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*. Deep GPs are an attractive model class to study for several reasons. **?** showed that the probabilistic nature of deep GPs gaurds against overfitting, and allows us to use the marginal likelihood to automatically tune the model architecture without the need for cross-validation. **?** showed that stochastic variational inference is possible in deep GPs. Together, these results suggest that deep GPs are a promising alternative to neural nets. For the analysis in this chapter, deep GPs are attractive 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 **?**.

1.1 Definition and relation to neural networks

In this chapter, we'll define a deep GP as any prior on functions constructed by composing draws from GP priors. An example of a deep GP is a composition of vector-valued functions, where each function in each layer is drawn independently from a GP prior:

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

A multilayer neural network also implements a composition of vector-valued functions, one per layer. The rest of this section shows the precise relationship between Gaussian processes and neural nets, as well as two equivalent ways of constructing neural networks which give rise to deep GPs.

1.1.1 Single-layer models

First, we will relate neural networks with one hidden layer to Gaussian processes, using the standard neural network architecture known as the multi-layer perceptrion (MLP). In the typical definition of an MLP, the hidden units of the first layer are defined as:

$$\mathbf{h}^{(1)}(\mathbf{x}) = \sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right) \tag{1.2}$$

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

$$f(\mathbf{x}) = \mathbf{V}^{(1)}\sigma\left(\mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}\right) = \mathbf{V}^{(1)}\mathbf{h}^{(1)}(\mathbf{x}) \tag{1.3}$$

where $\mathbf{V}^{(1)}$ is another weight matrix.

?, chapter 2 showed that GPs can be viewed as a prior on neural networks with infinitely many hidden units and unknown weights. More precisely, for any model of the

form

$$f(\mathbf{x}) = \frac{1}{K}\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{h}(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K}\alpha_i h_i(\mathbf{x}), \tag{1.4}$$

with fixed[1] features $[h_1(\mathbf{x}), \ldots, h_K(\mathbf{x})]^{\mathsf{T}} = \mathbf{h}(\mathbf{x})$ and i.i.d. α'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 $f(\mathbf{x}), f(\mathbf{x}')$ have a joint distribution approaching a Gaussian:

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

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 $\boldsymbol{\alpha}$ 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: $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^{\mathsf{T}}\mathbf{h}(\mathbf{x}')$. 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 the hidden units of the MLP.

1.1.2 Multiple hidden layers

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

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

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 would become

$$f(\mathbf{x}) = \frac{1}{K}\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right) . \tag{1.7}$$

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

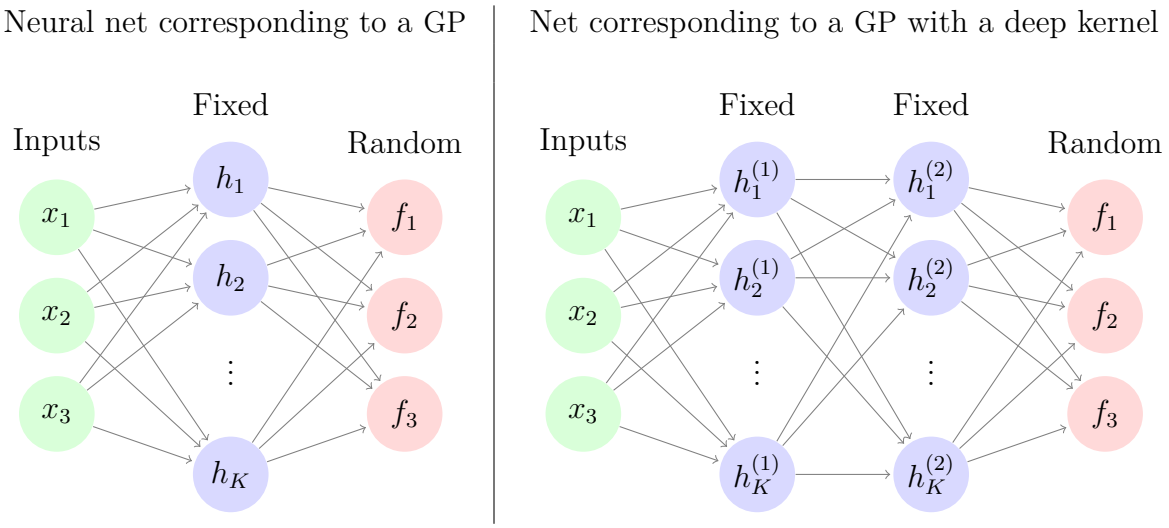


Figure 1.1 *Left:* GPs can be understood as a one-hidden-layer MLP with infinitely many fixed hidden units having unknown weights. *Right:* Multiple layers of fixed hidden units gives rise to a GP with a deep kernel, but not a deep GP.

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

$$k(\mathbf{x}, \mathbf{x}') = \left(\mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}))\right)^{\mathsf{T}} \mathbf{h}^{(2)}(\mathbf{h}^{(1)}(\mathbf{x}')) \tag{1.8}$$

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

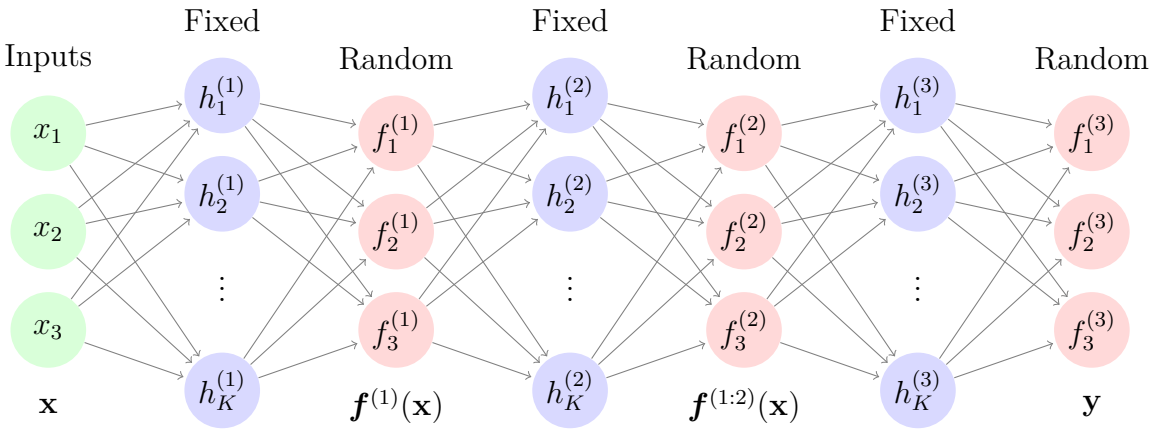
1.1.3 Two network architectures equivalent to deep GPs

There are two equivalent neural net architectures which correspond to deep GPs: one with fixed non-linearities, and another with GP-distributed nonlinearties.

To construct a neural network with fixed nonlinearities that corresponds to a deep GP, 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 $\boldsymbol{f}^{(\ell)}(\mathbf{x})$ in between each 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 informa-

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



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

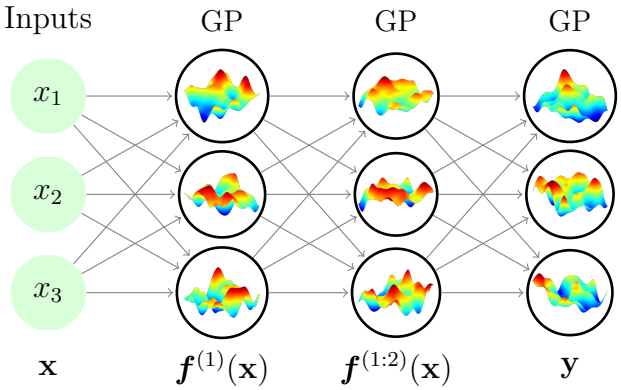


Figure 1.2 Two equivalent views of deep GPs as neural networks. *Top:* A neural network whose every second layer is a weighted sum of an infinite number of fixed hidden units, where the weights are initially unknown. *Bottom:* A neural network with a finite number of hidden units, each with a different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

tion bottlenecks. To see this, we can simply substitute equation (1.3) into equation (1.6) to get

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

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 $\mathbf{W}^{(\ell)}\mathbf{V}^{(\ell-1)}$.

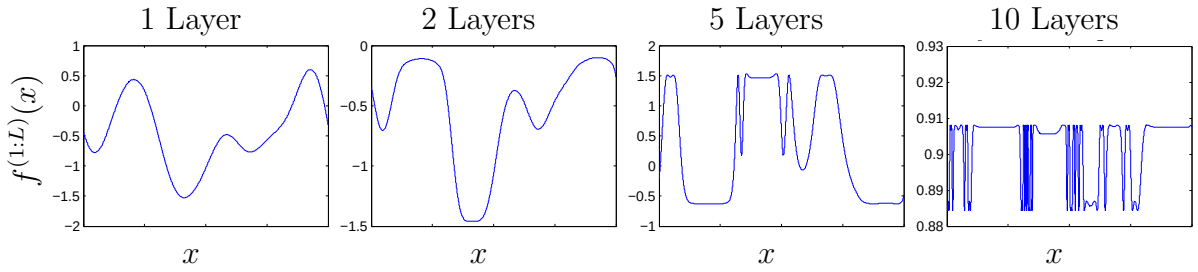


Figure 1.3 A one-dimensional draw from a deep GP prior, shown at different depths. The x-axis is the same for all plots. After a few layers, the functions begin to be either nearly flat, or highly varying, everywhere. This is a consequence of the distribution on derivatives becoming heavy-tailed. The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

The second, more direct way to construct a network architecture corresponding to a deep GP is to integrate out all $\mathbf{V}^{(\ell)}$, 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 1.1(bottom) shows this architecture.

1.2 Characterizing deep Gaussian process priors

In this section, we develop several theoretical results that explore the behavior of deep GPs as a function of their depth. This will allow us in section 1.3 to formally identify a pathology that emerges in very deep networks.

Specifically, we will 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.

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:

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

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 $\mathcal{N}(0, \sigma^2/w^2)$. 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. 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 log of the magnitude of the derivatives has moments

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

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 log-normal as L grows:

$$\log\left|\frac{\partial f^{(1:L)}(x)}{\partial x}\right| = \sum_{\ell=1}^{L} \log\left|\frac{\partial f^{(\ell)}(x)}{\partial x}\right| \overset{L\to\infty}{\sim} \mathcal{N}\Big(Lm_{\log}, L^2 v_{\log}\Big) \tag{1.12}$$

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, at varying depths. 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 a deep GP prior.

Lemma 1.2.1. *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 **?**:

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

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. □

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

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

Lemma 1.2.2. *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 $f_d(\mathbf{x})$ are independent, 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. □

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

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

By lemma 1.2.2, 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. □

Theorem 1.2.3 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

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

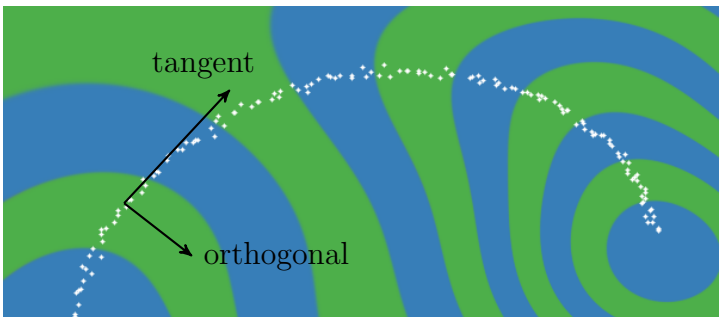


Figure 1.4 Representing a 1-D data manifold. Colors correspond to the computed representation as a function of the input space. The representation (blue & green) varies in directions tangent to the data manifold (white), preserving information for later layers. The representation changes little in directions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

As in **?**, 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 which the representation varies. **?** analyzed the Jacobian at location of the training points, but because the priors

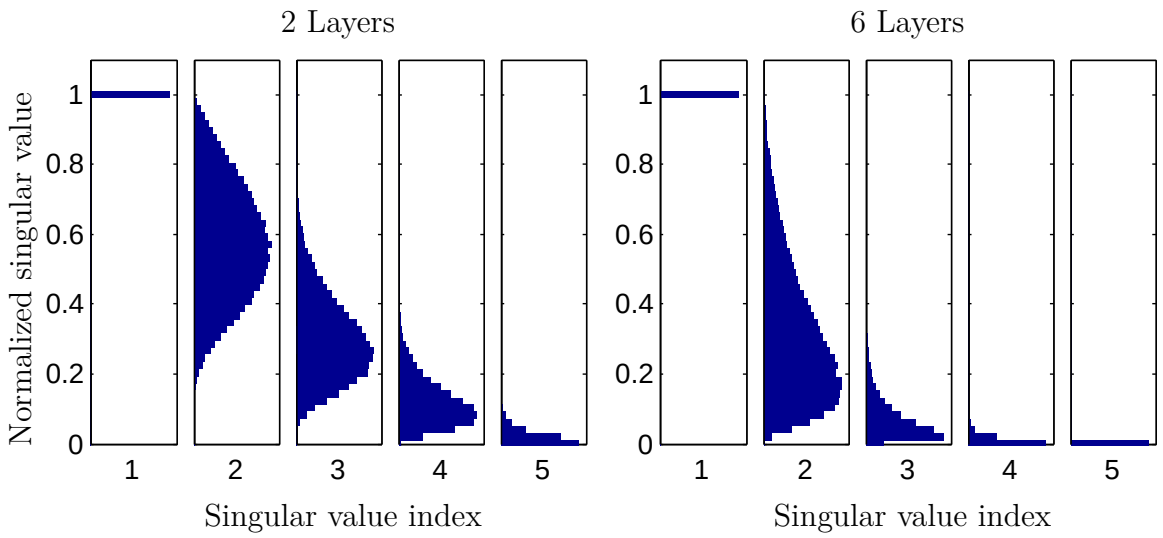


Figure 1.5 The distribution of normalized singular values of the Jacobian of a function drawn from a 5-dimensional deep GP prior 25 layers deep (*Left*) and 50 layers deep (*Right*). As nets get deeper, the largest singular value tends to become much larger than the others. This implies that with high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

we are examining are stationary, the distribution of the Jacobian is identical everywhere. Figure 1.5 shows the singular value spectrum for 5-dimensional deep GPs of different depths. As the net gets deeper, the largest singular value dominates, implying there is usually only one effective degree of freedom in representation 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*, again implying that such models are unsuitable to model 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. 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

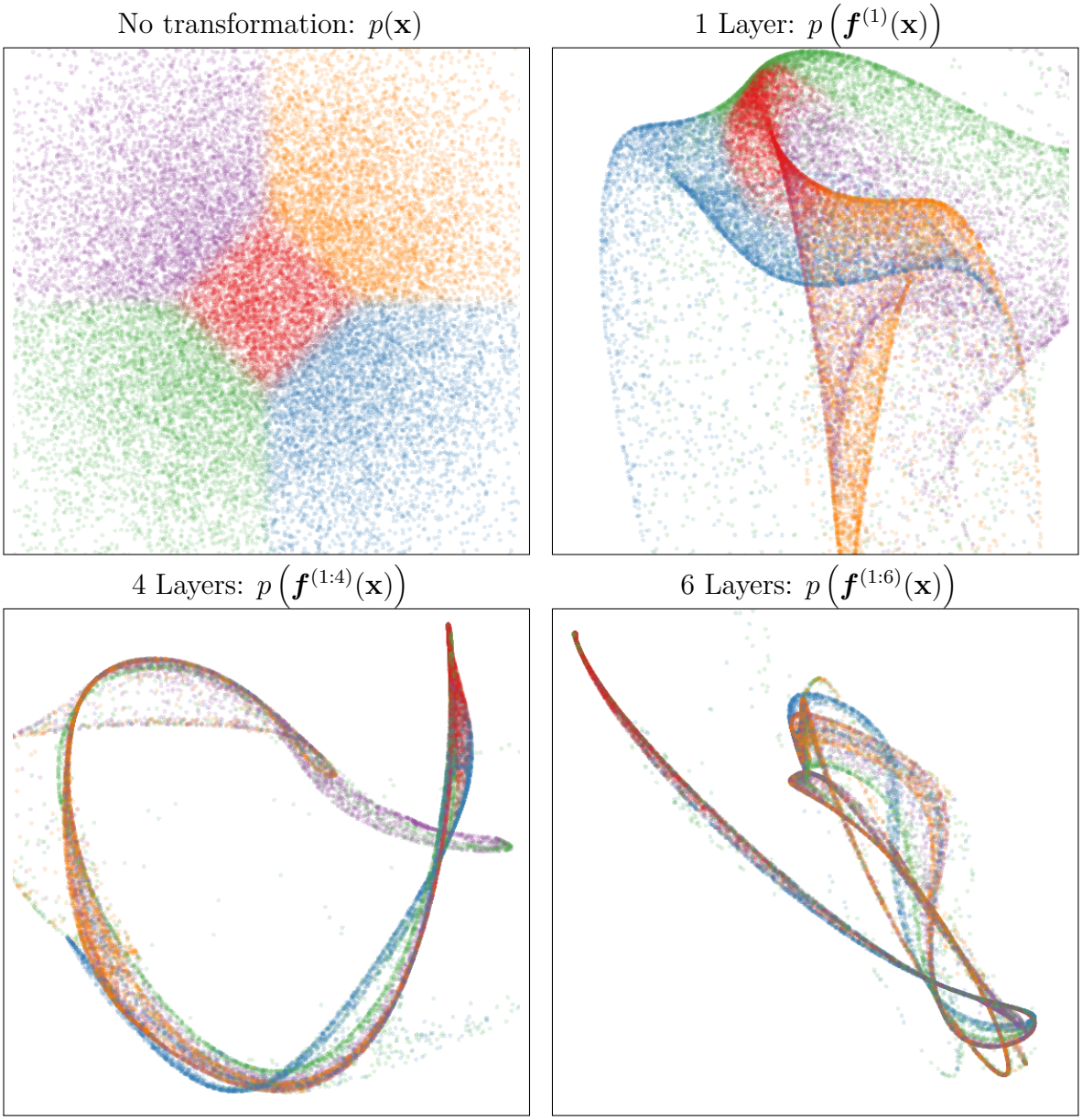


Figure 1.6 Points warped by a function drawn from a deep GP prior. *Top left:* Points drawn from a 2-dimensional Gaussian distribution, color-coded by their location. *Subsequent panels:* Those same points, successively warped by functions drawn from a GP prior. As the number of layers increases, the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

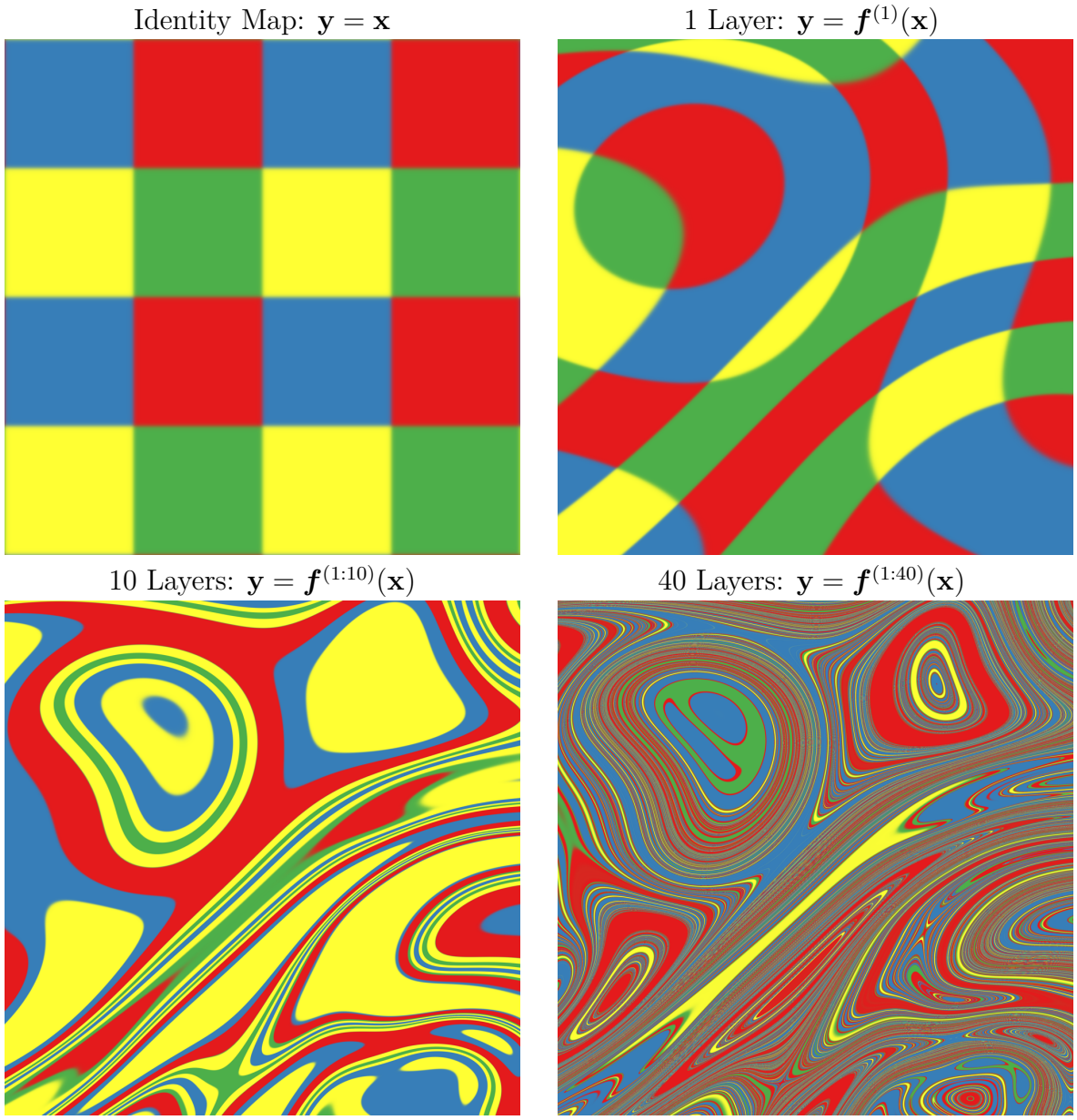


Figure 1.7 A vizualization of the feature map implied by a draw from a deep GP. Colors correspond to the location $\mathbf{y} = \boldsymbol{f}(\mathbf{x})$ that each point is mapped to after being warped by a deep GP. The number of directions in which the color changes rapidly corresponds to the number 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 $\boldsymbol{f}$ 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, the roughly circular shape remains in the top-left corner even after 40 independent warpings.

latent 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 **?**, 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 $f_C(\mathbf{x})$. Figure 1.8 shows a graphical representation of the two connectivity architectures. Similar connections between non-adjacent layers can also be found the primate visual cortex (**?**). Formally, this functional dependence can be written as

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

Visualizations of the resulting prior on functions are shown in figures 1.9, 1.10 and 1.12.

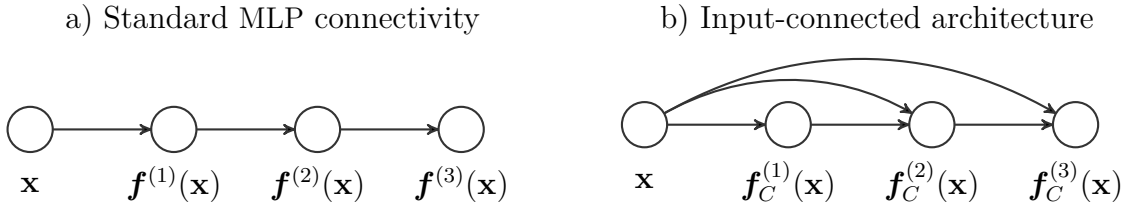


Figure 1.8 Two different architectures for deep neural networks. *Left:* The standard architecture connects each layer's outputs to the next layer's inputs. *Right:* The input-connected architecture also connects the original input $\mathbf{x}$ to each layer.

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

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

which, in the case of a deep GP, 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. As explained in section 1.1, GPs are

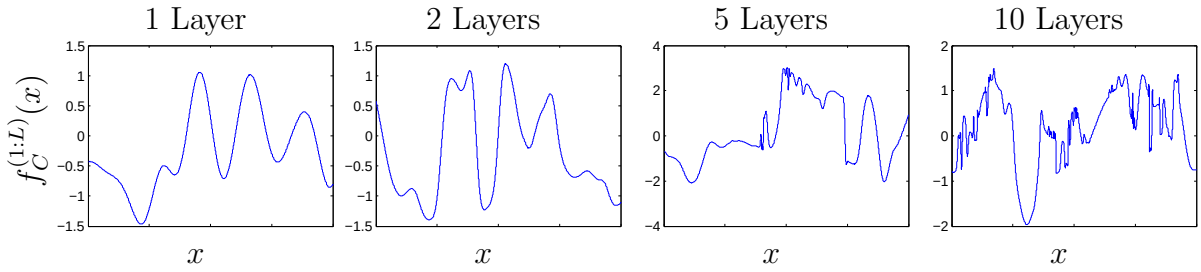


Figure 1.9 A draw from a 1D deep GP prior with each layer connected to the input. The x-axis is the same for all plots. Even after many layers, the functions remain smooth in some regions, while varying rapidly in other regions. Compare to standard-connectivity deep GP draws shown in figure 1.3.

examples of a particular limit of Bayesian neural networks. The figures and analysis done in this section could have also been done using Bayesian neural networks with finite numbers of nodes, but would have been 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

? 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 and 1.7, structured kernels can be constructed through addition and multiplication which allow non-local extrapolation.

We can also build non-local kernels by composing fixed feature maps. To return to an example given in **??**, periodic kernels can be viewed as a 2-layer-deep kernel, in which the first layer maps $x \to [\sin(x), \cos(x)]$, and the second layer maps through basis functions corresponding to the SE kernel.

We can construct other useful kernels by composing fixed feature maps several times, creating deep kernels. This section builds on the work of **?**, who derived several kinds of deep kernels by applying multiple layers of feature mappings.

In principle, we can compose the feature mapping of any two kernels to get a new

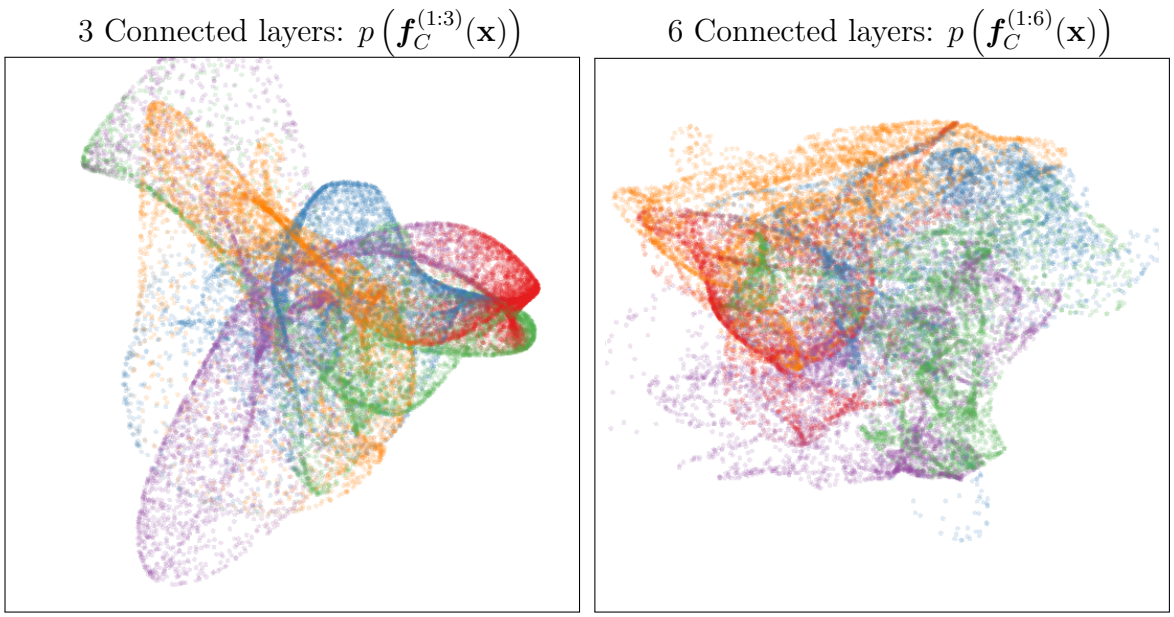


Figure 1.10 Points warped by a draw from a deep GP, with each layer connected to the input $\mathbf{x}$. As depth increases, the density becomes more complex without concentrating everywhere along one-dimensional filaments.

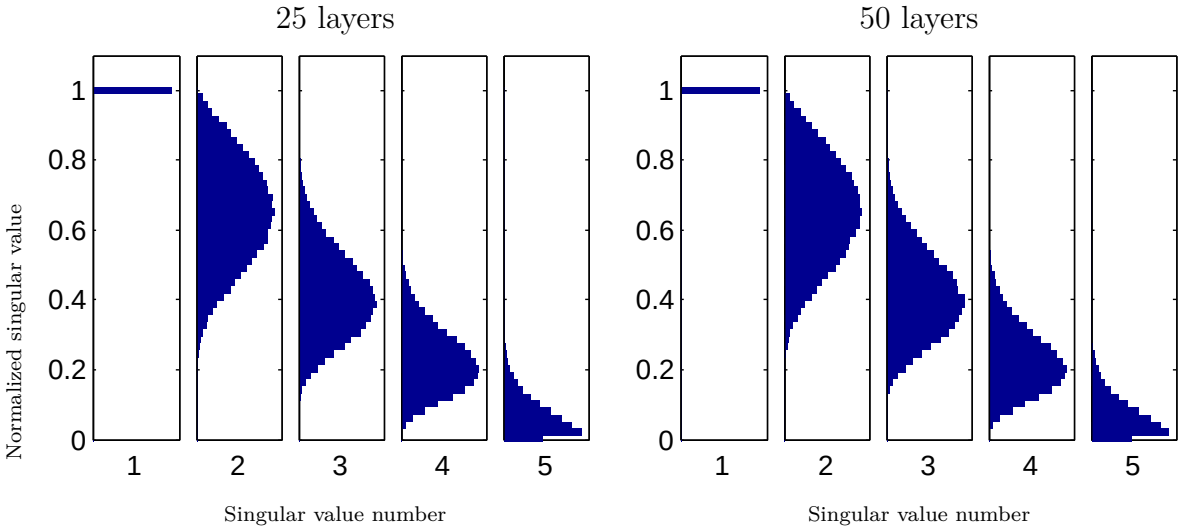


Figure 1.11 The distribution of singular values drawn from 5-dimensional input-connected deep GP priors, 25 and 50 layers deep. The singular values remain roughly the same scale as one another, meaning that the model outputs are sensitive to most directions of variation in the input.

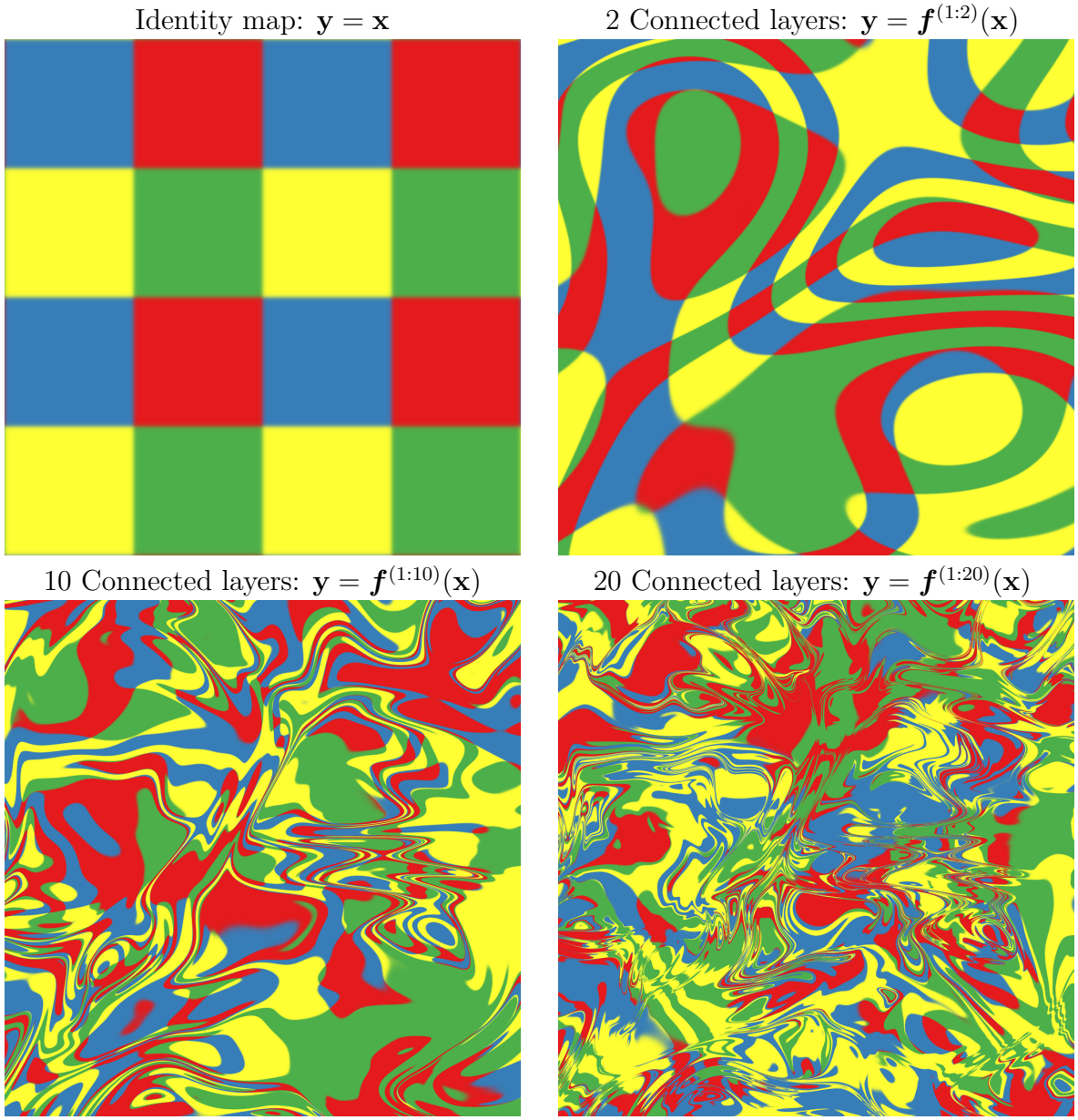


Figure 1.12 Feature mapping of a deep GP with each layer connected to the input $\mathbf{x}$. Compare to the mappings shown in figure 1.7. Just as the densities in figure 1.10 remained locally two-dimensional even after many warpings, in the mapping shown here there are sometimes two directions that one can move locally in $\mathbf{x}$ to in order to change the values of $\boldsymbol{f}(\mathbf{x})$. 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.

one:

$$k_a(\mathbf{x}, \mathbf{x}') = \mathbf{h}_a(\mathbf{x})^\mathsf{T}\mathbf{h}_a(\mathbf{x}') \tag{1.18}$$

$$k_b(\mathbf{x}, \mathbf{x}') = \mathbf{h}_b(\mathbf{x})^\mathsf{T}\mathbf{h}_b(\mathbf{x}') \tag{1.19}$$

$$(k_b \circ k_a)(\mathbf{x}, \mathbf{x}') = k_b(\mathbf{h}_a(\mathbf{x}), \mathbf{h}_a(\mathbf{x}')) = [\mathbf{h}_b(\mathbf{h}_a(\mathbf{x}))]^\mathsf{T} \mathbf{h}_b(\mathbf{h}_a(\mathbf{x}')) \tag{1.20}$$

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

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

$$\begin{aligned}(\mathrm{SE} \circ k)(\mathbf{x}, \mathbf{x}')(\mathbf{x}, \mathbf{x}') &= k_{SE}(\mathbf{h}(\mathbf{x}), \mathbf{h}(\mathbf{x}')) = \\ &= \exp\left(-\frac{1}{2}||\mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}')||_2^2\right) \\ &= \exp\left(-\frac{1}{2}\left[\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}) - 2\mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}') + \mathbf{h}(\mathbf{x}')^\mathsf{T}\mathbf{h}(\mathbf{x}')\right]\right) \\ &= \exp\left(-\frac{1}{2}\Big[k(\mathbf{x}, \mathbf{x}) - 2k(\mathbf{x}, \mathbf{x}') + k(\mathbf{x}', \mathbf{x}')\Big]\right) .\end{aligned} \tag{1.21}$$

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

1.5.1 Infinitely deep kernels

What happens when we repeat this composition of feature maps many times, starting with the squared-exp kernel? If the output variance $k(\mathbf{x}, \mathbf{x}) = 1$, then in the infinite limit, this operation converges to $(\mathrm{SE} \circ \mathrm{SE} \circ \mathrm{SE} \circ \ldots \mathrm{SE})(\mathbf{x}, \mathbf{x}') = 1$ for all pairs of inputs, which corresponds to a prior on constant functions $f(\mathbf{x}) = c$.

As before, we can overcome this degeneracy by connecting the inputs $\mathbf{x}$ to each layer. To do so, we simply concatenate the feature vector at each layer, $\mathbf{h}^{(\ell)}(\mathbf{x})$, with the input vector $\mathbf{x}$:

$$\begin{aligned}k^{(\ell+1)}(\mathbf{x}, \mathbf{x}') &= \exp\left(-\frac{1}{2}\left|\left|\begin{bmatrix}\mathbf{h}^{(\ell)}(\mathbf{x}) \\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}\mathbf{h}^{(\ell)}(\mathbf{x}') \\ \mathbf{x}'\end{bmatrix}\right|\right|_2^2\right) \\ &= \exp\left(-\frac{1}{2}\Big[k^{(\ell)}(\mathbf{x}, \mathbf{x}) - 2k^{(\ell)}(\mathbf{x}, \mathbf{x}') + k^{(\ell)}(\mathbf{x}', \mathbf{x}') - ||\mathbf{x} - \mathbf{x}'||_2^2\Big]\right)\end{aligned} \tag{1.22}$$

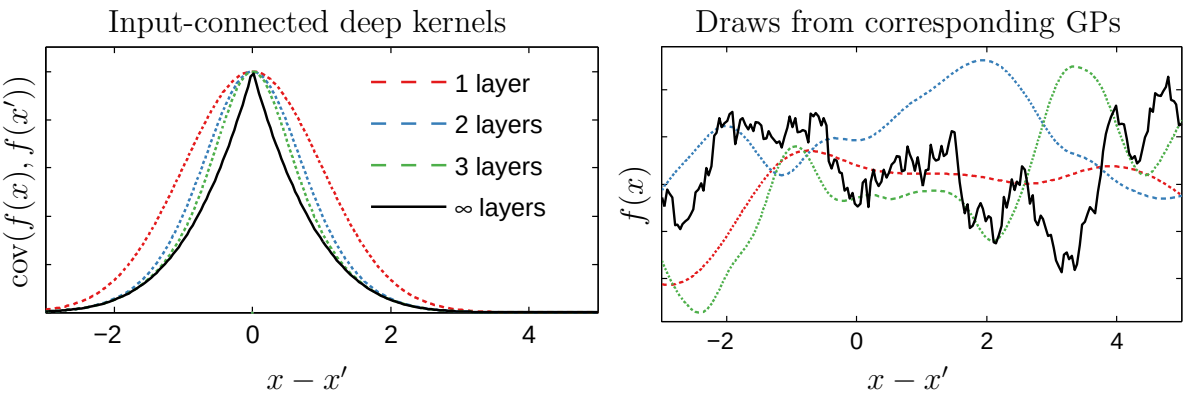


Figure 1.13 *Left:* Input-connected deep kernels of different depths. By connecting the inputs $\mathbf{x}$ to each layer, the kernel can still depend on its input even after arbitrarily many layers of computation. *Right:* Draws from GPs with deep input-connected kernels.

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

$$k^{(\infty)}(\mathbf{x}, \mathbf{x}') - \log\left(k^{(\infty)}(\mathbf{x}, \mathbf{x}')\right) = 1 + \frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2 \,. \tag{1.23}$$

The solution to this recurrence has no closed form, but 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 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 prior.

We can also consider two other 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, both these architectures converge to $k(\mathbf{x}, \mathbf{x}') = \delta(\mathbf{x}, \mathbf{x}')$, a 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. 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 processs

?, chapter 2 discussed the 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 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 also 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. Depending on kernel parameters, these latent variables may or may not depend on the inputs in 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.

Nonparametric neural networks

? proposed a prior on arbitrarily deep Bayesian networks with an unknown and unbounded number of hidden units in each layer. Their architecture has connections only between adjacent layers, and may also be expected to have similar pathologies to those of deep GPs as the number of layers increases.

Deep density networks (**?**) are constructed through a series of warpings of fixed output 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.

? 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:

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \qquad \text{with each } f_d, W_{d,j} \overset{\text{iid}}{\sim} \mathcal{GP}(\mathbf{0}, \text{SE} + \text{WN})\,. \tag{1.24}$$

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

$$\mathbf{y}(\mathbf{x}) = \mathbf{W}^{(3)}(\mathbf{x})\mathbf{W}^{(2)}(\mathbf{x})\mathbf{W}^{(1)}(\mathbf{x})\boldsymbol{f}(\mathbf{x}) \tag{1.25}$$

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.

Recurrent networks

? and **?** analyze a related problem with gradient-based learning in recurrent nets, 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.

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.

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.

The experiments of **?** suggested that most of the 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.

Source code

Source code to produce all figures is available at `github.com/duvenaud/deep-limits`. The source 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 showed that well-defined priors allow explicit examination of the assumptions being made about functions being modeled. 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 directions of variation but one. 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 (**?**), regularization (**?**) and network architecture (**?**). However, the interactions between these different design decisions can be complex and difficult to characterize. If we can identify classes of 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 details.