

# Chapter 1

## Deep Gaussian Processes

“I asked myself: On any given day, would I rather be wrestling with a sampler, or proving theorems?”

– Peter Orbanz, personal communication

Choosing appropriate architectures and regularization strategies of deep networks is important for good predictive performance. In this chapter, we propose to study this problem by viewing deep nets as priors on functions. By viewing neural networks this way, we can analyze their properties without reference to any particular dataset, loss function, or training method. Instead, we can ask what sorts of information-processing structures these priors give rise to, and check whether those structures are the same sort which we expect to find in useful models.

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. Firstly, [Damianou and Lawrence \(2013\)](#) showed that the probabilistic nature of deep GPs guards against overfitting. Second, [Hensman et al. \(2014\)](#) showed that stochastic variational inference is possible in deep GPs, allowing mini-batch training. Third, the availability of an approximate marginal likelihood allows us 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.

The ideas contained in this chapter were developed through discussions with Oren Rippel, Ryan Adams and Zoubin Ghahramani, and appear in [Duvenaud et al. \(2014\)](#).

## 1.1 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.

### 1.1.1 Definition of deep GPs

We define a deep GP as a prior on functions constructed by composing draws from GP priors. An example of a deep GP is a composition of vector-valued functions, with each function in each layer drawn independently from GP priors:

$$\begin{aligned} \mathbf{f}^{(1:L)}(\mathbf{x}) &= \mathbf{f}^{(L)}\left(\mathbf{f}^{(L-1)}\left(\dots \mathbf{f}^{(2)}\left(\mathbf{f}^{(1)}(\mathbf{x})\right)\dots\right)\right) \\ \text{with each } f_d^{(\ell)} &\stackrel{\text{ind}}{\sim} \text{GP}\left(0, k_d^{(\ell)}(\mathbf{x}, \mathbf{x}')\right) \end{aligned} \quad (1.1)$$

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

### 1.1.2 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) ([Rosenblatt, 1962](#)). In the typical definition of an MLP with one hidden layer, the hidden unit activations are defined as:

$$\mathbf{h}(\mathbf{x}) = \sigma(\mathbf{b} + \mathbf{V}\mathbf{x}) \quad (1.2)$$

where  $\mathbf{h}$  are the hidden unit activations,  $\mathbf{b}$  is a bias vector,  $\mathbf{V}$  is a weight matrix and  $\sigma$  is a one-dimensional nonlinear function, usually sigmoidal, applied element-wise. The

output vector  $\mathbf{f}(\mathbf{x})$  is simply a weighted sum of these hidden unit activations:

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}\sigma(\mathbf{b} + \mathbf{V}\mathbf{x}) = \mathbf{W}\mathbf{h}(\mathbf{x}) \quad (1.3)$$

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

Neal (1995, chapter 2) showed that 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

$$f(\mathbf{x}) = \frac{1}{K}\mathbf{w}^\top \mathbf{h}(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^K w_i h_i(\mathbf{x}), \quad (1.4)$$

with fixed<sup>1</sup> features  $[h_1(\mathbf{x}), \dots, h_K(\mathbf{x})]^\top = \mathbf{h}(\mathbf{x})$  and i.i.d.  $w$ 's with zero mean and finite variance  $\sigma^2$ , 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:

$$\lim_{K \rightarrow \infty} p\left(\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \frac{\sigma^2}{K} \begin{bmatrix} \sum_{i=1}^K h_i(\mathbf{x})h_i(\mathbf{x}) & \sum_{i=1}^K h_i(\mathbf{x})h_i(\mathbf{x}') \\ \sum_{i=1}^K h_i(\mathbf{x}')h_i(\mathbf{x}) & \sum_{i=1}^K h_i(\mathbf{x}')h_i(\mathbf{x}') \end{bmatrix}\right) \quad (1.5)$$

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.

We can also work backwards to derive a one-layer MLP from any GP. Mercer's theorem implies that any positive-definite kernel function corresponds to an inner product of features:  $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}')$ . Thus in the one-hidden-layer case, the correspondence between MLPs and GPs is simple: the implicit features  $\mathbf{h}(\mathbf{x})$  of the kernel correspond to hidden units of an MLP.

An MLP with a finite number of fixed hidden nodes gives rise to a GP if and only if the weights  $\mathbf{w}$  are jointly Gaussian distributed.

---

<sup>1</sup>The above derivation gives the same result if the parameters of the hidden units are random, since their distribution on outputs always the same with probability one. However, to avoid confusion, we refer to layers with infinitely-many nodes as "fixed".

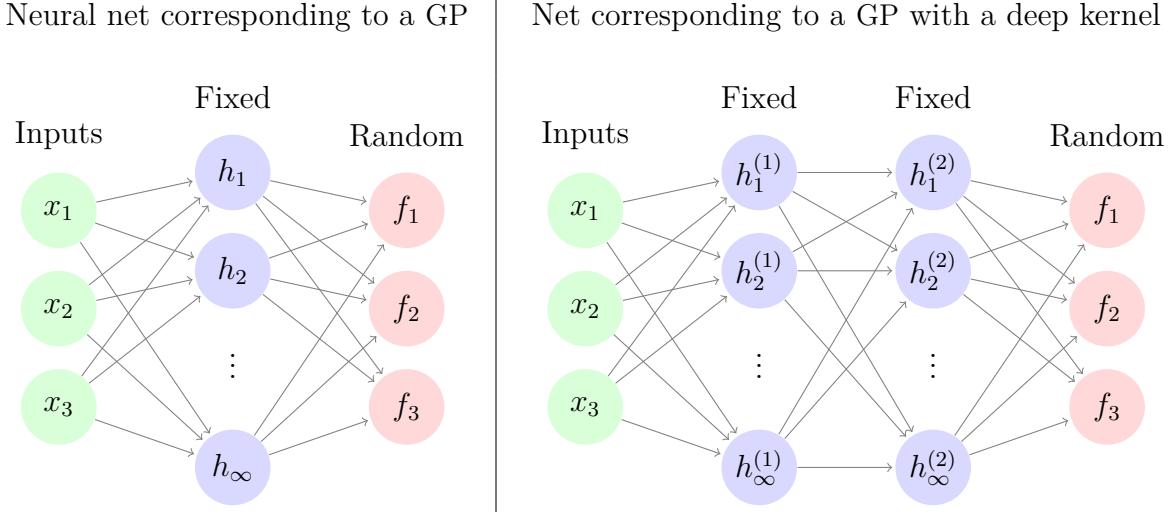


Figure 1.1 *Left:* GPs can be derived as a one-hidden-layer MLP with infinitely many fixed hidden units having unknown weights. *Right:* Multiple layers of fixed hidden units gives rise to a GP with a deep kernel, but not a deep GP.

### 1.1.3 Multiple hidden layers

Next, we examine infinitely-wide MLPs with 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  $\ell$ th layer units are given by

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma \left( \mathbf{b}^{(\ell)} + \mathbf{V}^{(\ell)} \mathbf{h}^{(\ell-1)}(\mathbf{x}) \right). \quad (1.6)$$

This architecture is shown on the right of figure 1.1. For example, if we extend the model given by equation (1.3) to have two layers of feature mappings, the resulting model becomes

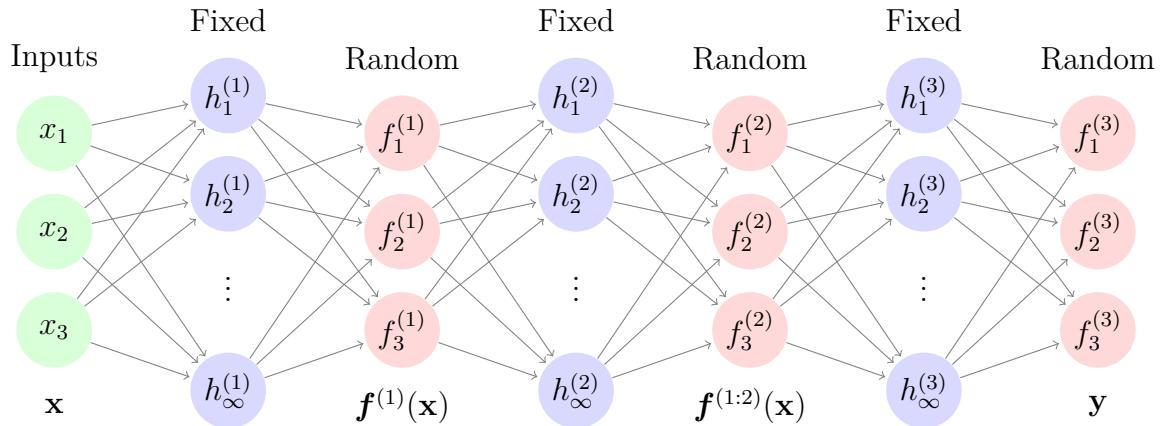
$$f(\mathbf{x}) = \frac{1}{K} \mathbf{w}^\top \mathbf{h}^{(2)} \left( \mathbf{h}^{(1)}(\mathbf{x}) \right). \quad (1.7)$$

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

$$k(\mathbf{x}, \mathbf{x}') = \left( \mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x})) \right)^\top \mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}')). \quad (1.8)$$

Deep kernels, explored in section 1.5, imply a fixed representation as opposed to a prior over representations. Thus, unless we richly parameterize these kernels, their

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

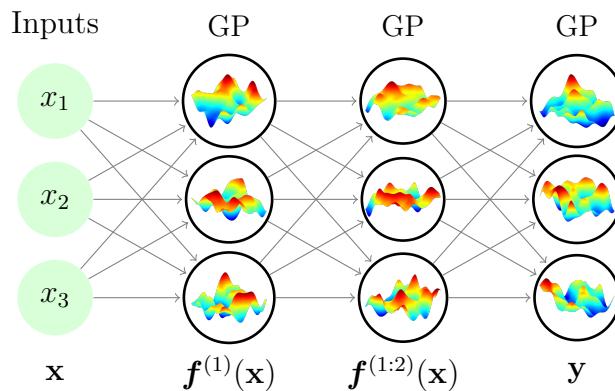


Figure 1.2 Two equivalent views of deep GPs as neural networks. *Top:* A neural network whose every second layer is a weighted sum of an infinite number of fixed hidden units, where the weights are initially unknown. *Bottom:* A neural network with a finite number of hidden units, each with a different 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.

capacity to learn an appropriate representation is limited in comparison to more flexible models such as deep neural networks or deep GPs.

#### 1.1.4 Two network architectures equivalent to deep GPs

There are two equivalent neural network architectures that correspond to deep GPs: one with fixed nonlinearities, and another with 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 1.1(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 1.2. The  $D^{(\ell)}$  outputs  $\mathbf{f}^{(\ell)}(\mathbf{x})$  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  $D^{(\ell)}$  outputs.

This alternating-layer architecture has an interpretation as a series of linear information bottlenecks. To see this, we can simply substitute equation (1.3) into equation (1.6) to get

$$\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma \left( \mathbf{b}^{(\ell)} + [\mathbf{V}^{(\ell)} \mathbf{W}^{(\ell-1)}] \mathbf{h}^{(\ell-1)}(\mathbf{x}) \right) \quad (1.9)$$

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

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  $\mathbf{f}^{(1:\ell)}(\mathbf{x})$  represent the output of the hidden nodes at the  $\ell^{th}$  layer. This second view lets us compare deep GP models to standard neural net architectures more directly. Figure 1.1(bottom) shows this architecture.

## 1.2 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 will also show that the Jacobian of a multivariate deep GP is a product of independent Gaussian matrices with independent entries. These results will allow us to identify a pathology that emerges in very deep networks in section 1.3.

### 1.2.1 One-dimensional asymptotics

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

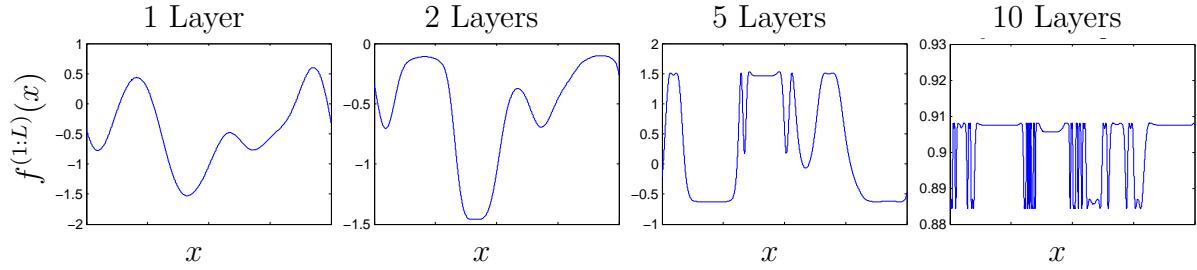


Figure 1.3 A one-dimensional draw from a deep GP prior shown at different depths. The  $x$ -axis is the same for all plots. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed. 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 a particular region 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.

$$\text{SE}(x, x') = \sigma^2 \exp\left(\frac{-(x - x')^2}{2w^2}\right). \quad (1.10)$$

The parameter  $\sigma^2$  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  $\mathcal{N}(0, \sigma^2/w^2)$ . 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 we choose kernel parameters so 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:

$$\begin{aligned} m_{\log} &= \mathbb{E} \left[ \log \left| \frac{\partial f(x)}{\partial x} \right| \right] = 2 \log \left( \frac{\sigma}{w} \right) - \log 2 - \gamma \\ v_{\log} &= \mathbb{V} \left[ \log \left| \frac{\partial f(x)}{\partial x} \right| \right] = \frac{\pi^2}{4} + \frac{\log^2 2}{2} - \gamma^2 - \gamma \log 4 + 2 \log \left( \frac{\sigma}{w} \right) \left[ \gamma + \log 2 - \log \left( \frac{\sigma}{w} \right) \right] \end{aligned} \quad (1.11)$$

where  $\gamma \approx 0.5772$  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 log-normal

as  $L$  grows:

$$\log \left| \frac{\partial f^{(1:L)}(x)}{\partial x} \right| = \log \prod_{\ell=1}^L \left| \frac{\partial f^{(\ell)}(x)}{\partial x} \right| = \sum_{\ell=1}^L \log \left| \frac{\partial f^{(\ell)}(x)}{\partial x} \right| \xrightarrow{L \rightarrow \infty} \mathcal{N}(Lm_{\log}, L^2 v_{\log}) \quad (1.12)$$

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 1.3 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.

### 1.2.2 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 1.2.1.** *The partial derivatives of a function mapping  $\mathbb{R}^D \rightarrow \mathbb{R}$  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 Solak et al. (2003):

$$\text{cov} \left( \frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}} \right) = \frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial x_{d_1} \partial x'_{d_2}} \Big|_{\mathbf{x}=\mathbf{x}'} \quad (1.13)$$

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.  $\square$

For example, in the case of the multivariate squared-exp kernel, the covariance be-

tween derivatives has the form:

$$\begin{aligned} f(\mathbf{x}) &\sim \text{GP} \left( 0, \sigma^2 \prod_{d=1}^D \exp \left( -\frac{1}{2} \frac{(x_d - x'_d)^2}{w_d^2} \right) \right) \\ \implies \text{cov} \left( \frac{\partial f(\mathbf{x})}{\partial x_{d_1}}, \frac{\partial f(\mathbf{x})}{\partial x_{d_2}} \right) &= \begin{cases} \frac{\sigma^2}{w_{d_1}^2} & \text{if } d_1 = d_2 \\ 0 & \text{if } d_1 \neq d_2 \end{cases} \end{aligned} \quad (1.14)$$

**Lemma 1.2.2.** *The Jacobian of a set of  $D$  functions  $\mathbb{R}^D \rightarrow \mathbb{R}$  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  $\mathbf{f}(\mathbf{x})$  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  $f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_D(\mathbf{x})$  are independent by construction, it follows that each row of  $J$  is independent. Lemma 1.2.1 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.  $\square$

**Theorem 1.2.3.** *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  $\ell$  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 simply the product of the immediate Jacobian matrices of each function. Thus the Jacobian of the composed (deep) function  $\mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\dots \mathbf{f}^{(3)}(\mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x})))) \dots))$  is

$$J^{(1:L)}(\mathbf{x}) = J^{(L)} J^{(L-1)} \dots J^{(3)} J^{(2)} J^{(1)}. \quad (1.15)$$

By lemma 1.2.2, each  $J_{i,j}^{(\ell)} \stackrel{\text{ind}}{\sim} \mathcal{N}$ , so the complete Jacobian is a product of independent Gaussian matrices, with each entry of each matrix drawn independently.  $\square$

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

## 1.3 Formalizing a pathology

A common use of deep neural networks is building useful representations of data manifolds. What properties make a representation useful? Rifai et al. (2011a) argue that

good representations of data manifolds are invariant in directions orthogonal to the data manifold. They also argue that, conversely, a good representation must also change in directions tangent to the data manifold, in order to preserve relevant information. Figure 1.4 visualizes such a representation.

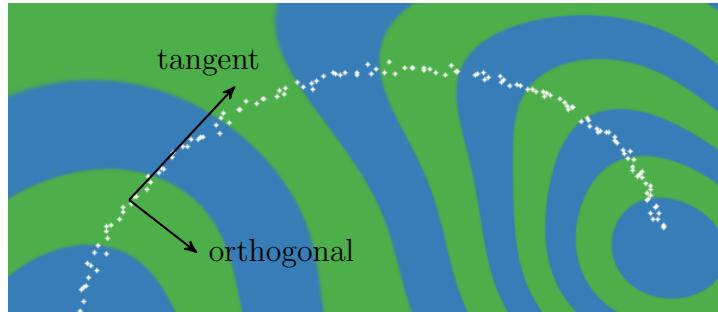


Figure 1.4 Representing a 1-D data manifold. Colors are a function of the computed representation of the input space. The representation (blue & green) varies in directions tangent to the data manifold (white), preserving information for later layers. The representation changes little in directions orthogonal to the manifold, making it robust to noise in those directions.

As in Rifai et al. (2011b), 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 roughly correspond to the number of directions in data-space in which the representation varies. Figure 1.5 shows the singular value spectrum for 5-dimensional deep GPs of different depths<sup>2</sup>. As the net gets deeper, the largest singular value dominates, implying there is usually only one effective degree of freedom in the representations being computed.

Figure 1.6 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.

To visualize this pathology in another way, figure 1.7 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

<sup>2</sup>Rifai et al. (2011b) 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.

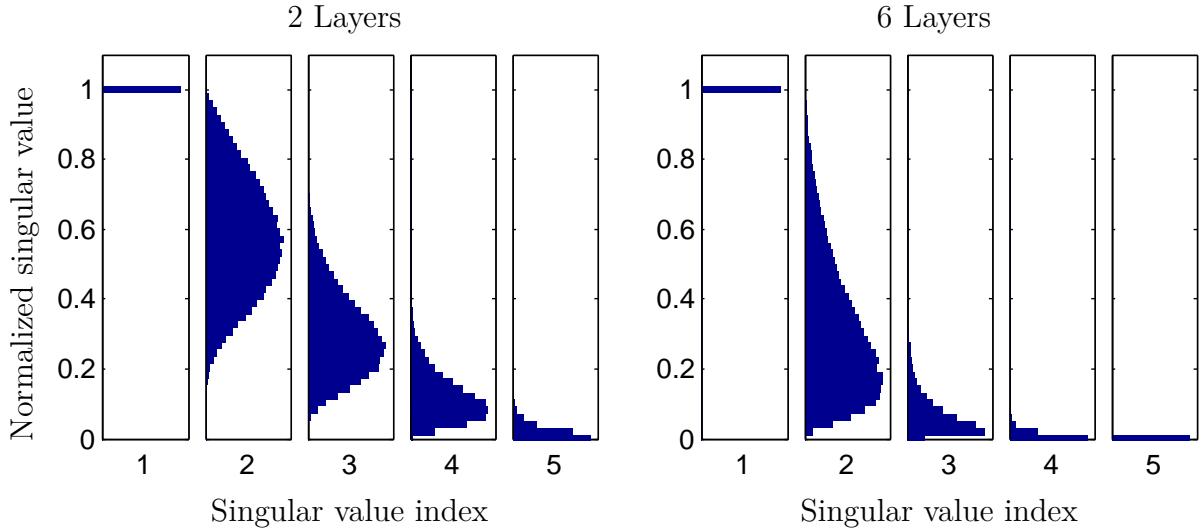


Figure 1.5 The distribution of normalized singular values of the Jacobian of a function drawn from a 5-dimensional deep GP prior 25 layers deep (*Left*) and 50 layers deep (*Right*). As nets get deeper, the largest singular value tends to become much larger than the others. This implies that with high probability, these functions vary little in all directions but one, making them unsuitable for computing representations of manifolds of more than one dimension.

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.

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.

## 1.4 Fixing the pathology

As suggested by [Neal \(1995, chapter 2\)](#), we can fix the pathologies exhibited in figures figure 1.6 and 1.7 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

$$f_C^{(1:L)}(\mathbf{x}) = f^{(L)} \left( f_C^{(1:L-1)}(\mathbf{x}), \mathbf{x} \right), \quad \forall L \quad (1.16)$$

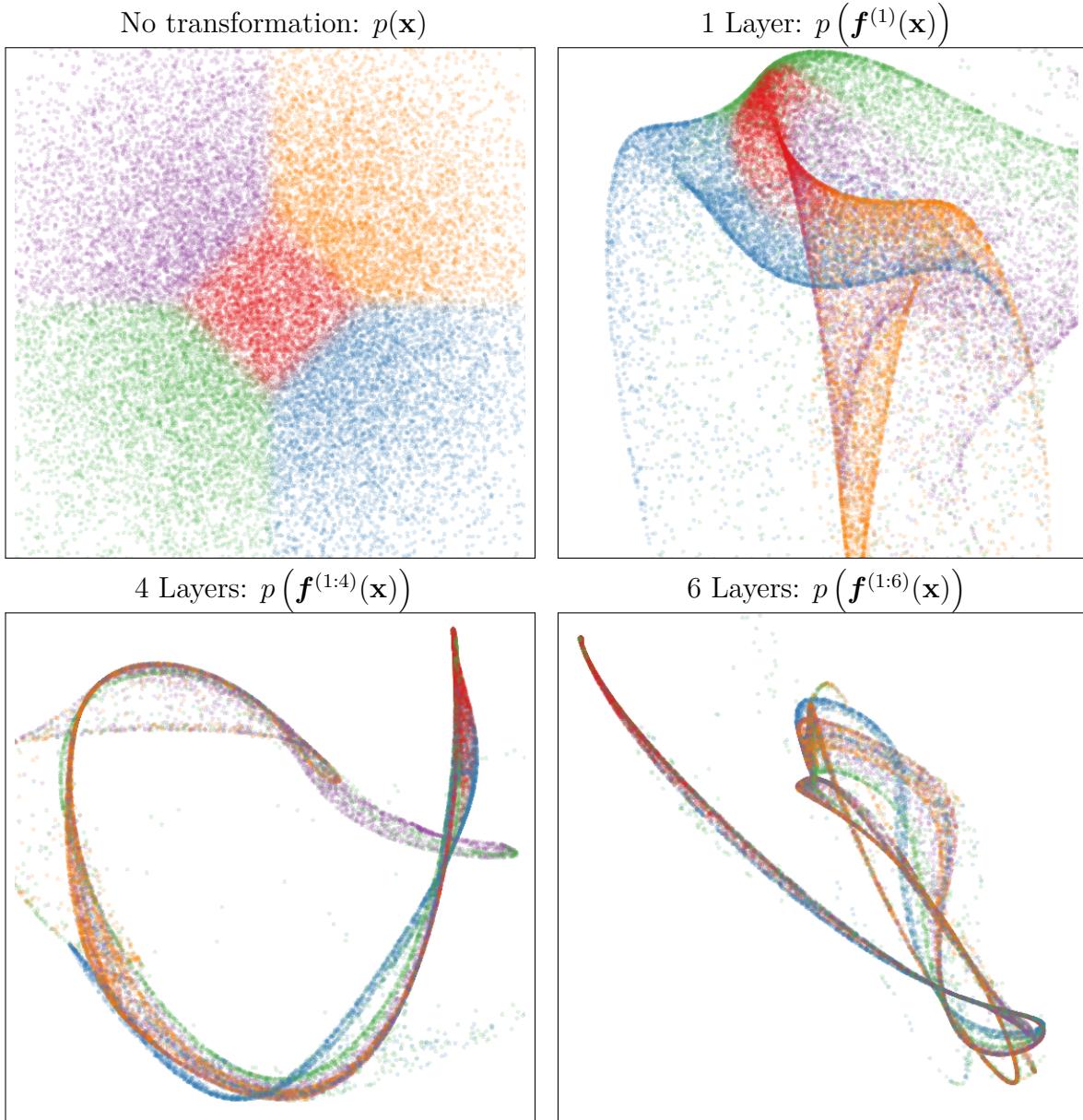


Figure 1.6 Points warped by a function drawn from a deep GP prior. *Top left:* Points drawn from a 2-dimensional Gaussian distribution, color-coded by their location. *Subsequent panels:* Those same points, successively warped by compositions of functions drawn from a GP prior. As the number of layers 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 the hidden units.

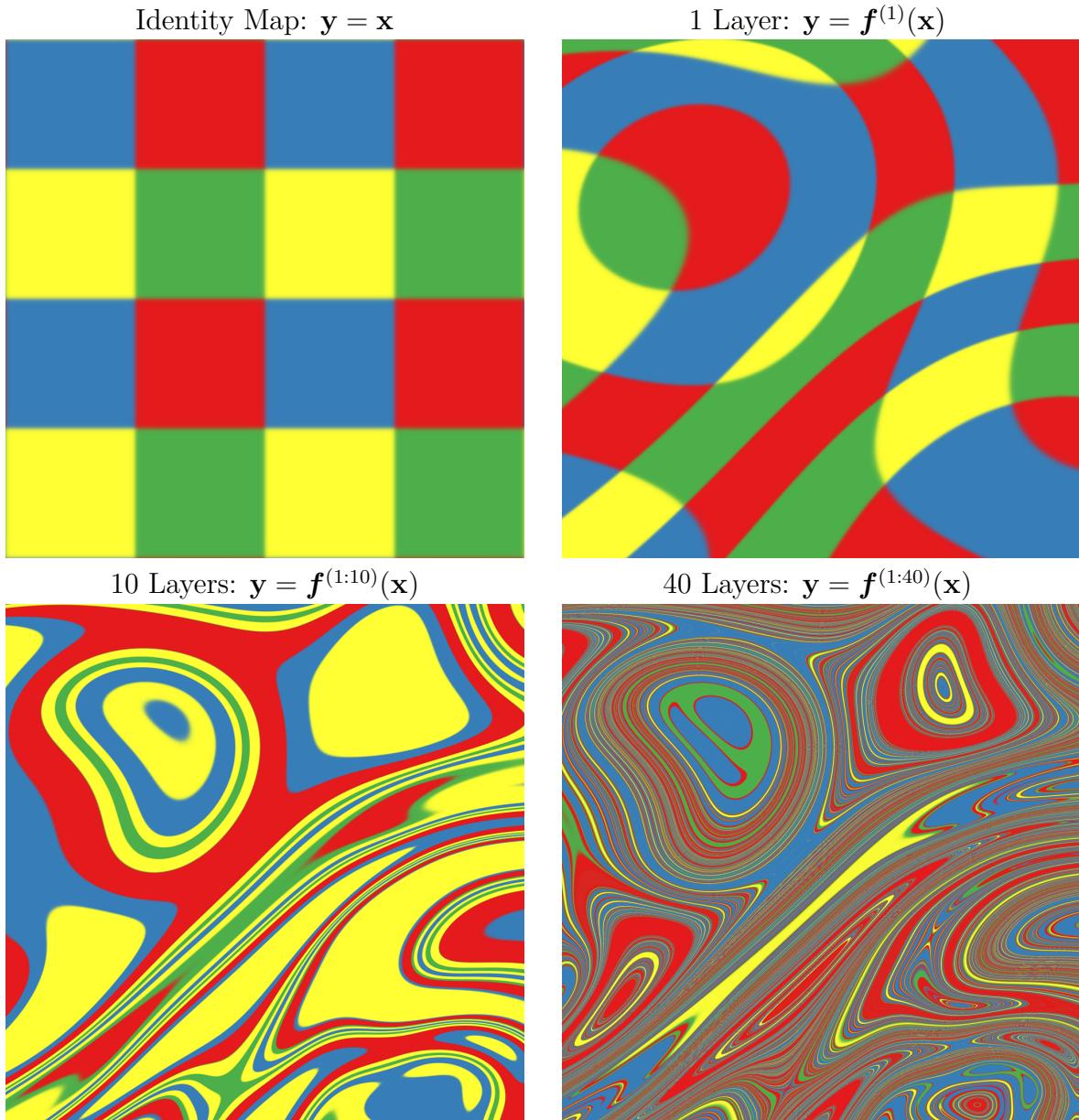
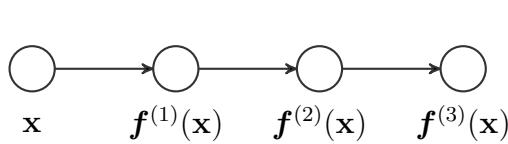


Figure 1.7 A visualization of the feature map implied by a function  $\mathbf{f}$  drawn from a deep GP. Colors are a function of the 2D representation  $\mathbf{y} = \mathbf{f}(\mathbf{x})$  that each point is mapped to. The number of directions in which the color changes rapidly corresponds to the number of large singular values in the Jacobian. Just as the densities in figure 1.6 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  $\mathbf{f}$  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, the roughly circular shape remains in the top-left corner even after 40 independent warps.

Figure 1.8 shows a graphical representation of the two connectivity architectures.

a) Standard MLP connectivity



b) Input-connected architecture

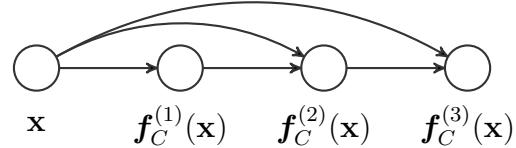


Figure 1.8 Two different architectures for deep neural networks. *Left:* The standard architecture connects each layer’s outputs to the next layer’s inputs. *Right:* The input-connected architecture also connects the original input  $\mathbf{x}$  to each layer.

Similar connections between non-adjacent layers can also be found in the primate visual cortex (Maunsell and van Essen, 1983). Visualizations of the resulting prior on functions are shown in figures 1.9, 1.10 and 1.12.

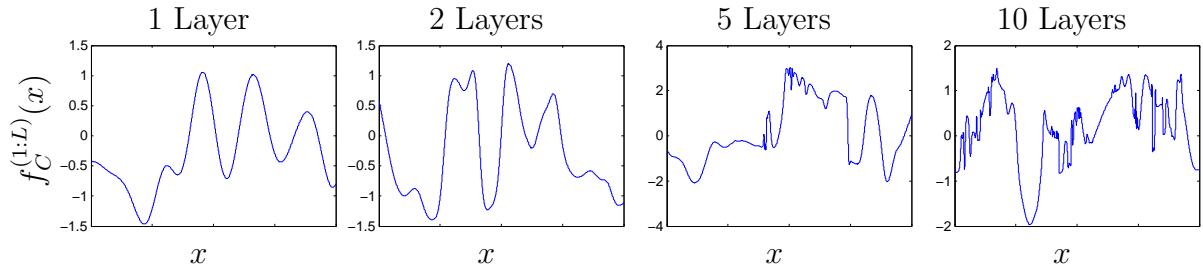


Figure 1.9 A draw from a 1D deep GP prior with each layer connected to the input. The  $x$ -axis is the same for all plots. Even after many layers, the functions remain relatively smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 1.3.

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

$$J_C^{(1:L)} = J^{(L)} \begin{bmatrix} J_C^{(1:L-1)} \\ I_D \end{bmatrix}. \quad (1.17)$$

where  $I_D$  is a  $D$ -dimensional identity matrix. In the case of a deep GP, the input-connected Jacobian is still a product of independent Gaussian matrices. Figure 1.11 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

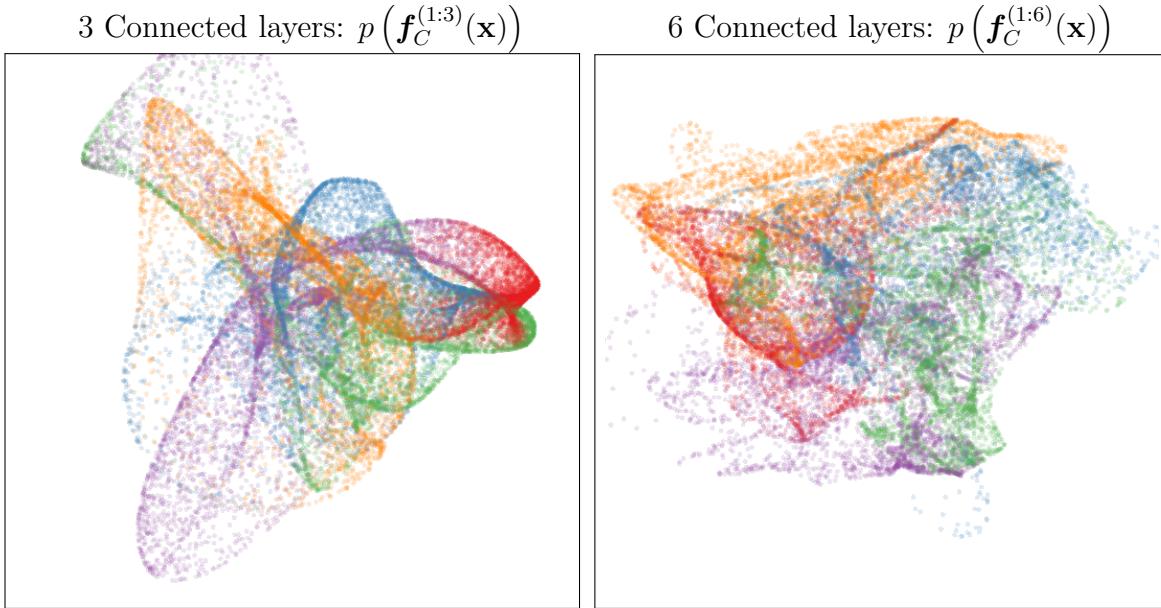


Figure 1.10 Points warped by a draw from a deep GP with each layer connected to the input  $\mathbf{x}$ . As depth increases, the density becomes more complex without concentrating only along one-dimensional filaments.

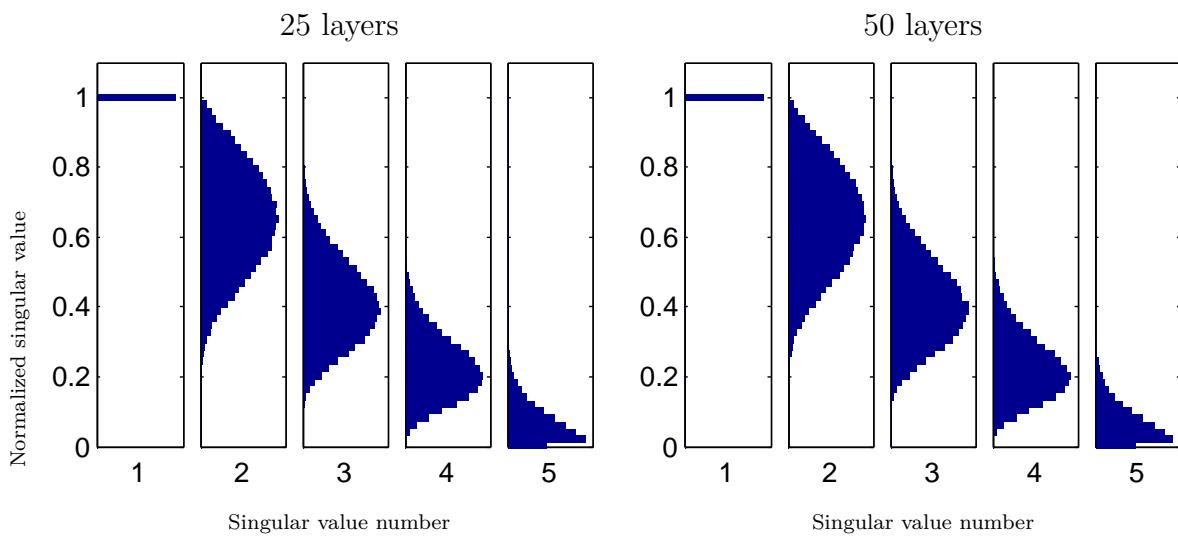


Figure 1.11 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors, 25 layers deep (*Left*) and 50 layers deep (*Right*). Compared to the standard architecture, the singular values are more likely to remain the same size as one another, meaning that the model outputs are more often sensitive to several directions of variation in the input.

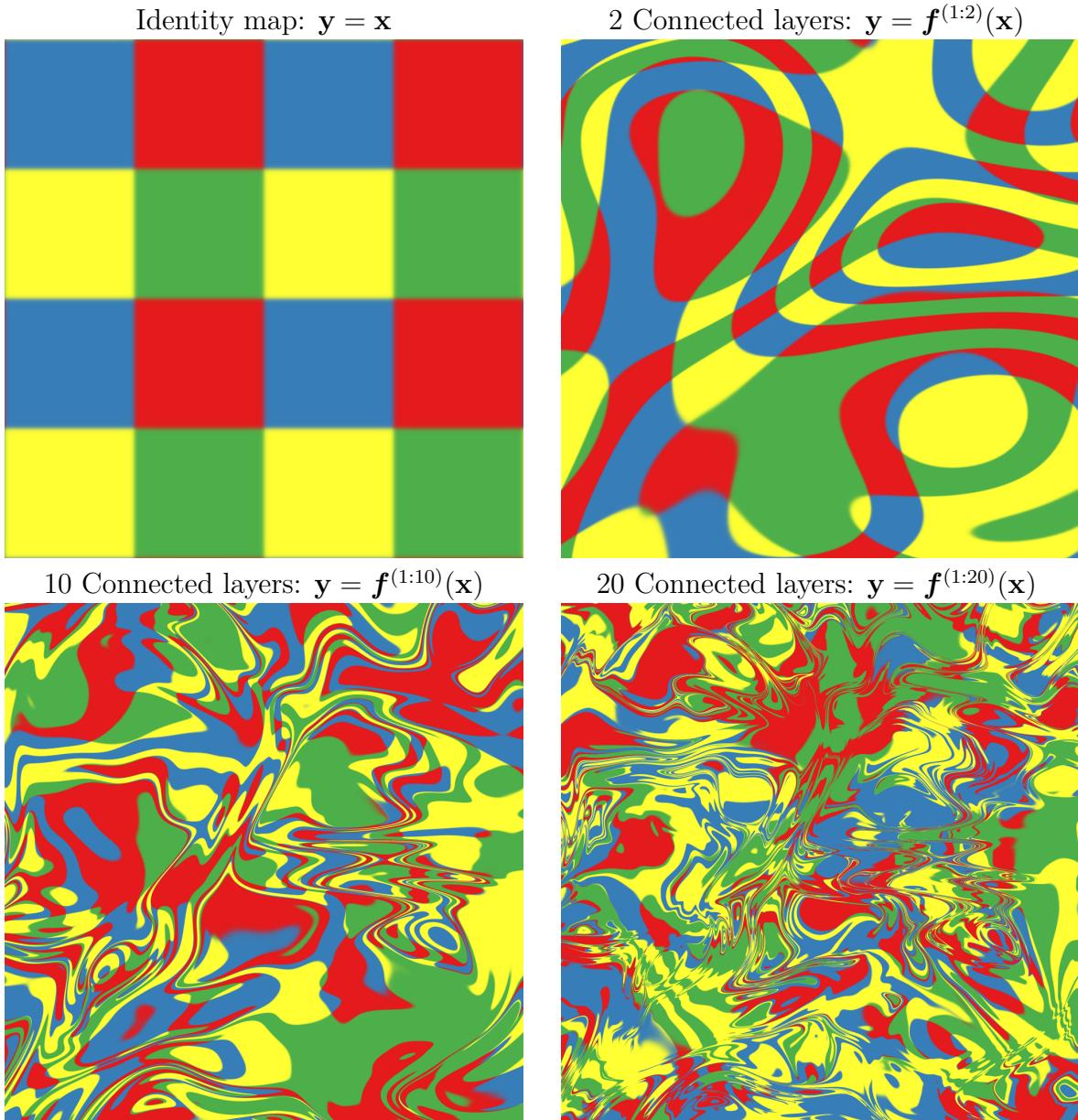


Figure 1.12 The feature map implied by a function  $\mathbf{f}$  drawn from a deep GP prior with each layer connected to the input  $\mathbf{x}$ , visualized at various depths. Compare to the map shown in figure 1.7. In the mapping shown here there are sometimes two directions that one can move locally in  $\mathbf{x}$  to in order to change the value of  $\mathbf{f}(\mathbf{x})$ . This means that the input-connected prior puts mass on a greater variety of types of representations, some of which depend on all aspects of the input.

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.

## 1.5 Deep kernels

Bengio et al. (2006) showed that kernel machines have limited generalization ability when they use “local” kernels such as the squared-exp. However, as we saw in sections 1.7 to 1.7, structured, non-local kernels can allow extrapolation. Another way to build non-local kernels is by composing fixed feature maps, creating *deep kernels*. This section builds on the work of Cho and Saul (2009), who derived several kinds of deep kernels by applying multiple layers of feature mappings.

To return to an example given in ??, periodic kernels can be viewed as a 2-layer-deep kernel, in which the first layer maps  $x \rightarrow [\sin(x), \cos(x)]$ , and the second layer maps through basis functions corresponding to the SE kernel.

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

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{h}_a(\mathbf{x})^\top \mathbf{h}_a(\mathbf{x}') \quad (1.18)$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{h}_b(\mathbf{x})^\top \mathbf{h}_b(\mathbf{x}') \quad (1.19)$$

$$(k_b \circ k_a)(\mathbf{x}, \mathbf{x}') = k_b(\mathbf{h}_a(\mathbf{x}), \mathbf{h}_a(\mathbf{x}')) = [\mathbf{h}_b(\mathbf{h}_a(\mathbf{x}))]^\top \mathbf{h}_b(\mathbf{h}_a(\mathbf{x}')) \quad (1.20)$$

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  $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}')$ , then

$$(\text{SE} \circ k)(\mathbf{x}, \mathbf{x}') = k_{\text{SE}}(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')) \quad (1.21)$$

$$= \exp\left(-\frac{1}{2}\|\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')\|_2^2\right) \quad (1.22)$$

$$= \exp\left(-\frac{1}{2}[\mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\top \mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\top \mathbf{h}(\mathbf{x}')]\right) \quad (1.23)$$

$$= \exp\left(-\frac{1}{2}[k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{x}') + k(\mathbf{x}', \mathbf{x}')]\right). \quad (1.24)$$

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

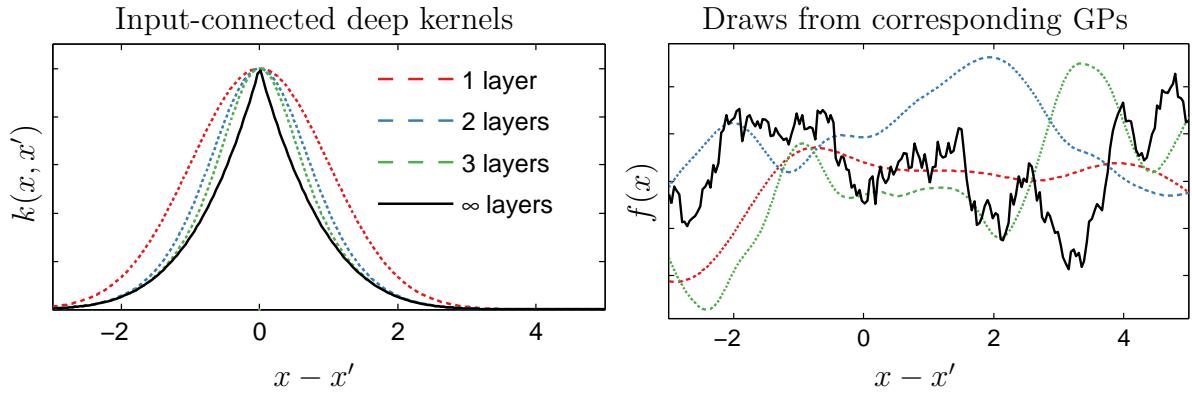


Figure 1.13 *Left*: Input-connected deep kernels of different depths. By connecting the input  $\mathbf{x}$  to each layer, the kernel can still depend on its input even after arbitrarily many layers of composition. *Right*: Draws from GPs with deep input-connected kernels.

### 1.5.1 Infinitely deep kernels

What happens when we repeatedly compose 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  $(\text{SE} \circ \text{SE} \circ \text{SE} \circ \dots \circ \text{SE})(\mathbf{x}, \mathbf{x}') = 1$  for all pairs of inputs. A constant covariance corresponds to a prior on constant functions  $f(\mathbf{x}) = c$ .

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:

$$\begin{aligned} k_C^{(1:\ell+1)}(\mathbf{x}, \mathbf{x}') &= \exp \left( -\frac{1}{2} \left\| \begin{bmatrix} \mathbf{h}^{(1:\ell)}(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} - \begin{bmatrix} \mathbf{h}^{(1:\ell)}(\mathbf{x}') \\ \mathbf{x}' \end{bmatrix} \right\|_2^2 \right) \\ &= \exp \left( -\frac{1}{2} [k_C^{(1:\ell)}(\mathbf{x}, \mathbf{x}) - 2k_C^{(1:\ell)}(\mathbf{x}, \mathbf{x}') + k_C^{(1:\ell)}(\mathbf{x}', \mathbf{x}') - \|\mathbf{x} - \mathbf{x}'\|_2^2] \right) \end{aligned} \quad (1.25)$$

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

$$k_C^{(1:\infty)}(\mathbf{x}, \mathbf{x}') - \log(k_C^{(1:\infty)}(\mathbf{x}, \mathbf{x}')) = 1 + \frac{1}{2} \|\mathbf{x} - \mathbf{x}'\|_2^2. \quad (1.26)$$

The solution to this recurrence has no closed form, but for one input dimension, has a similar shape to the Ornstein-Uhlenbeck covariance  $\text{OU}(x, x') = \exp(-|x - x'|)$  but with lighter tails. Samples from a GP prior with this kernel are not differentiable, and are locally fractal. Figure 1.13 shows this kernel at different depths, as well as samples from

the resulting GP priors.

We 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.

### 1.5.2 When are deep kernels useful models?

Kernels correspond to fixed feature maps, and so kernel learning is an example of implicit representation learning. As we saw in sections 1.7 and 1.7, kernels can capture rich structure and can enable many types of generalization. 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.

## 1.6 Related work

### Deep Gaussian processes

[Neal \(1995\)](#), 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 chapter can be seen as a more detailed investigation, and vindication, of these claims.

The first instance of deep GPs being used in practice was ([Lawrence and Moore, 2007](#)), 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 [Damianou and Lawrence \(2013\)](#), 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, [Wang and Neal \(2012\)](#) investigated single-layer GP regression models that had additional latent inputs.

## Nonparametric neural networks

[Adams et al. \(2010\)](#) proposed a prior on arbitrarily deep Bayesian networks with an unknown and unbounded number of parametric hidden units in each layer. Their architecture has connections only between adjacent layers, and so may be expected to have similar pathologies to those of deep GPs as the number of layers increases.

[Wilson et al. \(2012\)](#) 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:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}(\mathbf{x})\mathbf{f}(\mathbf{x}) \quad \text{where each } f_d, W_{d,j} \stackrel{\text{iid}}{\sim} \mathcal{GP}(\mathbf{0}, \text{SE} + \text{WN}). \quad (1.27)$$

We can easily define a “deep” Gaussian process regression network:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}^{(3)}(\mathbf{x})\mathbf{W}^{(2)}(\mathbf{x})\mathbf{W}^{(1)}(\mathbf{x})\mathbf{f}(\mathbf{x}) \quad (1.28)$$

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 (1.15)), and so might be amenable to a similar analysis to that of section 1.2.

## Information-preserving architectures

Deep density networks ([Rippel and Adams, 2013](#)) 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 1.3.

## Recurrent networks

Bengio et al. (1994) and Pascanu et al. (2012) analyze a related problem with gradient-based learning in recurrent networks, the “exploding-gradients” problem. They note 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.

Hochreiter and Schmidhuber (1997) addressed the exploding-gradients problem by including hidden units designed to have stable gradients.

## Deep kernels

The first systematic examination of deep kernels was done by Cho and Saul (2009), 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.

Hermans and Schrauwen (2012) 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 Neal (1995, chapter 2).

## Analyses of deep learning

Montavon et al. (2010) 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.

The experiments of Saxe et al. (2011) suggested that most of the performance of convolutional neural networks could be attributed to the architecture alone. Later, Saxe et al. (2013) looked at the dynamics of gradient-based training methods in deep *linear* networks as a tractable approximation to standard deep (nonlinear) neural networks.

## 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 1.7 and 1.12 using neural nets instead of GPs at each layer.

## 1.7 Conclusions

This chapter demonstrated that well-defined priors allow explicit examination of the assumptions being made about functions being modeled by different neural network architectures. As an example of the sort of analysis made possible this way, we attempted to gain insight into the properties of deep neural networks by characterizing the sorts of functions likely to be obtained under different choices of priors on compositions of functions.

First, we identified deep Gaussian processes as an easy-to-analyze model corresponding to multi-layer preceptrons with nonparametric activation functions. We then 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. Finally, we 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 features.

Much recent work on deep networks has focused on weight initialization ([Martens, 2010](#)), regularization ([Lee et al., 2007](#)) and network architecture ([Gens and Domingos, 2013](#)). If we can identify priors that give our models desirable properties, these in turn may suggest regularization, initialization, and architecture choices that also provide such properties.

Existing neural network practice also requires expensive tuning of model hyperparameters such as the number of layers, the size of each layer, and regularization penalties by cross-validation. One advantage of deep GPs is that the approximate marginal likelihood allows a principled method for automatically determining such model choices.

# 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. (page 20)
- 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. (page 21)
- 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. (page 17)
- Youngmin Cho and Lawrence K. Saul. Kernel methods for deep learning. In *Advances in Neural Information Processing Systems*, pages 342–350, 2009. (pages 17 and 21)
- Andreas Damianou and Neil D. Lawrence. Deep Gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013. (pages 1 and 19)
- David Duvenaud, Oren Rippel, Ryan P. Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. In *17th International Conference on Artificial Intelligence and Statistics*, Reykjavik, Iceland, April 2014. (page 2)
- Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine learning*, 2013. (page 22)
- James Hensman, Andreas Damianou, and Neil D. Lawrence. Deep Gaussian processes for large datasets. In *Artificial Intelligence and Statistics Late-breaking Posters*, 2014. (page 1)

- Michiel Hermans and Benjamin Schrauwen. Recurrent kernel machines: Computing with infinite echo state networks. *Neural Computation*, 24(1):104–133, 2012. (page 21)
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (page 21)
- 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. (page 19)
- 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. (page 22)
- James Martens. Deep learning via Hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning*, pages 735–742, 2010. (page 22)
- 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. (page 14)
- 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. (page 21)
- Radford M. Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995. (pages 3, 11, 19, and 21)
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *arXiv preprint arXiv:1211.5063*, 2012. (page 21)
- 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. (page 9)
- 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. (page 10)

- Oren Rippel and Ryan P. Adams. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013. (page 20)
- Frank Rosenblatt. Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. *Brain Mechanisms*, pages 555–559, 1962. (page 2)
- 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. (page 21)
- Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Dynamics of learning in deep linear neural networks. In *NIPS Workshop on Deep Learning*, 2013. (page 21)
- 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. (page 8)
- Chunyi Wang and Radford M. Neal. Gaussian process regression with heteroscedastic or non-gaussian residuals. *arXiv preprint arXiv:1212.6246*, 2012. (page 20)
- 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. (page 20)