. P. On the medians of Gamma distributions and an equation of Ramanujan. *Proceedings of the American Mathematical Society*, 121:245–251, 1994.

De Matthews, A., Hron, J., Rowland, M., Turner, R., and Ghahramani, Z. Gaussian process behaviour in wide deep neural networks. In *International Conference on Learning Representations*, 2018.

Dikov, G. and Bayer, J. Bayesian learning of neural network architectures. In *Artificial Intelligence and Statistics*, 2019.

Doshi, F., Miller, K., Van Gael, J., and Teh, Y. W. Variational inference for the Indian buffet process. In *Artificial Intelligence and Statistics*, 2009.

Dua, D. and Graff, C. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Dupont, E., Doucet, A., and Teh, Y. W. Augmented neural ODEs. In *Neural Information Processing Systems*, 2019.

Eldan, R. and Shamir, O. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, 2016.

Ghosh, S., Yao, J., and Doshi-Velez, F. Model selection in Bayesian neural networks via horseshoe priors. *Journal of Machine Learning Research*, 20:1–46, 2019.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT press, 2016.

Hassibi, B. and Stork, D. G. Second order derivatives for network pruning: optimal brain surgeon. In *Neural Information Processing Systems*, 1992.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016.

Hjort, N. L., Holmes, C., Müller, P., and Walker, S. G. *Bayesian Nonparametrics*. Cambridge University Press, 2010.

Hoffman, M. D., Blei, D., Wang, C., and Paisley, J. Stochastic variational inference. *Journal of Machine Learning Research*, 14, 2013.

Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Hughes, M. C. and Sudderth, E. Memoized online variational inference for Dirichlet process mixture models. In *Neural Information Processing Systems*, 2013.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: convergence and generalization in neural networks. In *Neural Information Processing Systems*, 2018.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Kurihara, K., Welling, M., and Teh, Y. W. Collapsed variational Dirichlet process mixture models. In *International Joint Conference on Artificial Intelligence*, 2007a.

Kurihara, K., Welling, M., and Vlassis, N. Accelerated variational Dirichlet process mixtures. In *Neural Information Processing Systems*, 2007b.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Neural Information Processing Systems*, 2017.

LeCun, Y., Denker, J. S., and Solla, S. A. Optimal brain damage. In *Neural Information Processing Systems*, 1990.

Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. Deep neural networks as Gaussian processes. In *International Conference on Learning Representations*, 2018.

Liu, X., Duh, K., Liu, L., and Gao, J. Very deep transformers for neural machine translation. *arXiv:2008.07772*, 2020.

Lu, Z., Pu, H., Wang, F., Hu, Z., and Wang, L. The expressive power of neural networks: A view from the width. In *Neural Information Processing Systems*, 2017.

Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Neural Information Processing Systems*, 2018.

Luo, Z., Sun, Z., Zhou, W., Wu, Z., and Kamata, S.-i. Constructing infinite deep neural networks with flexible expressiveness while training. *Neurocomputing*, 487:257–268, 2022.

Mendoza, H., Klein, A., Feurer, M., Springenberg, J. T., and Hutter, F. Towards automatically-tuned neural networks. In *ICML Workshop on Automatic Machine Learning*, 2016.

Mitzenmacher, M. and Upfal, E. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, 2017.

Moran, G. E., Ročková, V., and George, E. I. Spike-and-slab lasso biclustering. *The Annals of Applied Statistics*, 15: 148–173, 2021.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. MIT press, 2012.

Neal, R. M. Priors for infinite networks. In *Bayesian Learning for Neural Networks*, pp. 29–53. Springer, 1996.

Novak, R., Xiao, L., Hron, J., Lee, J., Alemi, A. A., Sohl-Dickstein, J., and Schoenholz, S. S. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, 2018.

Pleiss, G. and Cunningham, J. P. The limitations of large width in neural networks: A deep Gaussian process perspective. In *Neural Information Processing Systems*, 2021.

Ranganath, R. and Blei, D. Correlated random measures. *Journal of the American Statistical Association*, 113:417–430, 2018.

Robbins, H. and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics*, pp. 400–407, 1951.

Short, M. Improved inequalities for the Poisson and Binomial distribution and upper tail quantile functions. *International Scholarly Research Notices*, 2013, 2013.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Neural Information Processing Systems*, 2017.

Wainwright, M. J. and Jordan, M. I. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc, 2008.

Xu, W., Chen, R. T., Li, X., and Duvenaud, D. Infinitely deep Bayesian neural networks with stochastic differential equations. *arXiv:2102.06559*, 2021.

Zhai, K. and Boyd-Graber, J. Online latent Dirichlet allocation with infinite vocabulary. In *International Conference on Machine Learning*, 2013.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A. The truncated Poisson family

We prove Theorem 3.2: For any $\delta \in [0.5, 1[$, $\mathcal{TP}(\delta)$ is unbounded with connected bounded members. For $\delta = 0.95$, we have

1. $\lambda - \ln 2 \leq m(q^{0.95}(\lambda)) \leq 1.3\lambda + 5$ (8)
2. $\forall n \in \mathbb{N}^*,\ \ n \in \text{argmax}(q^{0.95}(n + 0.5))$ (9)
3. $\lambda > 0$ is a continuous parameter. (10)

Proof. We first show that $\mathcal{TP}(\delta)$ is unbounded with connected bounded members for any $\delta \in [0.5, 1[$.

- Each distribution is the truncation of a Poisson distribution so it has a finite support by definition. Hence, $\mathcal{TP}(\delta)$ has bounded members and satisfies (1).
- The assertion (9) is true for any δ, we show it below. It ensures that $\mathcal{TP}(\delta)$ is unbounded and satisfies (2).
- λ is a continuous positive real number by design of the family. Hence the members of $\mathcal{TP}(\delta)$ are continuously connected and the family satisfies (3).

So $\mathcal{TP}(\delta)$ is unbounded with connected bounded members for any $\delta \in [0.5, 1[$.

We prove the additional assertions. By definition of the quantile truncation, we will use that $m(q^\delta(\lambda)) \geq m(q^\eta(\lambda))$ if $\delta \geq \eta$, and that $m(q^\delta) \leq \delta$-quantile of q.

1. **Lower bound**: According to (Choi, 1994), the median of Poisson(λ) is greater or equal than $\lambda - \ln 2$. Hence, by definition of the quantile truncation, we have $m(q^\delta(\lambda)) \geq \text{median}(\text{Poisson}(\lambda)) \geq \lambda - \ln 2$ for any $\delta \geq 0.5$.

2. The modes of a Poisson(λ) are $\lceil \lambda \rceil - 1$ and $\lfloor \lambda \rfloor$ (which are the same value for non-integer λ). For a truncation at level δ with $\delta \geq 0.5$, the lowest term truncated can at most reach the median. Since $\text{median}(\text{Poisson}(\lambda)) \geq \lambda - \ln 2$, the lowest truncated term of Poisson($n + 0.5$) is at most $n + 0.5 - \log 2 > n$. But we also have $n = \lfloor n + 0.5 \rfloor$. So the mode n does not get truncated. Since the mode is not affected by re-normalization of the probabilities, n is still the mode of Poisson($n + 0.5$).

1. **Upper bound**: Let $X \sim \text{Poisson}(\lambda)$ for some $\lambda > 0$.
   - We recall a standard Chernoff bounds argument from (Mitzenmacher & Upfal, 2017)[4]:
     $\forall x > \lambda, \mathbb{P}(X \geq x) \leq \frac{(e\lambda)^x e^{-\lambda}}{x^x}$.
   - For $x = 1.3\lambda$, it yields: $\mathbb{P}(X \geq 1.3\lambda) \leq e^{0.3\lambda} \left(\frac{1}{1.3}\right)^{1.3\lambda} = \left(e^{0.3} \frac{1}{1.3^{1.3}}\right)^\lambda < 0.96^\lambda$.
     For $\lambda > 70$, we have $\mathbb{P}(X \geq 1.3\lambda) \leq 0.96^{70} < 0.05$, so $m(q^{0.95}(\lambda)) \leq 1.3\lambda \leq 1.3\lambda + 5$.
   - Note that for $\mu \geq \lambda$, we have $m(q(\mu)) \geq m(q(\lambda))$. In particular, we have: $m(q(\lambda)) \leq m(q(\lceil \lambda \rceil))$. Also we have $1.3\lfloor \lambda \rfloor + 5 \leq 1.3\lambda + 5$. So it suffices to show that $\forall \lambda \in]0, 70], m(q(\lceil \lambda \rceil)) \leq 1.3\lfloor \lambda \rfloor + 5$ and we will have that $\forall \lambda \in]0, 70], m(q(\lambda)) \leq 1.3\lambda + 5$.
     It suffices to check manually that, $\forall k \in \{1, 2, ..., 70\}, m(q(k)) \leq 1.3(k-1) + 5$. We check this in Python and report the figure 5 where each $(1.3(k-1) + 5) - m(q(k))$ is positive for $k \in \{1, 2, ..., 70\}$.
   - This concludes the proof.

- This concludes the proof for the three assertions about $\delta = 0.95$.

□

[4] And also Wikipedia https://en.wikipedia.org/wiki/Poisson_distribution#Other_properties

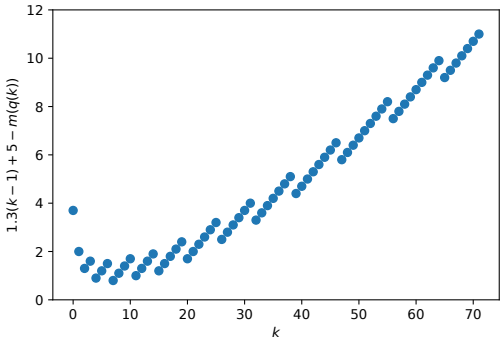


Figure 5. Empirical evaluation of $\delta_k = (1.3(k-1)+5) - m(q(k))$ for $k \in \{1, 2, ..., 70\}$. We observe that $\delta_k \geq 0$.

B. Experiments

B.1. Synthetic experiment

Dataset. The spiral dataset $D(\omega) = \{(x_i, y_i)\}$ is generated by the following generative process:

$$t \sim \text{Uniform}([0, 1]) \tag{12}$$

$$u = \sqrt{t} \tag{13}$$

$$y \sim \text{Uniform}(\{-1, 1\}) \tag{14}$$

$$x \sim \mathcal{N}\left(\begin{bmatrix} yu \cdot \cos\left(\omega u \cdot \frac{\pi}{2}\right) \\ yu \cdot \sin\left(\omega u \cdot \frac{\pi}{2}\right) \end{bmatrix}, 0.02\right) \tag{15}$$

The square root of t is taken to rebalance the density along the curve $u \mapsto \begin{bmatrix} u \cdot \cos\left(\omega|u| \cdot \frac{\pi}{2}\right) \\ u \cdot \sin\left(\omega|u| \cdot \frac{\pi}{2}\right) \end{bmatrix}$.

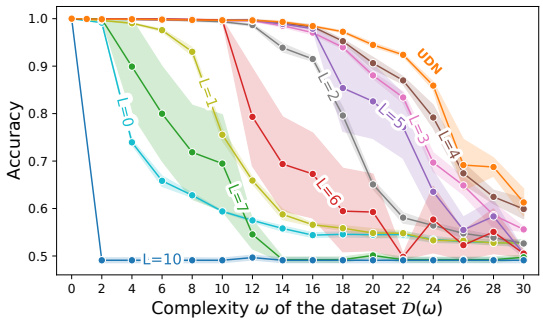
Training. For each ω, we independently sample a train, a validation and a test dataset of each 1024 samples. We use the following hyperparameters for training:

- Prior on the neural network weights: $\theta \sim \mathcal{N}(0, 1)$
- Prior on the truncation ℓ: $\ell - 1 \sim \text{Poisson}(0.5)$. We recall that $\ell \geq 1$ so we shift the Poisson by 1.
- Optimizer: Adam (Kingma & Ba, 2015)
- Learning rate: 0.005. Results with different learning rates are compared in Figure 6.
- Learning rate for λ: we use a learning rate that is $1/10$th of the general learning rate of the neural network weights.
- Initialization of the variational truncated Poisson family: $\lambda = 1.0$
- Number of epochs: 4000

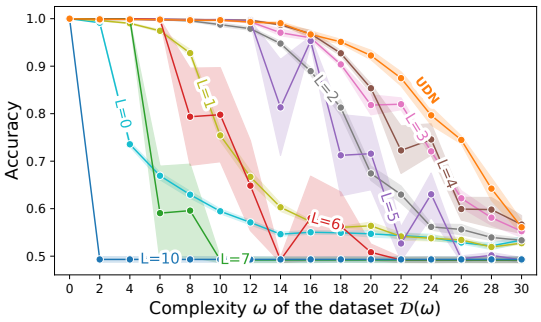
B.2. CIFAR-10 experiment

Architecture generator f We detail the construction of the architecture generator f for the CIFAR experiment of section 6. A ResNet building block B_k is the succession of three convolution layers $[1 \times 1 \times 2^k]$, $[3 \times 3 \times 2^k]$, and $[1 \times 1 \times 2^{k+2}]$ with a residual shortcut from the input to the output of the block. In these blocks, Batchnorm and ReLU follow every convolution. The architecture generator used by the UDN and the finite neural networks is the following:

$$f : \ell \in \mathbb{N}^* \mapsto f_l := \begin{cases} B_6 & \text{if } \ell \in [\![1, 3]\!] \\ B_7 & \text{if } \ell \in [\![4, 8]\!] \\ B_8 & \text{else} \end{cases}$$



(a) Learning rate 0.005



(b) Learning rate 0.01

Figure 6. Classification accuracy for a range of models and datasets $\mathcal{D}(\omega)$. The UDN (orange) achieves the best accuracy for almost each complexity ω. The deep finite models ($L \geq 5$) are sensitive to a higher learning rate, meanwhile the UDN is robust even with its large (infinite) depth. – The error bands are computed with 5 independent runs for each pair of dataset and model.

Output generator o The output generator o_ℓ at each depth ℓ receives the hidden state h_ℓ, performs a 2D average pooling with kernel $(4, 4)$ and applies a linear layer of the adequate dimension to $\mathbb{R}^{10}$.

B.3. Hyperparameters

Training. For the training on CIFAR-10, we follow the optimizer and the learning rate schedule from He et al. (2016). We increase our prior on the truncation since we suspect that CIFAR requires deep architectures. Similarly, we initialize the variational truncated Poisson family to $\lambda_0 = 5$. We show the trajectories of the variational families during the optimization for different values of λ_0 in figure 7.

- Prior on the neural network weights: $\theta \sim \mathcal{N}(0, 1)$
- Prior on the truncation ℓ: $\ell - 1 \sim \text{Poisson}(1)$.
- Optimizer: SGD with `momentum = 0.9, weight_decay=1e-4`
- Number of epochs: 500
- Learning rate schedule: `[0.01]*5 + [0.1]*195 + [0.01]*100 + [0.001]*100`
- Learning rate for λ: we used the same learning rate for λ and the weights
- Initialization of the variational truncated Poisson family: $\lambda_0 = 5.0$.

B.4. Additional experiments on Regression datasets

We run additional experiments on tabular datasets. We perform regression with the UDN for nine regression datasets from the UCI repository (Dua & Graff, 2017): *Boston Housing* (boston) *Concrete Strength* (concrete), *Energy Efficiency* (energy), *Kin8nm* (kin8nm), *Naval Propulsion* (naval), *Power Plant* (power), *Protein Structure* (protein), *Wine Quality* (wine) and *Yacht Hydrodynamics* (yacht). For regression (instead of classification in the previous experiments), the UDN models the response variable with a Gaussian distribution whose mean is given by the output layers of the infinite neural network. In figure 8, we show the performance of the UDN against the finite variants (f2, f3, f4, f5, f6 with respectively exactly 2, 3, 4, 5, and 6 layers). Figure 8 also indicates the posterior mean of the truncation level ℓ learned during inference. Overall, we notice that for regression tasks on tabular data, very deep networks are not necessary to achieve good prediction performance. For the datasets yacht, boston, energy, concrete and power, the posterior ℓ is lower than two and the UDN offers similar performances than the finite alternatives. For the other datasets such as wine, naval, kin8nm, and protein, the UDN learns a higher number of layers and performs competitively or better than the finite alternatives.

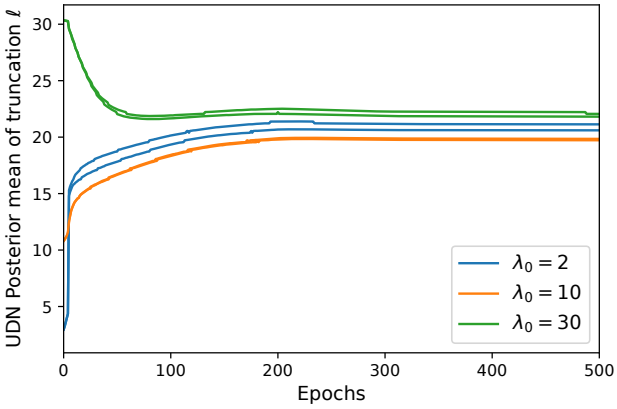


Figure 7. Evolution of the posterior mean of the truncation over multiple training with different initializations of the variational family. Trained on CIFAR-10. In all cases, the variational families explore the posterior space and converge to similar regions.

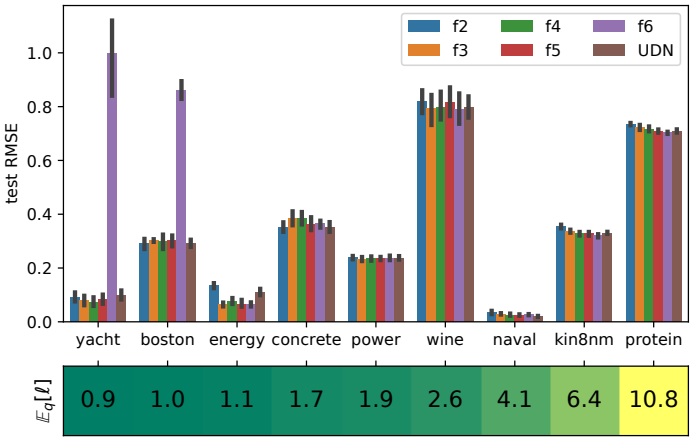


Figure 8. (top) Regression root mean square error (RMSE) on the test set of 9 UCI datasets. The optimal training epoch is chosen with a validation set. Error bars are obtained by repeating the experiment on 10 different train-valid-test sets. (bottom) Expectation of the number of layers ℓ in the UDN posterior. For the first five datasets, the expectation is lower than two. For the last four datasets, the number of layers gets higher and the UDN shows competitive performances against the finite alternatives.