Avoiding pathologies in very deep networks

David Duvenaud University of Cambridge
Oren Rippel Harvard, M.I.T.
Ryan P. Adams Harvard University
Zoubin Ghahramani University of Cambridge

Abstract

Choosing appropriate architectures and regularization strategies of deep networks is crucial to good predictive performance. To shed light on this problem, we analyze the analogous problem of constructing useful priors on compositions of functions. Specifically, we study the deep Gaussian process, a type of infinitely-wide, deep neural network. We show that in standard architectures, the representational capacity of the network tends to capture fewer degrees of freedom as the number of layers increases, retaining only a single degree of freedom in the limit. We propose an alternate network architecture which does not suffer from this pathology. We also examine deep covariance functions, obtained by composing infinitely many feature transforms. Lastly, we characterize the class of models obtained by performing dropout on Gaussian processes.

In this paper, we propose to study the problem of choosing neural net architectures by viewing deep neural networks as priors on functions. By viewing neural networks this way, one can analyze their properties without reference to any particular dataset, loss function, or training method.

As a starting point, we will relate neural networks to Gaussian processes, and examine a class of infinitely-wide, deep neural networks called *deep Gaussian processes*: compositions of functions drawn from GP priors. Deep GPs are an attractive model class to study for several reasons. First, **?** showed that the probabilistic nature of deep GPs guards against overfitting. Second, **?** showed that stochastic variational inference is possible in deep GPs, allowing mini-batch training on large datasets. Third, the availability of an approximation to the marginal likelihood allows one to automatically tune the model architecture without the need for cross-validation. Finally, deep GPs are attractive from a model-analysis point of view, because they abstract away some of the details of finite neural networks.

Our analysis will show that in standard architectures, the representational capacity of standard deep networks tends to decrease as the number of layers increases, retaining only a single degree of freedom in the limit. We propose an alternate network architecture that connects the input to each layer that does not suffer from this pathology. We also examine *deep kernels*, obtained by composing arbitrarily many fixed feature transforms. Finally, we characterise the prior obtained by performing dropout regularization on GPs, showing equivalences to existing models.

1 Relating deep neural networks to deep GPs

This section gives a precise definition of deep GPs, reviews the precise relationship between neural networks and Gaussian processes, and gives two equivalent ways of constructing neural networks which give rise to deep GPs.

1.1 Definition of deep GPs

We define a deep GP as a distribution on functions constructed by composing functions drawn from GP priors. An example of a deep GP is a composition of vector-valued functions, with each function drawn

independently from GP priors:

$$\boldsymbol{f}^{(1:L)}(\mathbf{x}) = \boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\dots \boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\mathbf{x}))\dots)) \quad (1)$$
$$\text{with each} \quad f_d^{(\ell)} \overset{\text{ind}}{\sim} \text{GP}\left(0, k_d^{(\ell)}(\mathbf{x}, \mathbf{x}')\right)$$

Multilayer neural networks also implement compositions of vector-valued functions, one per layer. Therefore, understanding general properties of function compositions might helps us to gain insight into deep neural networks.

1.2 Single-hidden-layer models

First, we relate neural networks to standard "shallow" Gaussian processes, using the standard neural network architecture known as the multi-layer perceptron (MLP) [**?**]. In the typical definition of an MLP with one hidden layer, the hidden unit activations are defined as:

$$\mathbf{h}(\mathbf{x}) = \sigma\left(\mathbf{b} + \mathbf{V}\mathbf{x}\right) \quad (2)$$

where $\mathbf{h}$ are the hidden unit activations, $\mathbf{b}$ is a bias vector, $\mathbf{V}$ is a weight matrix and σ is a one-dimensional nonlinear function, usually a sigmoid, applied element-wise. The output vector $\boldsymbol{f}(\mathbf{x})$ is simply a weighted sum of these hidden unit activations:

$$\boldsymbol{f}(\mathbf{x}) = \mathbf{W}\sigma\left(\mathbf{b} + \mathbf{V}\mathbf{x}\right) = \mathbf{W}\mathbf{h}(\mathbf{x}) \quad (3)$$

where $\mathbf{W}$ is another weight matrix.

?, chapter 2 showed that some neural networks with infinitely many hidden units, one hidden layer, and unknown weights correspond to Gaussian processes. More precisely, for any model of the form

$$f(\mathbf{x}) = \frac{1}{K}\mathbf{w}^\mathsf{T}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} w_i h_i(\mathbf{x}), \quad (4)$$

with fixedfeatures $[h_1(\mathbf{x}), \dots, h_K(\mathbf{x})]^\mathsf{T} = \mathbf{h}(\mathbf{x})$ and i.i.d. w's with zero mean and finite variance σ^2, the central limit theorem implies that as the number of features K grows, any two function values $f(\mathbf{x})$ and $f(\mathbf{x}')$ have a joint distribution approaching a Gaussian:

$$\lim_{K\to\infty} p\left(\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \frac{\sigma^2}{K}\begin{bmatrix} \sum_{i=1}^K h_i(\mathbf{x})h_i(\mathbf{x}) & \sum_{i=1}^K h_i(\mathbf{x})h_i(\mathbf{x}') \\ \sum_{i=1}^K h_i(\mathbf{x}')h_i(\mathbf{x}) & \sum_{i=1}^K h_i(\mathbf{x}')h_i(\mathbf{x}') \end{bmatrix}\right) \quad (5)$$

A joint Gaussian distribution between any set of function values is the definition of a Gaussian process.

The result is surprisingly general: it puts no constraints on the features (other than having uniformly bounded activation), nor does it require that the feature weights $\mathbf{w}$ be Gaussian distributed. An MLP with a finite number of nodes also gives rise to a GP, but only if the distribution on $\mathbf{w}$ is Gaussian.

One can also work backwards to derive a one-layer MLP from any GP: Mercer's theorem, discussed in **??**, implies that any positive-definite kernel function corresponds to an inner product of features: $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')$.

Thus, in the one-hidden-layer case, the correspondence between MLPs and GPs is straightforward: the implicit features $\mathbf{h}(\mathbf{x})$ of the kernel correspond to hidden units of an MLP.

1.3 Multiple hidden layers

Next, we examine infinitely-wide MLPs having multiple hidden layers. There are several ways to construct such networks, giving rise to different priors on functions.

In an MLP with multiple hidden layers, activation of the ℓth layer units are given by

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(\ell)} + \mathbf{V}^{(\ell)}\mathbf{h}^{(\ell-1)}(\mathbf{x})\right) . \quad (6)$$

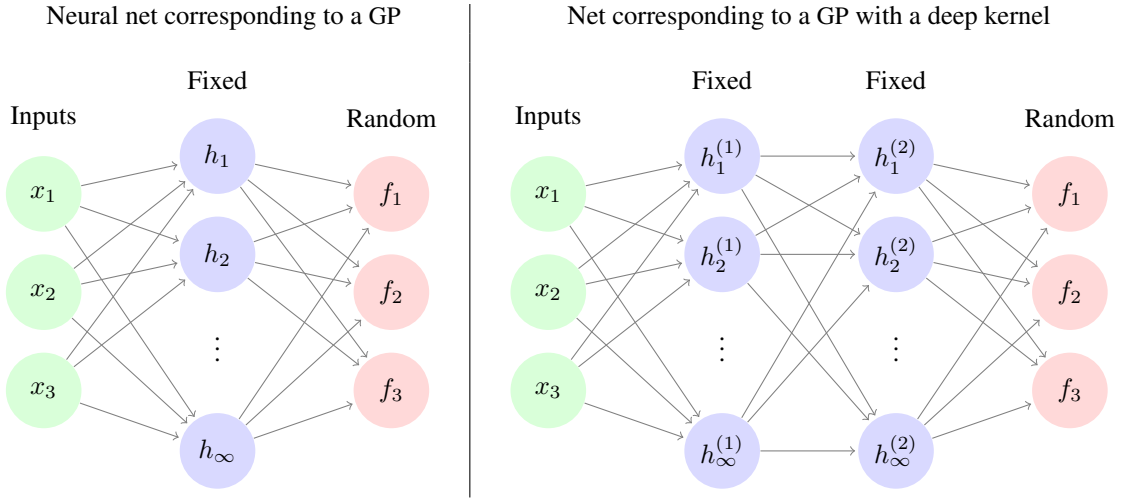


Figure 1: *Left:* GPs can be derived as a one-hidden-layer MLP with infinitely many fixed hidden units having unknown weights. *Right:* Multiple layers of fixed hidden units gives rise to a GP with a deep kernel, but not a deep GP.

This architecture is shown on the right of figure 1. For example, if we extend the model given by equation (3) to have two layers of feature mappings, the resulting model becomes

$$f(\mathbf{x}) = \frac{1}{K}\mathbf{w}^\mathsf{T}\mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right) \,. \tag{7}$$

If the features $\mathbf{h}^{(1)}(\mathbf{x})$ and $\mathbf{h}^{(2)}(\mathbf{x})$ are fixed with only the last-layer weights $\mathbf{w}$ unknown, this model corresponds to a shallow GP with a *deep kernel*, given by

$$k(\mathbf{x}, \mathbf{x}') = \left[\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}))\right]^\mathsf{T} \mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}')) \,. \tag{8}$$

Deep kernels, explored in section 5, imply a fixed representation as opposed to a prior over representations. Thus, unless we richly parameterize these kernels, their capacity to learn an appropriate representation will be limited in comparison to more flexible models such as deep neural networks or deep GPs.

1.4 Two network architectures equivalent to deep GPs

There are two equivalent neural network architectures that correspond to deep GPs: one having fixed nonlinearities, and another having GP-distributed nonlinearities.

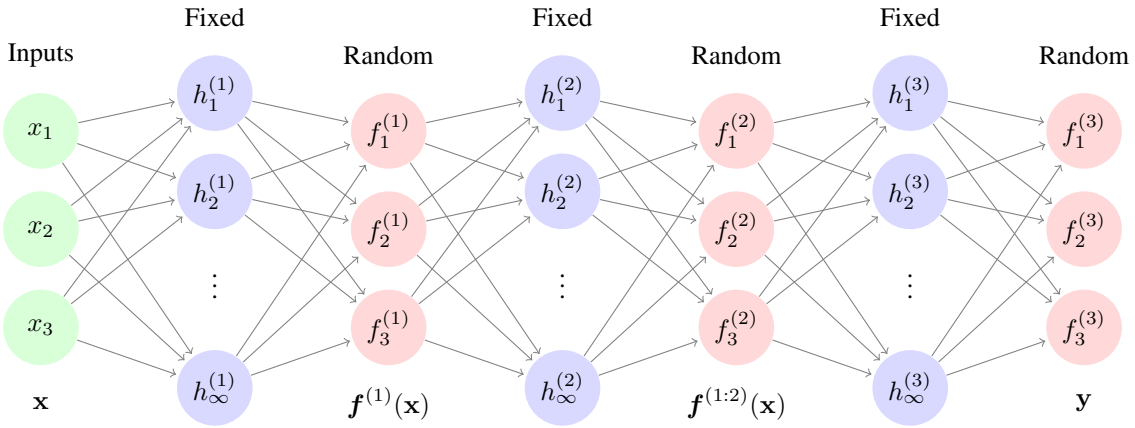
To construct a neural network corresponding to a deep GP using only fixed nonlinearities, one can start with the infinitely-wide deep GP shown in figure 1(right), and introduce a finite set of nodes in between each infinitely-wide set of fixed basis functions. This architecture is shown in the top of figure 2. The $D^{(\ell)}$ nodes $\boldsymbol{f}^{(\ell)}(\mathbf{x})$ in between each fixed layer are weighted sums (with random weights) of the fixed hidden units of the layer below, and the next layer's hidden units depend only on these $D^{(\ell)}$ nodes.

This alternating-layer architecture has an interpretation as a series of linear information bottlenecks. To see this, substitute equation (3) into equation (6) to get

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(\ell)} + \left[\mathbf{V}^{(\ell)}\mathbf{W}^{(\ell-1)}\right]\mathbf{h}^{(\ell-1)}(\mathbf{x})\right) \tag{9}$$

where $\mathbf{W}^{(\ell-1)}$ is the weight matrix connecting $\mathbf{h}^{(\ell-1)}$ to $\boldsymbol{f}^{(\ell-1)}$. Thus, ignoring the intermediate outputs $\boldsymbol{f}^{(\ell)}(\mathbf{x})$, a deep GP is an infinitely-wide, deep MLP with each pair of layers connected by random, rank-D_ℓ matrices given by $\mathbf{V}^{(\ell)}\mathbf{W}^{(\ell-1)}$.

A neural net with fixed activation functions corresponding to a 3-layer deep GP



A net with nonparametric activation functions corresponding to a 3-layer deep GP

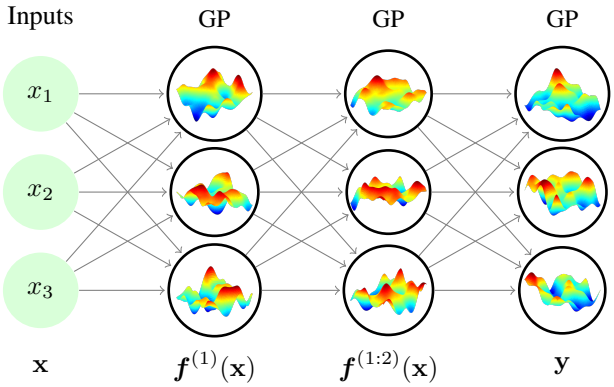


Figure 2: Two equivalent views of deep GPs as neural networks. *Top:* A neural network whose every other layer is a weighted sum of an infinite number of fixed hidden units, whose weights are initially unknown. *Bottom:* A neural network with a finite number of hidden units, each with a different unknown non-parametric activation function. The activation functions are visualized by draws from 2-dimensional GPs, although their input dimension will actually be the same as the output dimension of the previous layer.

The second, more direct way to construct a network architecture corresponding to a deep GP is to integrate out all $\mathbf{W}^{(\ell)}$, and view deep GPs as a neural network with a finite number of nonparametric, GP-distributed basis functions at each layer, in which $\boldsymbol{f}^{(1:\ell)}(\mathbf{x})$ represent the output of the hidden nodes at the ℓ^{th} layer. This second view lets us compare deep GP models to standard neural net architectures more directly. Figure 1(bottom) shows an example of this architecture.

2 Characterizing deep Gaussian process priors

This section develops several theoretical results characterizing the behavior of deep GPs as a function of their depth. Specifically, we show that the size of the derivative of a one-dimensional deep GP becomes log-normal distributed as the network becomes deeper. We also show that the Jacobian of a multivariate deep GP is a product of independent Gaussian matrices having independent entries. These results will allow us to identify a pathology that emerges in very deep networks in section 3.

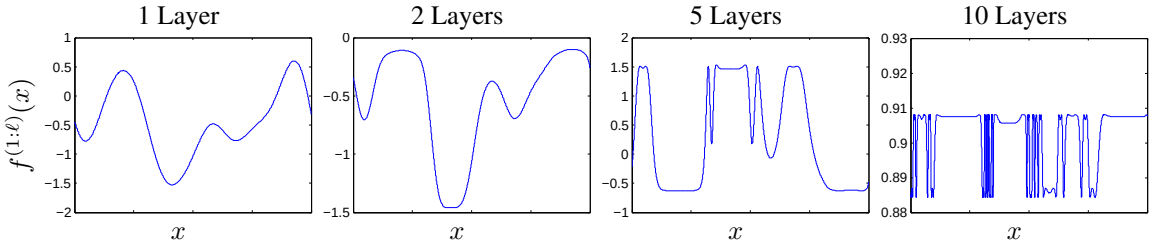


Figure 3: A function drawn from a one-dimensional deep GP prior, shown at different depths. The x-axis is the same for all plots. After a few layers, the functions begin to be either nearly flat, or quickly-varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed. As well, the function values at each layer tend to cluster around the same few values as the depth increases. This happens because once the function values in different regions are mapped to the same value in an intermediate layer, there is no way for them to be mapped to different values in later layers.

2.1 One-dimensional asymptotics

In this section, we derive the limiting distribution of the derivative of an arbitrarily deep, one-dimensional GP having a squared-exp kernel:

$$\mathrm{SE}(x, x') = \sigma^2 \exp\left(\frac{-(x-x')^2}{2w^2}\right) . \tag{10}$$

The parameter σ^2 controls the variance of functions drawn from the prior, and the lengthscale parameter w controls the smoothness. The derivative of a GP with a squared-exp kernel is point-wise distributed as $\mathcal{N}\left(0, {}^{\sigma^2}/_{w^2}\right)$. Intuitively, a draw from a GP is likely to have large derivatives if the kernel has high variance and small lengthscales.

By the chain rule, the derivative of a one-dimensional deep GP is simply a product of the derivatives of each layer, which are drawn independently by construction. The distribution of the absolute value of this derivative is a product of half-normals, each with mean $\sqrt{{}^{2\sigma^2}/_{\pi w^2}}$. If one chooses kernel parameters such that ${}^{\sigma^2}/_{w^2} = {}^{\pi}/_{2}$, then the expected magnitude of the derivative remains constant regardless of the depth.

The distribution of the log of the magnitude of the derivatives has finite moments:

$$\begin{aligned} m_{\log} &= \mathbb{E}\left[\log\left|\frac{\partial f(x)}{\partial x}\right|\right] = 2\log\left(\frac{\sigma}{w}\right) - \log 2 - \gamma \\ v_{\log} &= \mathbb{V}\left[\log\left|\frac{\partial f(x)}{\partial x}\right|\right] = \frac{\pi^2}{4} + \frac{\log^2 2}{2} - \gamma^2 - \gamma\log 4 + 2\log\left(\frac{\sigma}{w}\right)\left[\gamma + \log 2 - \log\left(\frac{\sigma}{w}\right)\right] \end{aligned} \tag{11}$$

where $\gamma \approxeq 0.5772$ is Euler's constant. Since the second moment is finite, by the central limit theorem, the limiting distribution of the size of the gradient approaches a log-normal as L grows:

$$\log\left|\frac{\partial f^{(1:L)}(x)}{\partial x}\right| = \log\prod_{\ell=1}^{L}\left|\frac{\partial f^{(\ell)}(x)}{\partial x}\right| = \sum_{\ell=1}^{L}\log\left|\frac{\partial f^{(\ell)}(x)}{\partial x}\right| \overset{L\to\infty}{\sim} \mathcal{N}\left(Lm_{\log}, L^2 v_{\log}\right) \tag{12}$$

Even if the expected magnitude of the derivative remains constant, the variance of the log-normal distribution grows without bound as the depth increases.

Because the log-normal distribution is heavy-tailed and its domain is bounded below by zero, the derivative will become very small almost everywhere, with rare but very large jumps. Figure 3 shows this behavior in a draw from a 1D deep GP prior. This figure also shows that once the derivative in one region of the input space becomes very large or very small, it is likely to remain that way in subsequent layers.

2.2 Distribution of the Jacobian

Next, we characterize the distribution on Jacobians of multivariate functions drawn from deep GP priors, finding them to be products of independent Gaussian matrices with independent entries.

Lemma 2.1. *The partial derivatives of a function mapping $\mathbb{R}^D \to \mathbb{R}$ drawn from a GP prior with a product kernel are independently Gaussian distributed.*

Proof. Because differentiation is a linear operator, the derivatives of a function drawn from a GP prior are also jointly Gaussian distributed. The covariance between partial derivatives with respect to input dimensions d_1 and d_2 of vector $\mathbf{x}$ are given by **?**:

$$\operatorname{cov}\left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}}\right) = \left.\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_{d_1} \partial x'_{d_2}}\right|_{\mathbf{x}=\mathbf{x}'} \tag{13}$$

If our kernel is a product over individual dimensions $k(\mathbf{x}, \mathbf{x}') = \prod_d^D k_d(x_d, x'_d)$, then the off-diagonal entries are zero, implying that all elements are independent. □

For example, in the case of the multivariate squared-exp kernel, the covariance between derivatives has the form:

$$\begin{aligned} f(\mathbf{x}) &\sim \text{GP}\left(0, \sigma^2 \prod_{d=1}^{D} \exp\left(-\frac{1}{2}\frac{(x_d - x'_d)^2}{w_d^2}\right)\right) \\ \implies \operatorname{cov}\left(\frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}}\right) &= \begin{cases} \frac{\sigma^2}{w_{d_1}^2} & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \end{cases} \end{aligned} \tag{14}$$

Lemma 2.2. *The Jacobian of a set of D functions $\mathbb{R}^D \to \mathbb{R}$ drawn from independent GP priors, each having product kernel is a $D \times D$ matrix of independent Gaussian R.V.'s*

Proof. The Jacobian of the vector-valued function $\boldsymbol{f}(\mathbf{x})$ is a matrix J with elements $J_{ij}(\mathbf{x}) = \frac{\partial f_i(\mathbf{x})}{\partial x_j}$. Because the GPs on each output dimension $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_D(\mathbf{x})$ are independent by construction, it follows that each row of J is independent. Lemma 2.1 shows that the elements of each row are independent Gaussian. Thus all entries in the Jacobian of a GP-distributed transform are independent Gaussian R.V.'s. □

Theorem 2.3. *The Jacobian of a deep GP with a product kernel is a product of independent Gaussian matrices, with each entry in each matrix being drawn independently.*

Proof. When composing L different functions, we denote the *immediate* Jacobian of the function mapping from layer $\ell - 1$ to layer ℓ as $J^{(\ell)}(\mathbf{x})$, and the Jacobian of the entire composition of L functions by $J^{(1:L)}(\mathbf{x})$. By the multivariate chain rule, the Jacobian of a composition of functions is given by the product of the immediate Jacobian matrices of each function. Thus the Jacobian of the composed (deep) function $\boldsymbol{f}^{(L)}(\boldsymbol{f}^{(L-1)}(\ldots \boldsymbol{f}^{(3)}(\boldsymbol{f}^{(2)}(\boldsymbol{f}^{(1)}(\mathbf{x})))\ldots))$ is

$$J^{(1:L)}(\mathbf{x}) = J^{(L)} J^{(L-1)} \ldots J^{(3)} J^{(2)} J^{(1)}. \tag{15}$$

By lemma 2.2, each $J_{i,j}^{(\ell)} \overset{\text{ind}}{\sim} \mathcal{N}$, so the complete Jacobian is a product of independent Gaussian matrices, with each entry of each matrix drawn independently. □

This result allows us to analyze the representational properties of a deep Gaussian process by examining the properties of products of independent Gaussian matrices.

3 Formalizing a pathology

A common use of deep neural networks is building useful representations of data manifolds. What properties make a representation useful? **?** argued that good representations of data manifolds are invariant in directions orthogonal to the data manifold. They also argued that, conversely, a good representation must also change in directions tangent to the data manifold, in order to preserve relevant information. Figure 4 visualizes a representation having these two properties.

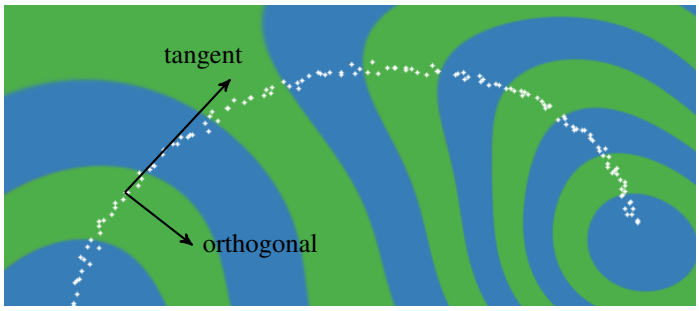


Figure 4: Representing a 1-D data manifold. Colors are a function of the computed representation of the input space. The representation (blue & green) changes little in directions orthogonal to the manifold (white), making it robust to noise in those directions. The representation also varies in directions tangent to the data manifold, preserving information for later layers.

As in **?**, we characterize the representational properties of a function by the singular value spectrum of the Jacobian. The number of relatively large singular values of the Jacobian indicate the number of directions in data-space in which the representation varies significantly. Figure 5 shows the distribution of the singular

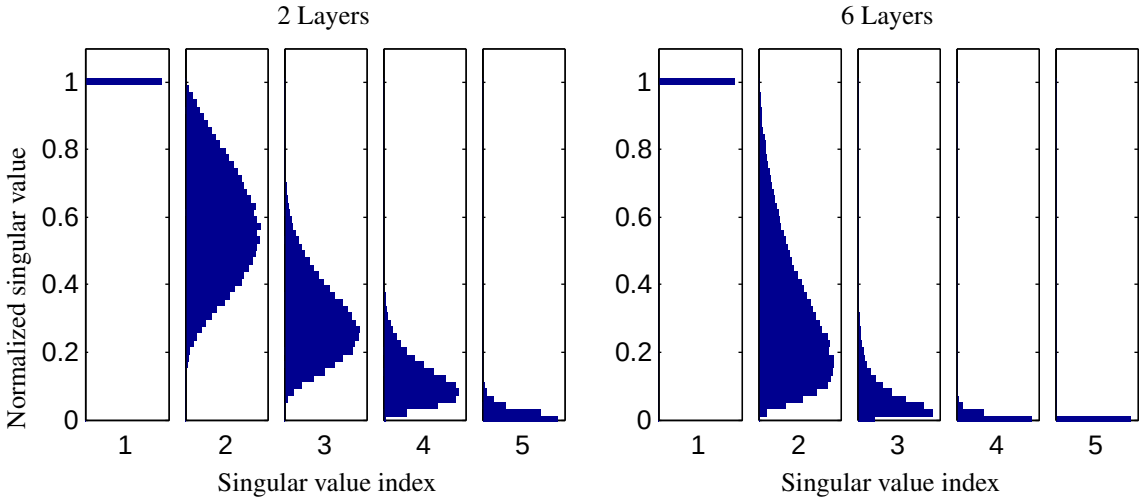


Figure 5: The distribution of normalized singular values of the Jacobian of a function drawn from a 5-dimensional deep GP prior 25 layers deep (*Left*) and 50 layers deep (*Right*). As nets get deeper, the largest singular value tends to become much larger than the others. This implies that with high probability, these functions vary little in all directions but one, making them unsuitable for computing representations of manifolds of more than one dimension.

value spectrum of draws from 5-dimensional deep GPs of different depths.[1] As the nets gets deeper, the largest singular value tends to dominate, implying there is usually only one effective degree of freedom in the representations being computed.

Figure 6 demonstrates a related pathology that arises when composing functions to produce a deep density model. The density in the observed space eventually becomes locally concentrated onto one-dimensional manifolds, or *filaments*. This again suggests that, when the width of the network is relatively small, deep compositions of independent functions are unsuitable for modeling manifolds whose underlying dimensionality is greater than one.

[1] **?** analyzed the Jacobian at location of the training points, but because the priors we are examining are stationary, the distribution of the Jacobian is identical everywhere.

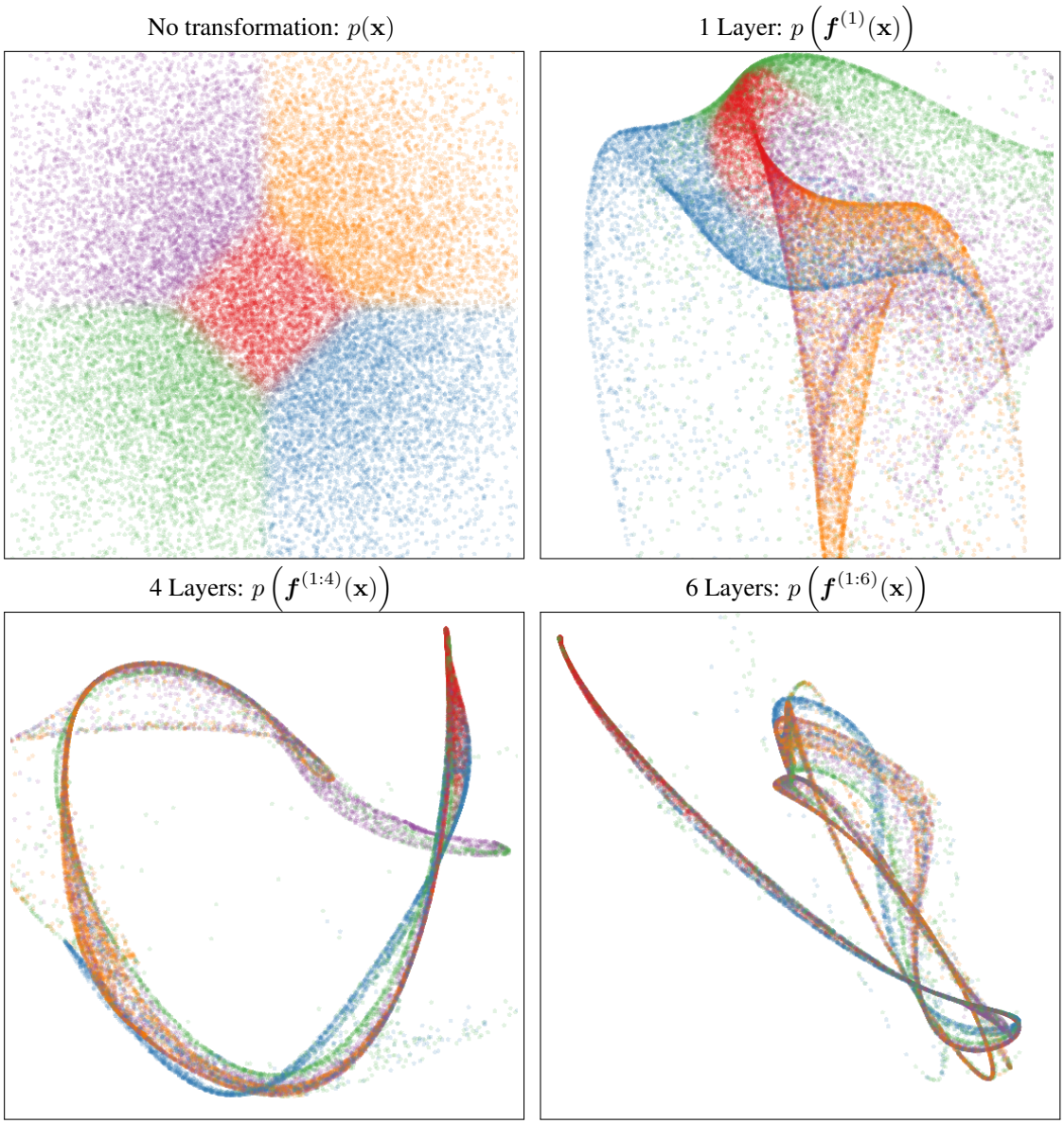


Figure 6: Points warped by a function drawn from a deep GP prior. *Top left:* Points drawn from a 2-dimensional Gaussian distribution, color-coded by their location. *Subsequent panels:* Those same points, successively warped by compositions of functions drawn from a GP prior. As the number of layers increases, the density concentrates along one-dimensional filaments. Warpings using random finite neural networks exhibit the same pathology, but also tend to concentrate density into 0-dimensional manifolds (points) due to saturation of all of the hidden units.

To visualize this pathology in another way, figure 7 illustrates a color-coding of the representation computed by a deep GP, evaluated at each point in the input space. After 10 layers, we can see that locally, there is usually only one direction that one can move in $\mathbf{x}$-space in order to change the value of the computed representation, or to cross a decision boundary. This means that such representations are likely to be unsuitable for decision tasks that depend on more than one property of the input.

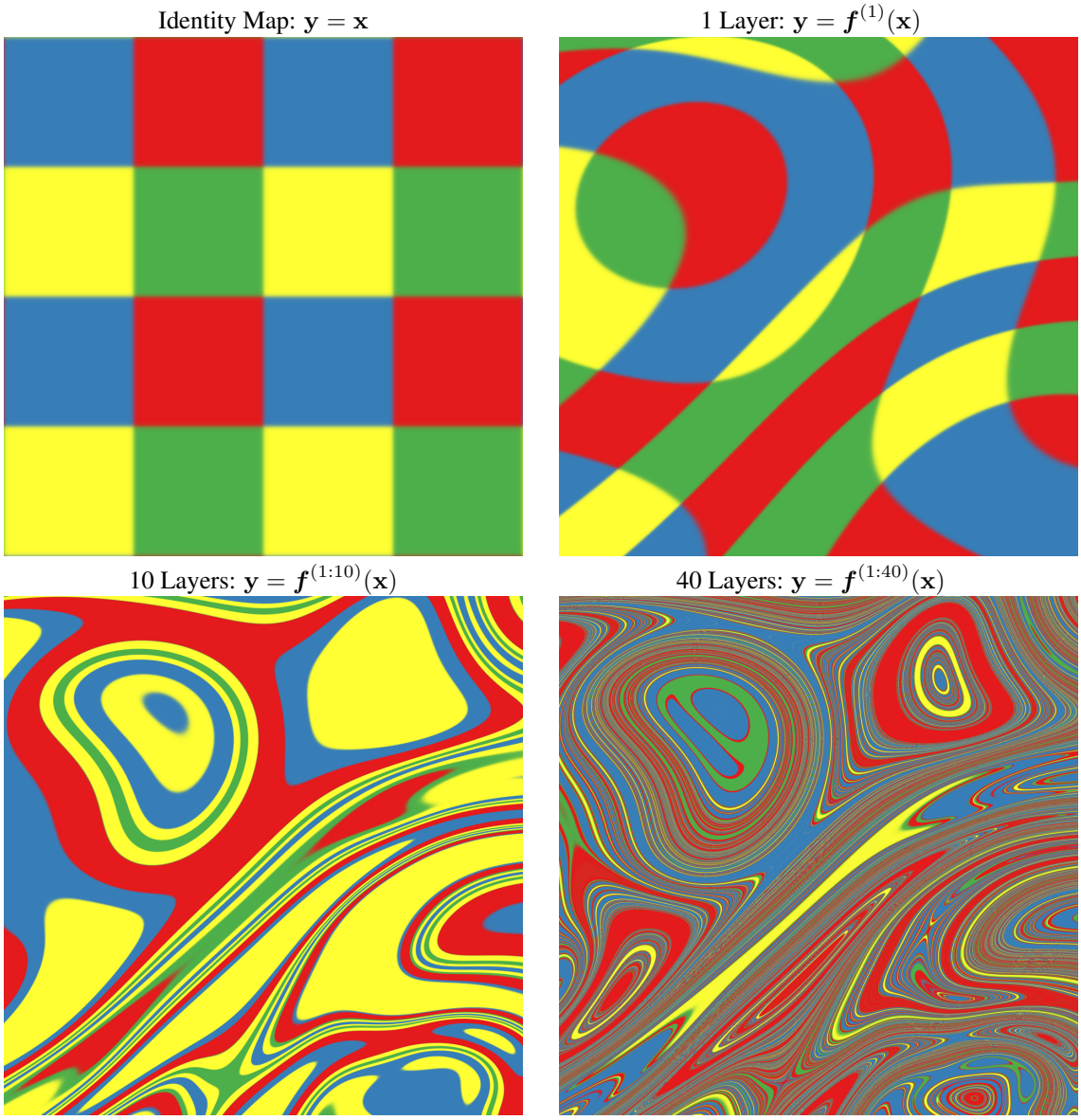


Figure 7: A visualization of the feature map implied by a function $\boldsymbol{f}$ drawn from a deep GP. Colors are a function of the 2D representation $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$ that each point is mapped to. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian. Just as the densities in figure 6 became locally one-dimensional, there is usually only one direction that one can move $\mathbf{x}$ in locally to change $\mathbf{y}$. This means that $\boldsymbol{f}$ is unlikely to be a suitable representation for decision tasks that depend on more than one aspect of $\mathbf{x}$. Also note that the overall shape of the mapping remains the same as the number of layers increase. For example, a roughly circular shape remains in the top-left corner even after 40 independent warpings.

To what extent are these pathologies present in the types of neural networks commonly used in practice? In simulations, we found that for deep functions with a fixed hidden dimension D, the singular value spectrum remained relatively flat for hundreds of layers as long as $D > 100$. Thus, these pathologies are unlikely to severely effect the relatively shallow, wide networks most commonly used in practice.

4 Fixing the pathology

As suggested by **?**, chapter 2, we can fix the pathologies exhibited in figures figure 6 and 7 by simply making each layer depend not only on the output of the previous layer, but also on the original input $\mathbf{x}$. We refer to these models as *input-connected* networks, and denote deep functions having this architecture with the subscript C, as in $f_C(\mathbf{x})$. Formally, this functional dependence can be written as

$$\boldsymbol{f}_C^{(1:L)}(\mathbf{x}) = \boldsymbol{f}^{(L)}\left(\boldsymbol{f}_C^{(1:L-1)}(\mathbf{x}), \mathbf{x}\right), \quad \forall L \tag{16}$$

Figure 8 shows a graphical representation of the two connectivity architectures.

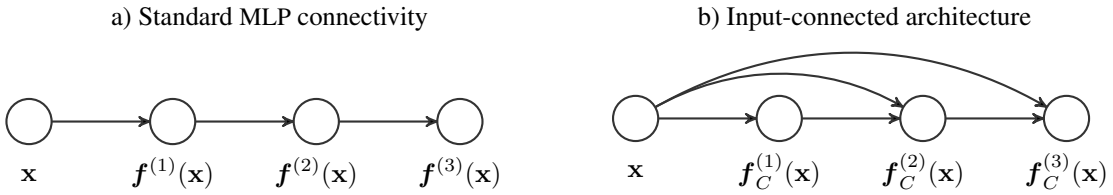


Figure 8: Two different architectures for deep neural networks. *Left:* The standard architecture connects each layer's outputs to the next layer's inputs. *Right:* The input-connected architecture also connects the original input $\mathbf{x}$ to each layer.

Similar connections between non-adjacent layers can also be found the primate visual cortex [**?**]. Visualizations of the resulting prior on functions are shown in figures 9, 10 and 12.

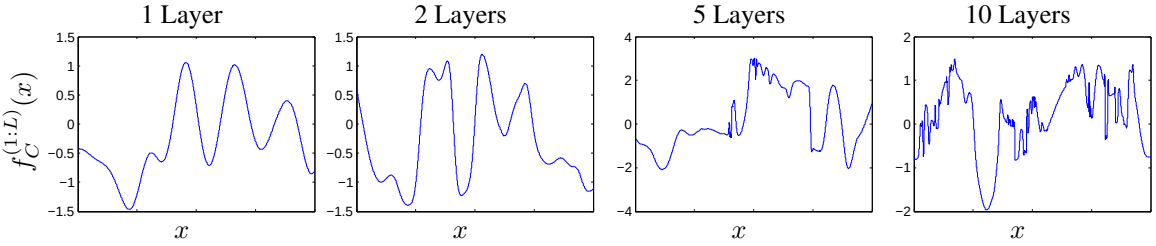


Figure 9: A draw from a 1D deep GP prior having each layer also connected to the input. The x-axis is the same for all plots. Even after many layers, the functions remain relatively smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 3.

The Jacobian of an input-connected deep function is defined by the recurrence

$$J_C^{(1:L)} = J^{(L)} \begin{bmatrix} J_C^{(1:L-1)} \\ I_D \end{bmatrix}. \tag{17}$$

where I_D is a D-dimensional identity matrix. Thus the Jacobian of an input-connected deep GP is a product of independent Gaussian matrices each with an identity matrix appended. Figure 11 shows that with this architecture, even 50-layer deep GPs have well-behaved singular value spectra.

The pathology examined in this section is an example of the sort of analysis made possible by a well-defined prior on functions. The figures and analysis done in this section could be done using Bayesian neural networks with finite numbers of nodes, but would be more difficult. In particular, care would need to be taken to ensure that the networks do not produce degenerate mappings due to saturation of the hidden units.

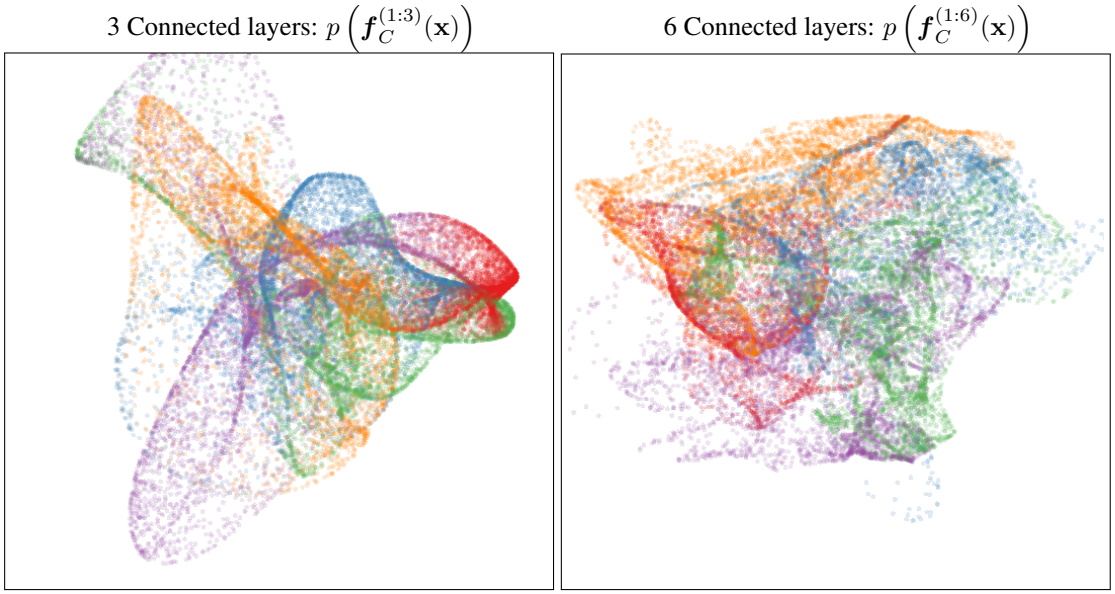


Figure 10: Points warped by a draw from a deep GP with each layer connected to the input $\mathbf{x}$. As depth increases, the density becomes more complex without concentrating only along one-dimensional filaments.

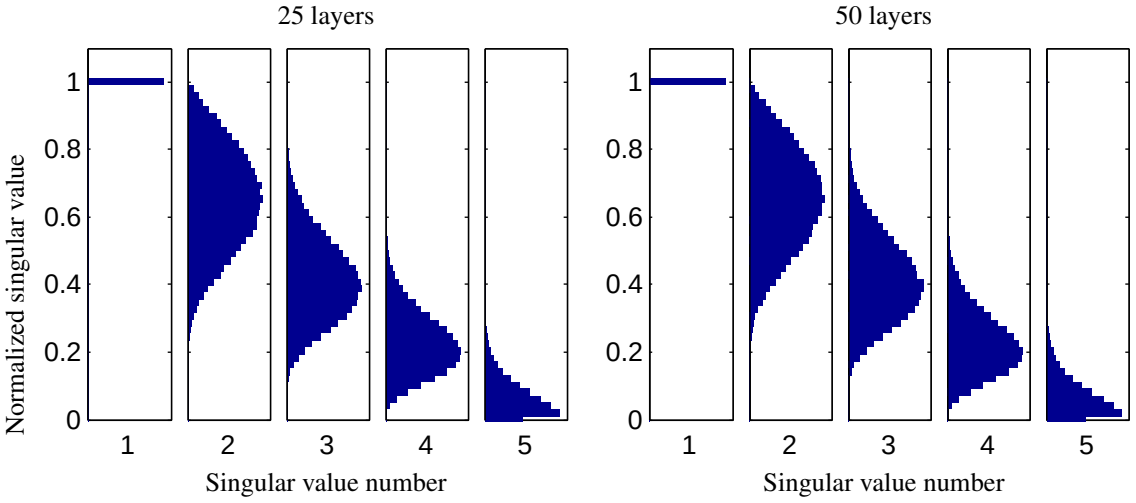


Figure 11: The distribution of singular values drawn from 5-dimensional input-connected deep GP priors, 25 layers deep (*Left*) and 50 layers deep (*Right*). Compared to the standard architecture, the singular values are more likely to remain the same size as one another, meaning that the model outputs are more often sensitive to several directions of variation in the input.

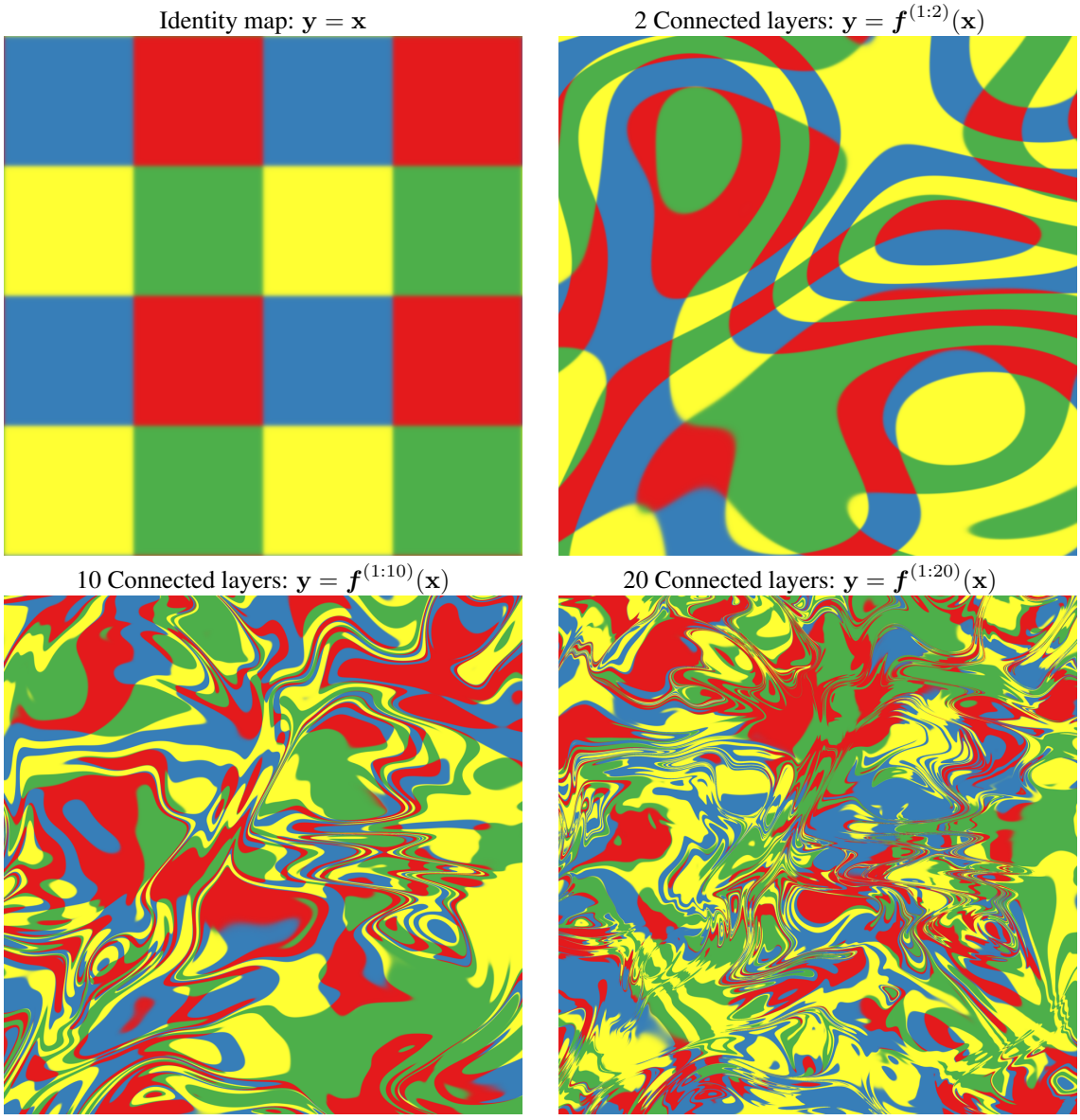


Figure 12: The feature map implied by a function $\boldsymbol{f}$ drawn from a deep GP prior with each layer also connected to the input $\mathbf{x}$, visualized at various depths. Compare to the map shown in figure 7. In the mapping shown here there are sometimes two directions that one can move locally in $\mathbf{x}$ to in order to change the value of $\boldsymbol{f}(\mathbf{x})$. This means that the input-connected prior puts significant probability mass on a greater variety of types of representations, some of which depend on all aspects of the input.

5 Deep kernels

? showed that kernel machines have limited generalization ability when they use "local" kernels such as the squared-exp. However, many interesting non-local kernels can be constructed which allow some forms of extrapolation. One way to build non-local kernels is by composing fixed feature maps, creating *deep kernels*. For example, periodic kernels can be viewed as a 2-layer-deep kernel, in which the first layer maps $x \to [\sin(x), \cos(x)]$, and the second layer maps through basis functions corresponding to the implicitly feature map giving rise to an SE kernel.

This section builds on the work of **?**, who derived several kinds of deep kernels by composing multiple layers of feature mappings.

In principle, one can compose the implicit feature maps of any two kernels k_a and k_b to get a new kernel, which we denote by $(k_b \circ k_a)$:

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{h}_a(\mathbf{x})^\mathsf{T} \mathbf{h}_a(\mathbf{x}') \tag{18}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{h}_b(\mathbf{x})^\mathsf{T} \mathbf{h}_b(\mathbf{x}') \tag{19}$$

$$(k_b \circ k_a)(\mathbf{x}, \mathbf{x}') = k_b\big(\mathbf{h}_a(\mathbf{x}), \mathbf{h}_a(\mathbf{x}')\big) = \left[\mathbf{h}_b\left(\mathbf{h}_a(\mathbf{x})\right)\right]^\mathsf{T} \mathbf{h}_b\left(\mathbf{h}_a(\mathbf{x}')\right) \tag{20}$$

However, this composition might not always have a closed form if the number of hidden features of either kernel is infinite.

Fortunately, composing the squared-exp (SE) kernel with the implicit mapping given by any other kernel has a simple closed form. If $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T} \mathbf{h}(\mathbf{x}')$, then

$$(\text{SE} \circ k)(\mathbf{x}, \mathbf{x}') = k_{SE}\big(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')\big) \tag{21}$$

$$= \exp\left(-\frac{1}{2}||\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')||_2^2\right) \tag{22}$$

$$= \exp\left(-\frac{1}{2}\left[\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\mathsf{T}\mathbf{h}(\mathbf{x}')\right]\right) \tag{23}$$

$$= \exp\left(-\frac{1}{2}\left[k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{x}') + k(\mathbf{x}', \mathbf{x}')\right]\right) . \tag{24}$$

This formula expresses the composed kernel $(\text{SE} \circ k)$ exactly in terms of evaluations of the original kernel k.

5.1 Infinitely deep kernels

What happens when one repeatedly composes feature maps many times, starting with the squared-exp kernel? If the output variance of the SE is normalized to $k(\mathbf{x}, \mathbf{x}) = 1$, then the infinite limit of composition with SE converges to $(\text{SE} \circ \text{SE} \circ \ldots \circ \text{SE})(\mathbf{x}, \mathbf{x}') = 1$ for all pairs of inputs. A constant covariance corresponds to a prior on constant functions $f(\mathbf{x}) = c$. This can be viewed as a degenerate limit.

As above, we can overcome this degeneracy by connecting the input $\mathbf{x}$ to each layer. To do so, we concatenate the composed feature vector at each layer, $\mathbf{h}^{(1:\ell)}(\mathbf{x})$, with the input vector $\mathbf{x}$ to produce an input-connected deep kernel $k_C^{(1:L)}$, defined by:

$$k_C^{(1:\ell+1)}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\left|\left|\begin{bmatrix}\mathbf{h}^{(1:\ell)}(\mathbf{x}) \\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}\mathbf{h}^{(1:\ell)}(\mathbf{x}') \\ \mathbf{x}'\end{bmatrix}\right|\right|_2^2\right) \tag{25}$$

$$= \exp\Big(-\frac{1}{2}\big[k_C^{(1:\ell)}(\mathbf{x}, \mathbf{x}) - 2k_C^{(1:\ell)}(\mathbf{x}, \mathbf{x}') + k_C^{(1:\ell)}(\mathbf{x}', \mathbf{x}') - ||\mathbf{x} - \mathbf{x}'||_2^2\big]\Big)$$

Starting with the squared-exp kernel, this repeated mapping satisfies

$$k_C^{(1:\infty)}(\mathbf{x}, \mathbf{x}') - \log\left(k_C^{(1:\infty)}(\mathbf{x}, \mathbf{x}')\right) = 1 + \frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2 . \tag{26}$$

The solution to this recurrence is related to the Lambert W-function [**?**] and has no closed form. In one

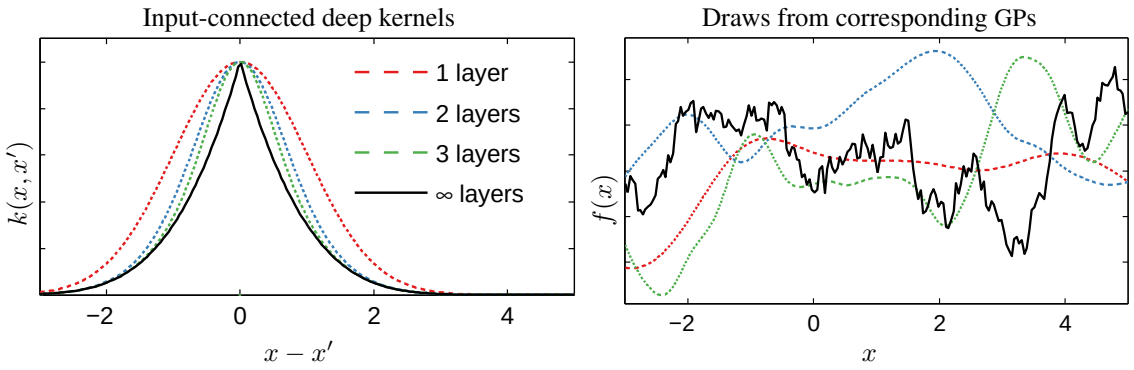


Figure 13: *Left:* Input-connected deep kernels of different depths. By connecting the input $\mathbf{x}$ to each layer, the kernel can still depend on its input even after arbitrarily many layers of composition. *Right:* Draws from GPs with deep input-connected kernels.

input dimension, it has a similar shape to the Ornstein-Uhlenbeck covariance $\mathrm{OU}(x, x') = \exp(-|x - x'|)$ but with lighter tails. Samples from a GP prior having this kernel are not differentiable, and are locally fractal. Figure 13 shows this kernel at different depths, as well as samples from the corresponding GP priors.

One can also consider two related connectivity architectures: one in which each layer is connected to the output layer, and another in which every layer is connected to all subsequent layers. It is easy to show that in the limit of infinite depth of composing SE kernels, both these architectures converge to $k(\mathbf{x}, \mathbf{x}') = \delta(\mathbf{x}, \mathbf{x}')$, the white noise kernel.

5.2 When are deep kernels useful models?

Kernels correspond to fixed feature maps, and so kernel learning is an example of implicit representation learning. Kernels can capture complex structure [**?**], and can enable many types of generalization, such as translation and rotation invariance in images [**?**]. More generally, **?** used a deep neural network to learn feature transforms for kernels, to learn invariances in an unsupervised manner. We believe that the relatively uninteresting properties of the deep kernels derived in this section simply reflect the fact that an arbitrary computation, even if it is "deep", is not likely to give rise to a useful representation unless combined with learning. To put it another way, any fixed representation is unlikely to be useful unless it has been chosen specifically for the problem at hand.

6 Dropout in Gaussian processes

Dropout is a recently-introduced method for regularizing neural networks [**??**]. Training with dropout entails independently setting to zero ("dropping") some proportion p of features or inputs, in order to improve the robustness of the resulting network, by reducing co-dependence between neurons. To maintain similar overall activation levels, the remaining weights are divided by p. Predictions are made by approximately averaging over all possible ways of dropping out neurons.

? and **?** analyzed dropout in terms of the effective prior induced by this procedure in several models, such as linear and logistic regression. In this section, we perform a similar analysis for GPs, examining the priors on functions that result from performing dropout in the one-hidden-layer neural network implicitly defined by a GP.

Recall from section 1 that some GPs can be derived as infinitely-wide one-hidden-layer neural networks, with fixed activation functions $\mathbf{h}(\mathbf{x})$ and independent random weights $\mathbf{w}$ having zero mean and finite variance $\sigma_{\mathbf{w}}^2$:

$$f(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^{K} w_i h_i(\mathbf{x}) \implies f \overset{K \to \infty}{\sim} \mathcal{GP}\big(0, \sigma_{\mathbf{w}}^2 \mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}')\big)\,. \tag{27}$$

6.1 Dropout on infinitely-wide hidden layers has no effect

First, we examine the prior obtained by dropping features from $\mathbf{h}(\mathbf{x})$ by setting weights in $\mathbf{w}$ to zero independently with probability p. For simplicity, we assume that $\mathbb{E}[\mathbf{w}] = \mathbf{0}$. If the weights w_i initially have finite variance σ_w^2 before dropout, then the weights after dropout (denoted by $r_i w_i$, where r_i is a Bernoulli random variable) will have variance:

$$r_i \overset{\text{iid}}{\sim} \text{Ber}(p) \qquad \mathbb{V}[r_i w_i] = p\sigma_w^2 \,. \tag{28}$$

Because equation (27) is a result of the central limit theorem, it does not depend on the exact form of the distribution on $\mathbf{w}$, but only on its mean and variance. Thus the central limit theorem still applies. Performing dropout on the features of an infinitely-wide MLP does not change the resulting model at all, except to rescale the output variance. Indeed, dividing all weights by $\sqrt{p}$ restores the initial variance:

$$\mathbb{V}\left[\frac{1}{\sqrt{p}} r_i w_i\right] = \frac{p}{p}\sigma_w^2 = \sigma_w^2 \tag{29}$$

in which case dropout on the hidden units has no effect at all. Intuitively, this is because no individual feature can have more than an infinitesimal contribution to the network output.

This result does not hold in neural networks having a finite number of hidden features with Gaussian-distributed weights, another model class that also gives rise to GPs.

6.2 Dropout on inputs gives additive covariance

One can also perform dropout on the D inputs to the GP. For simplicity, consider a stationary product kernel $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(x_d, x_d')$ which has been normalized such that $k(\mathbf{x}, \mathbf{x}) = 1$, and a dropout probability of $p = 1/2$. In this case, the generative model can be written as:

$$\mathbf{r} = [r_1, r_2, \ldots, r_D], \quad \text{each } r_i \overset{\text{iid}}{\sim} \text{Ber}\left(\frac{1}{2}\right), \quad f(\mathbf{x})|\mathbf{r} \sim \mathcal{GP}\left(0, \prod_{d=1}^{D} k_d(x_d, x_d')^{r_d}\right) \tag{30}$$

This is a mixture of 2^D GPs, each depending on a different subset of the inputs:

$$p(f(\mathbf{x})) = \sum_{\mathbf{r}} p(f(\mathbf{x})|\mathbf{r})\, p(\mathbf{r}) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \mathcal{GP}\left(f(\mathbf{x}) \,\middle|\, 0, \prod_{d=1}^{D} k_d(x_d, x_d')^{r_d}\right) \tag{31}$$

We present two results which might give intuition about this model.

Interpretation as a spike-and-slab prior on lengthscales First, if the kernel on each dimension has the form $k_d(x_d, x_d') = g\left(\frac{x_d - x_d'}{\ell_d}\right)$, as does the SE kernel, then any input dimension can be dropped out by setting its lengthscale ℓ_d to ∞. In this case, performing dropout on the inputs of a GP corresponds to putting independent spike-and-slab priors on the lengthscales, with each dimension's distribution independently having "spikes" at $\ell_d = \infty$ with probability mass of $1/2$.

Interpretation as an additive GP Another way to understand the resulting prior is to note that the dropout mixture given by equation (31) has the following covariance:

$$\text{cov}\left(\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix}\right) = \frac{1}{2^D} \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(x_d, x_d')^{r_d} \tag{32}$$

For dropout rates $p \neq 1/2$, the dth order terms will be weighted by $p^{(D-d)}(1-p)^d$.

Therefore, performing dropout on the inputs of a GP gives a non-Gaussian distribution that has the same first two moments as a GP having a covariance given by equation (32). This model class is called *additive* GPs, and have the property that they can sometimes allow non-local extrapolation [**?**]. To see why, note that a

GP whose covariance is a sum of kernels corresponds to a sum of functions, each distributed according to a GP having the corresponding kernel. Therefore, equation (32) describes a prior on a sum of 2^D functions, each depending on a different subset of input variables. Functions which depend only on a small number of input dimensions will be invariant to noise in all dimensions on which they don't depend. Isocontours of the resulting kernels are shown in Figure 14.

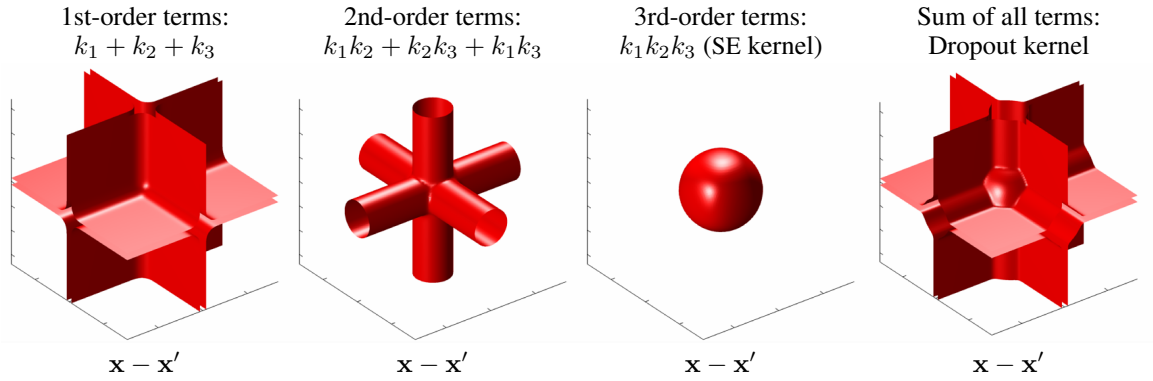


Figure 14: Isocontours of dropout kernel in $D = 3$ dimensions. The Dth-order kernel only considers nearby points relevant, while lower-order kernels allow the output to depend on distant training points, as long as they share one or more input value with a test point. Figure taken from **?**.

Thus intepreting dropout as a mixture of models, in which each model only depends on a subset of the input variables, helps to explain why dropout sometimes leads to better generalization.

7 Related work

7.0.1 Deep Gaussian processes

?, chapter 2 explored properties of arbitrarily deep Bayesian neural networks, including those that would give rise to deep GPs. He noted that infinitely deep random neural networks without extra connections to the input would be equivalent to a Markov chain, and therefore would lead to degenerate priors on functions. He also suggested connecting the input to each layer in order to fix this problem. Much of the analysis in this paper can be seen as a more detailed investigation, and vindication, of these claims.

The first instance of deep GPs being used in practice was [**?**], who presented a model called "hierarchical GP-LVMs", in which time was mapped through a composition of multiple GPs to produce observations.

The term "deep Gaussian processes" was first used by **?**, who developed a variational inference method, analyzed the effect of automatic relevance determination, and showed that deep GPS could learn with relatively little data. They used the term "deep GP" to refer both to supervised models (compositions of GPs) and to unsupervised models (compositions of GP-LVMs). This conflation may be reasonable, since the activations of the hidden layers are themselves latent variables, even in supervised settings: Depending on kernel parameters, each latent variable may or may not depend on the layer below.

In general, supervised models can also be latent-variable models. For example, **?** investigated single-layer GP regression models that had additional latent inputs.

7.0.2 Nonparametric neural networks

? proposed a prior on arbitrarily deep Bayesian networks having an unknown and unbounded number of parametric hidden units in each layer. Their architecture has connections only between adjacent layers, and so may have similar pathologies to the one discussed in this paper as the number of layers increases.

? introduced Gaussian process regression networks, which are defined as a matrix product of draws from GPs priors, rather than a composition. These networks have the form:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \qquad \text{where each } f_d, W_{d,j} \overset{\text{iid}}{\sim} \mathcal{GP}(\mathbf{0}, \text{SE}) + \mathcal{N}\left(0, \sigma_n^2\right). \tag{33}$$

We can easily define a "deep" Gaussian process regression network:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}^{(3)}(\mathbf{x})\mathbf{W}^{(2)}(\mathbf{x})\mathbf{W}^{(1)}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \tag{34}$$

which repeatedly adds and multiplies functions drawn from GPs, in contrast to deep GPs, which repeatedly compose functions. This prior on functions has a similar form to the Jacobian of a deep GP (equation (15)), and so might be amenable to a similar analysis to that of section 2.

7.0.3 Information-preserving architectures

Deep density networks [**?**] are constructed through a series of parametric warpings of fixed dimension, with penalty terms encouraging the preservation of information about lower layers. This is another promising approach to fixing the pathology discussed in section 3.

7.0.4 Recurrent networks

? and **?** analyzed a related problem with gradient-based learning in recurrent networks, the "exploding-gradients" problem. They noted that in recurrent neural networks, the size of the training gradient can grow or shrink exponentially as it is back-propagated, making gradient-based training difficult.

? addressed the exploding-gradients problem by introducing hidden units designed to have stable gradients. This architecture is known as long short-term memory.

7.0.5 Deep kernels

The first systematic examination of deep kernels was done by **?**, who derived closed-form composition rules for SE, polynomial, and arc-cosine kernels, and showed that deep arc-cosine kernels performed competitively in machine-vision applications when used in a SVM.

? constructed deep kernels in a time-series setting, constructing kernels corresponding to infinite-width *recurrent* neural networks. They also proposed concatenating the implicit feature vectors from previous time-steps with the current inputs, resulting in an architecture analogous to the input-connected architecture proposed by **?**, chapter 2.

7.0.6 Analyses of deep learning

? performed a layer-wise analysis of deep networks, and noted that the performance of MLPs degrades as the number of layers with random weights increases, a result consistent with the analysis of this paper.

The experiments of **?** suggested that most of the good performance of convolutional neural networks could be attributed to the architecture alone. Later, **?** looked at the dynamics of gradient-based training methods in deep *linear* networks as a tractable approximation to standard deep (nonlinear) neural networks.

7.0.7 Source code

Source code to produce all figures is available at `http://www.github.com/duvenaud/deep-limits`. This code is also capable of producing visualizations of mappings such as figures 7 and 12 using neural nets instead of GPs at each layer.

8 Conclusions

This paper used well-defined priors to explicitly examine the assumptions made by neural network models. We used deep Gaussian processes as an easy-to-analyze model corresponding to multi-layer preceptrons

having nonparametric activation functions. We showed that representations based on repeated composition of independent functions exhibit a pathology where the representations becomes invariant to all but one direction of variation. We then showed that this problem could be alleviated by connecting the input to each layer.

We also examined properties of deep kernels, corresponding to arbitrarily many compositions of fixed feature maps. Finally, we derived models obtained by performing dropout on Gaussian processes, finding a tractable approximation to exact dropout in GPs.

Much recent work on deep networks has focused on weight initialization [**?**], regularization [**?**] and network architecture [**?**]. If we can identify priors that give our models desirable properties, these might in turn suggest regularization, initialization, and architecture choices that also provide such properties.

Acknowledgements

We thank Carl Rasmussen, Andrew McHutchon, Neil Lawrence, Andreas Damianou, James Robert Lloyd, Creighton Heaukulani, Dan Roy, Mark van der Wilk, Miguel Hernández-Lobato and Andrew Tulloch for helpful discussions.

References

Ryan P. Adams, Hanna M. Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2010.

Pierre Baldi and Peter J. Sadowski. Understanding dropout. In *Advances in Neural Information Processing Systems*, pages 2814–2822, 2013.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 5(2):157–166, 1994.

Yoshua Bengio, Olivier Delalleau, and Nicolas Le Roux. The curse of highly variable functions for local kernel machines. *Advances in Neural Information Processing Systems*, 18:107–114, 2006. ISSN 1049-5258.

Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, pages 342–350, 2009.

Robert M. Corless, Gaston H. Gonnet, David E. G. Hare, David J. Jeffrey, and Donald E. Knuth. On the Lambert W function. *Advances in Computational Mathematics*, 5(1):329–359, 1996.

Andreas Damianou and Neil D. Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

David Duvenaud, Hannes Nickisch, and Carl E. Rasmussen. Additive Gaussian processes. In *Advances in Neural Information Processing Systems 24*, pages 226–234, Granada, Spain, 2011.

David Duvenaud, James Robert Lloyd, Roger B. Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, 2013.

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine learning*, 2013.

James Hensman, Andreas Damianou, and Neil D. Lawrence. Deep Gaussian processes for large datasets. In *Artificial Intelligence and Statistics Late-breaking Posters*, 2014.

Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012.

Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Imre Risi Kondor. *Group theoretical methods in machine learning*. PhD thesis, Columbia University, 2008.

Neil D. Lawrence and Andrew J. Moore. Hierarchical Gaussian process latent variable models. In *Proceedings of the 24th International Conference on Machine learning*, pages 481–488, 2007.

Honglak Lee, Chaitanya Ekanadham, and Andrew Ng. Sparse deep belief net model for visual area v2. In *Advances in Neural Information Processing Systems*, pages 873–880, 2007.

James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010.

John H. R. Maunsell and David C. van Essen. The connections of the middle temporal visual area (MT) and their relationship to a cortical hierarchy in the macaque monkey. *Journal of neuroscience*, 3(12): 2563–2586, 1983.

Grégoire Montavon, Mikio L. Braun, and Klaus-Robert Müller. Layer-wise analysis of deep networks with Gaussian kernels. *Advances in Neural Information Processing Systems*, 23:1678–1686, 2010.

Radford M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *arXiv preprint arXiv:1211.5063*, 2012.

Salah Rifai, Grégoire Mesnil, Pascal Vincent, Xavier Muller, Yoshua Bengio, Yann Dauphin, and Xavier Glorot. Higher order contractive auto-encoder. In *Machine Learning and Knowledge Discovery in Databases*, pages 645–660. Springer, 2011a.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning*, pages 833–840, 2011b.

Oren Rippel and Ryan P. Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. *Brain Mechanisms*, pages 555–559, 1962.

Ruslan Salakhutdinov and Geoffrey Hinton. Using deep belief nets to learn covariance kernels for Gaussian processes. *Advances in Neural information processing systems*, 20:1249–1256, 2008.

Andrew Saxe, Pang W. Koh, Zhenghao Chen, Maneesh Bhand, Bipin Suresh, and Andrew Y. Ng. On random weights and unsupervised feature learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1089–1096, 2011.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Dynamics of learning in deep linear neural networks. In *NIPS Workshop on Deep Learning*, 2013.

Ercan Solak, Roderick Murray-Smith, William E. Leithead, Douglas J. Leith, and Carl E. Rasmussen. Derivative observations in Gaussian process models of dynamic systems. In *Advances in Neural Information Processing Systems*, 2003.

Nitish Srivastava. Improving neural networks with dropout. Master's thesis, University of Toronto, 2013.

Chunyi Wang and Radford M. Neal. Gaussian process regression with heteroscedastic or non-gaussian residuals. *arXiv preprint arXiv:1212.6246*, 2012.

Sida Wang and Christopher Manning. Fast dropout training. In *Proceedings of the 30th International Conference on Machine Learning*, pages 118–126, 2013.

Andrew G. Wilson, David A. Knowles, and Zoubin Ghahramani. Gaussian process regression networks. In *Proceedings of the 29th International Conference on Machine Learning*, pages 599–606, 2012.