Visualizing Priors on Deep Functions




David Duvenaud, Oren Rippel, Ryan Adams, Zoubin Ghahramani

June 15, 2014

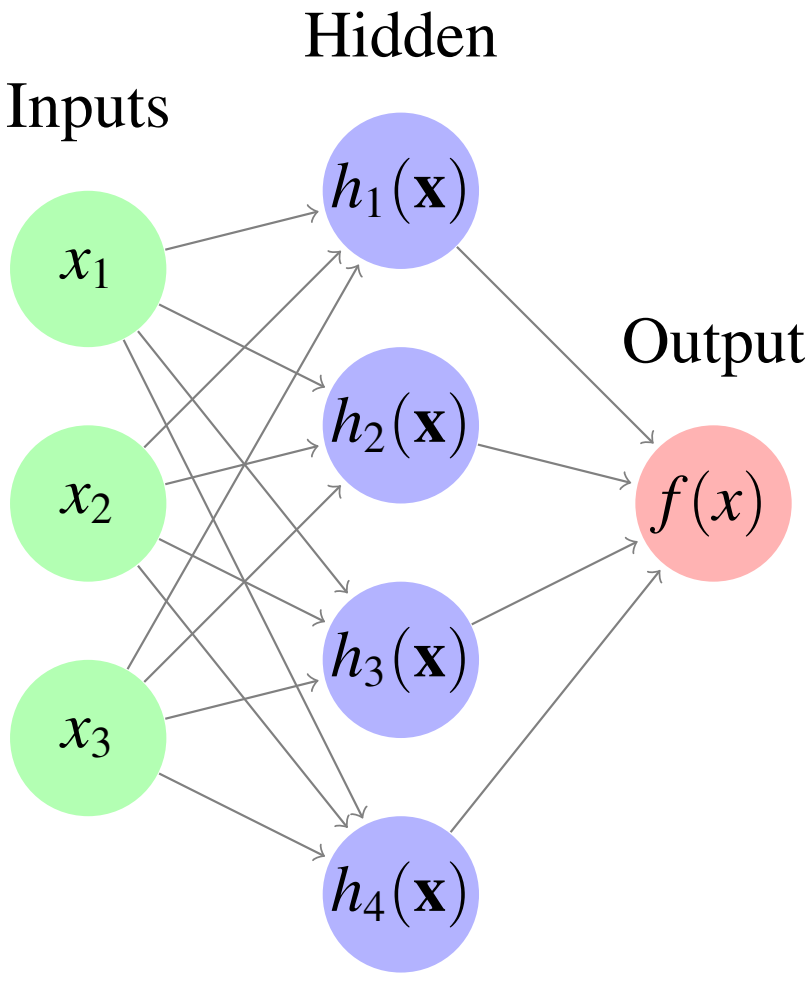
Motivation 1: Deep nets are hard to characterize

- deep learning experiments are annoying, too many fiddly parameters
- But - GPs are just neural nets, we can make them deep!

Motivation 2: Large nets are hard to regularize

- Neural nets are getting larger
- How to regularize billions of parameters?
- Closely related to constructing priors
- Priors are easy to analyze - just sample from the prior and look and what sorts of things you get!
- Can we suggest new regularization schemes or network architectures?

GPs as Neural Nets



A weighted sum of features,

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} w_i h_i(\mathbf{x})$$

with any weight distribution,

$$\mathbb{E}\left[w_i\right] = 0, \quad \mathbb{V}\left[w_i\right] = \sigma^2, \quad i.i.d.$$

by CLT, gives a GP as $K \to \infty$!

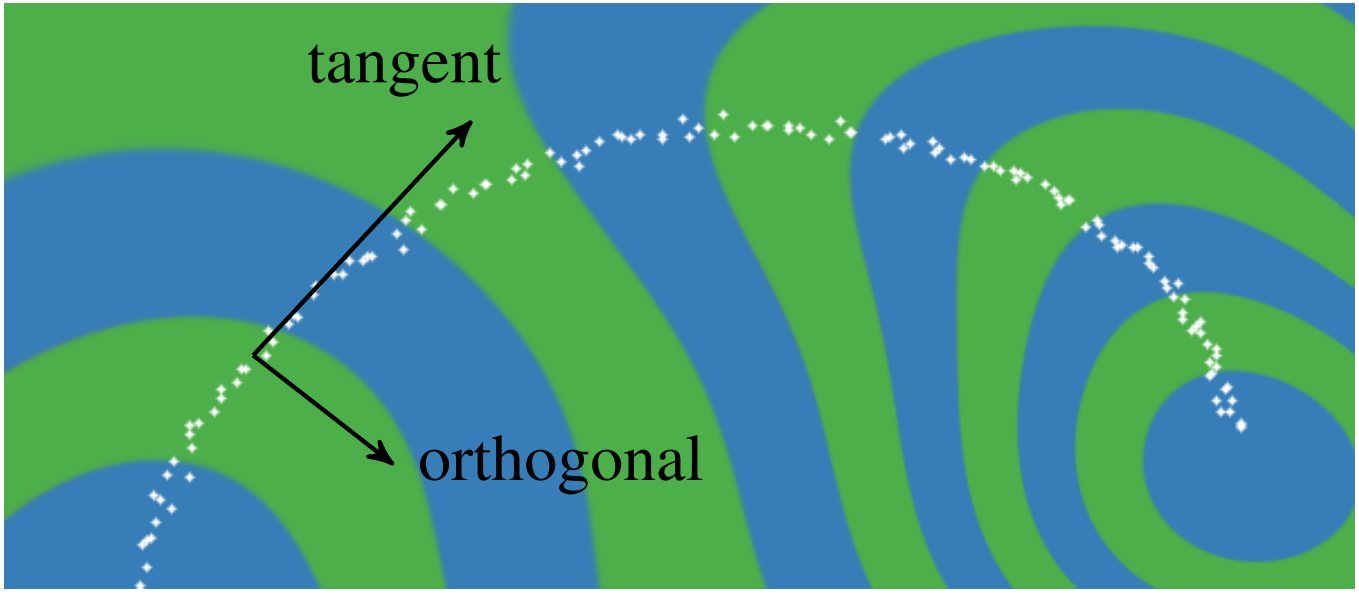
$$\mathrm{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x})h_i(\mathbf{x}')$$

Kernel learning as feature learning

- GPs have fixed features, integrate out feature weights.
- Mapping between kernels and features:
  $k(\mathbf{x}, \mathbf{x}') = \mathbf{h}(\mathbf{x})^\mathsf{T}\mathbf{h}(\mathbf{x}')$.
- Any PSD kernel can be written as inner product of features. (Mercer's Theorem)
- Kernel learning = feature learning

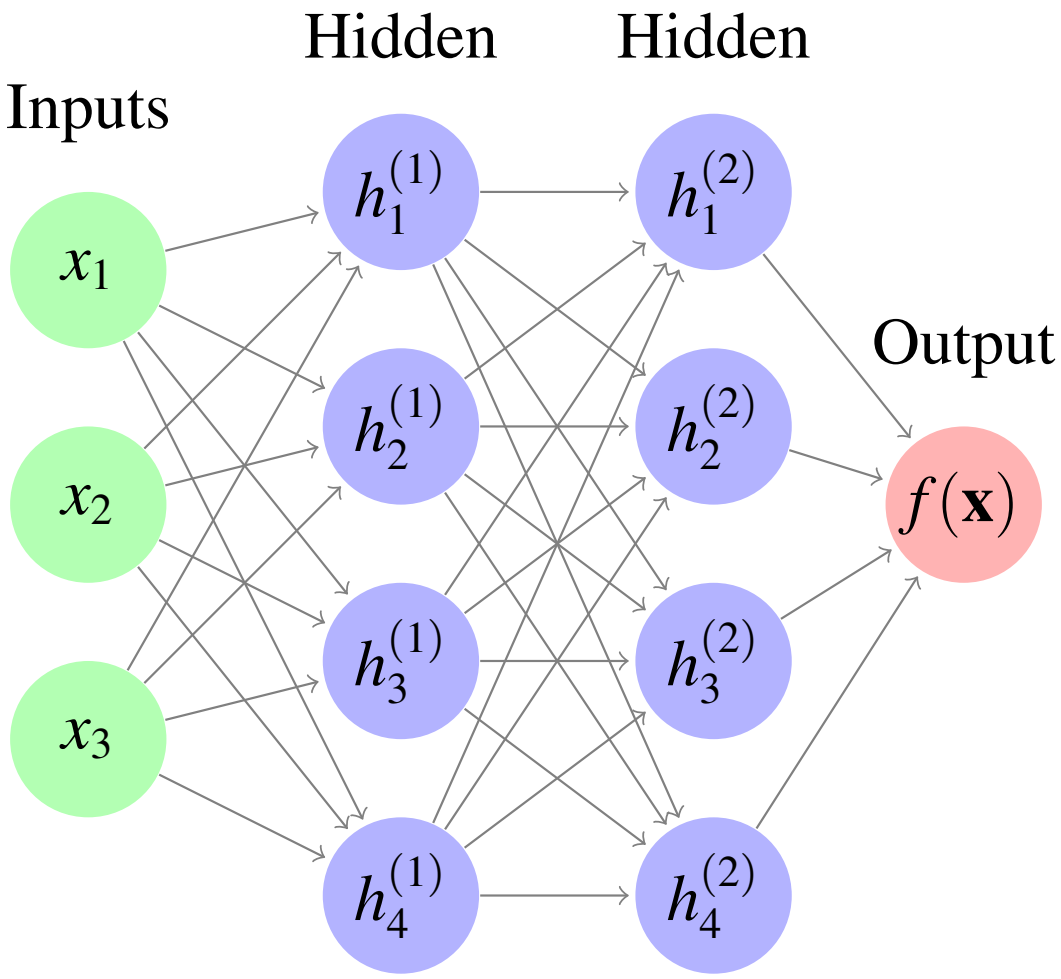
- What if we make the GP nueral network deep?

What makes a good representation?



- Rifai et. al. (2011): good representations of data manifolds are invariant in directions orthogonal to the data manifold.
- Conversely, a good representation changes in directions tangent to the data manifold, to preserve information.

Deep nets, deep kernels



Now our model is:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} w_i h_i^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right)$$
$$= \boldsymbol{w}^\top \mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right)$$

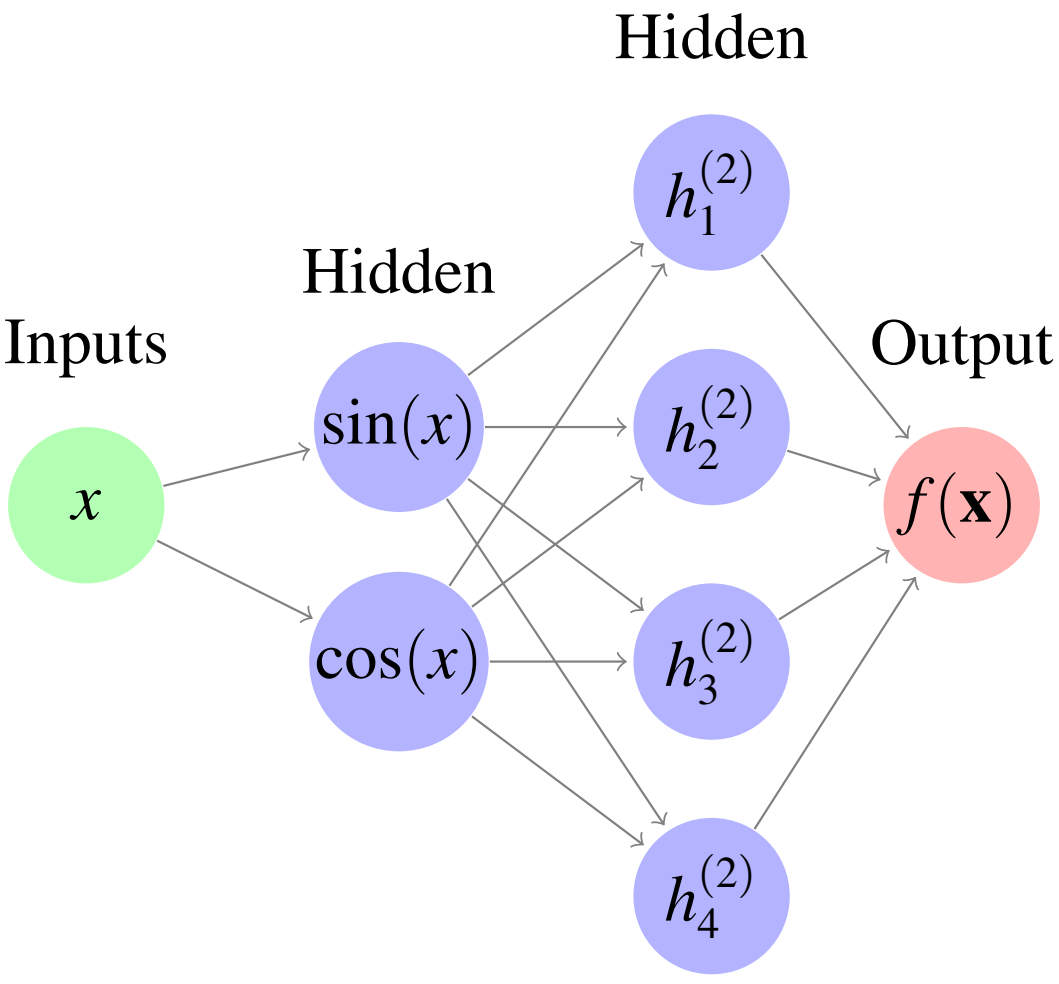
Instead of

$$k_1(\mathbf{x}, \mathbf{x}') = \mathbf{h}^{(1)}(\mathbf{x})^\top \mathbf{h}^{(1)}(\mathbf{x}'),$$

we have "deep kernel":

$$k_2(\mathbf{x}, \mathbf{x}')$$
$$= \left[\mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x})\right)\right]^\top \mathbf{h}^{(2)}\left(\mathbf{h}^{(1)}(\mathbf{x}')\right)$$

Example deep kernel: Periodic



Now our model is:

$$\mathbf{h}^1(x) = [\sin(x), \cos(x)]$$

we have "deep kernel":

$$k_2(\mathbf{x}, \mathbf{x}')$$

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

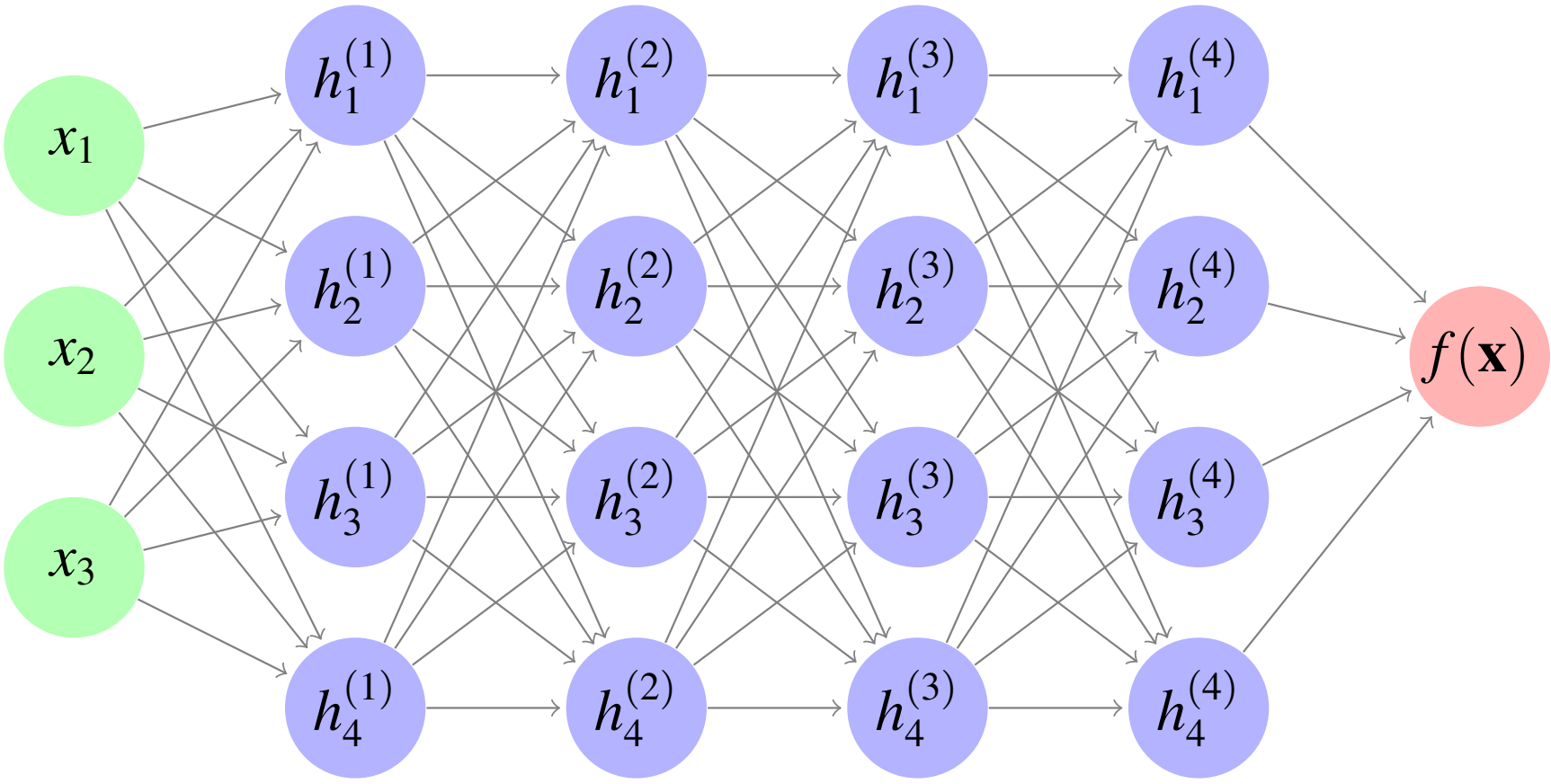
Deep Kernels

- (Cho, 2012) built kernels by composing feature mappings.
- Composing any kernel k_1 with a squared-exp kernel (SE):

$$
\begin{aligned}
& k_2(\mathbf{x}, \mathbf{x}') = \\
& = \left(\mathbf{h}^{SE}\left(\mathbf{h}^1(\mathbf{x})\right)\right)^{\mathsf{T}} \mathbf{h}^{SE}\left(\mathbf{h}^1(\mathbf{x}')\right) \\
& = \exp\left(-\frac{1}{2}||\mathbf{h}^1(\mathbf{x}) - \mathbf{h}^1(\mathbf{x}')||_2^2\right) \\
& = \exp\left(-\frac{1}{2}\left[\mathbf{h}^1(\mathbf{x})^{\mathsf{T}}\mathbf{h}^1(\mathbf{x}) - 2\mathbf{h}^1(\mathbf{x})^{\mathsf{T}}\mathbf{h}^1(\mathbf{x}') + \mathbf{h}^1(\mathbf{x}')^{\mathsf{T}}\mathbf{h}^1(\mathbf{x}')\right]\right) \\
& = \exp\left(-\frac{1}{2}\left[k_1(\mathbf{x}, \mathbf{x}) - 2k_1(\mathbf{x}, \mathbf{x}') + k_1(\mathbf{x}', \mathbf{x}')\right]\right)
\end{aligned}
$$

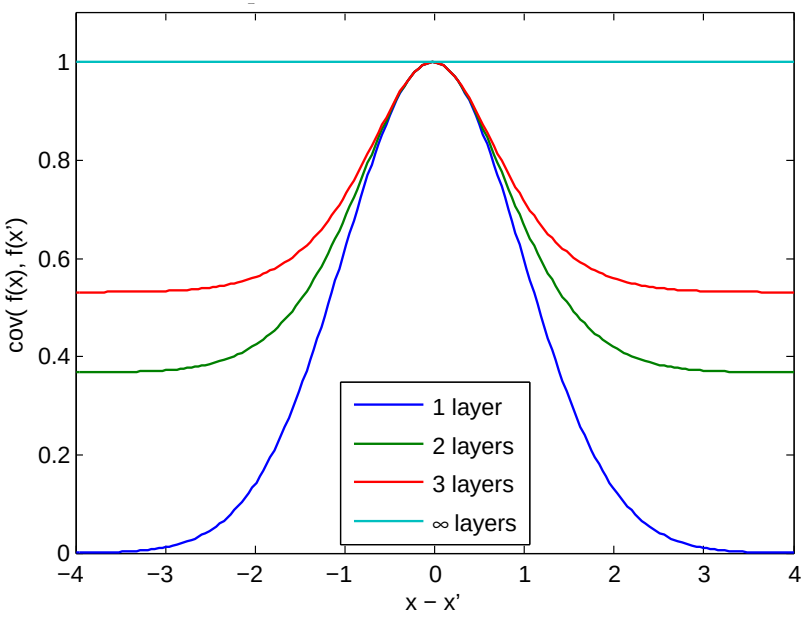
- A closed form. . . let's do it again!

We need to go deeper

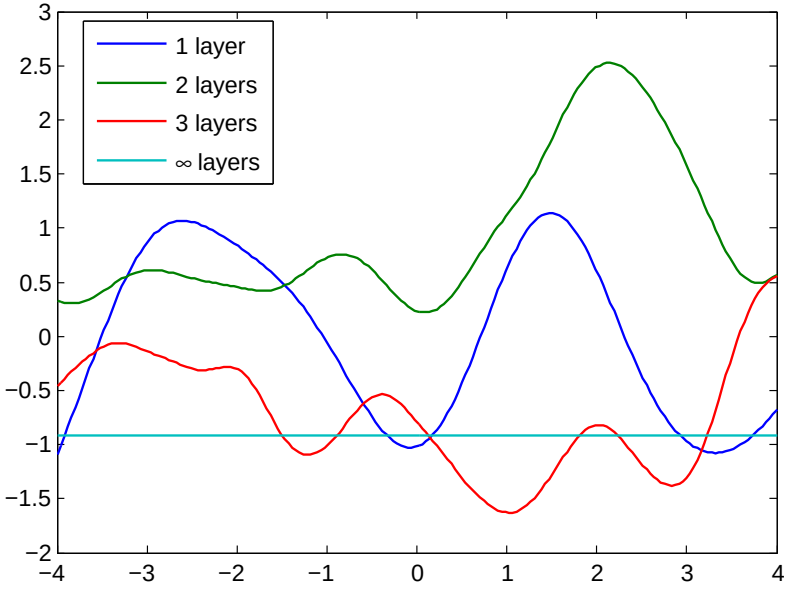


Infinitely Deep Kernels

- For SE kernel, $k_{L+1}(\mathbf{x}, \mathbf{x}') = \exp\left(k_L(\mathbf{x}, \mathbf{x}') - 1\right)$.
- What is the limit of composing SE features?



Kernel

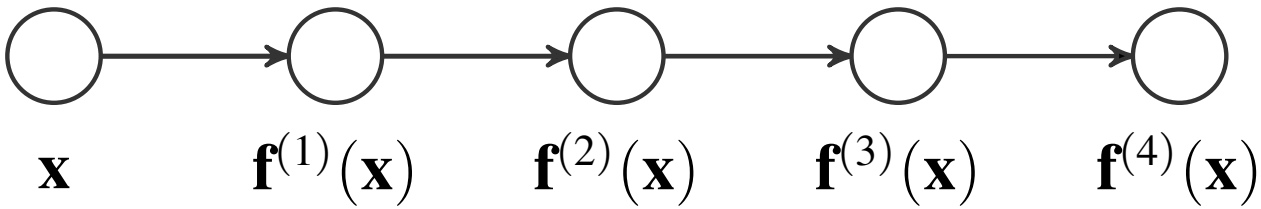


Draws from GP prior

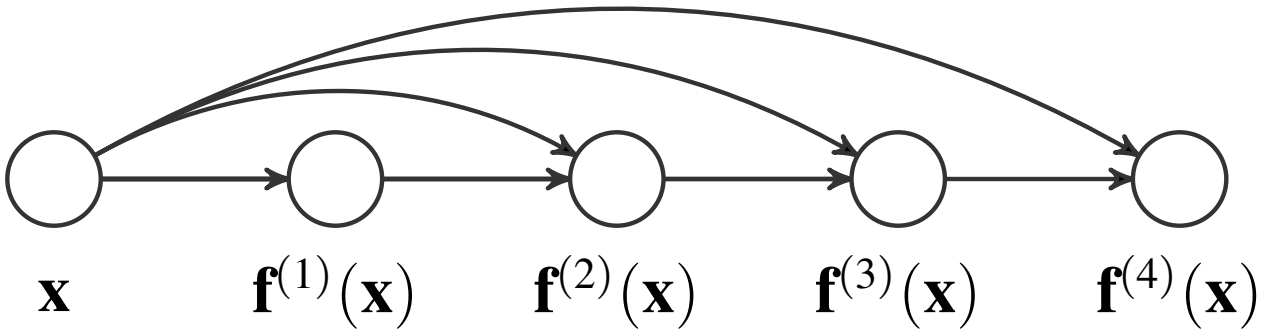
- $k_\infty(\mathbf{x}, \mathbf{x}') = 1$ everywhere. ☹

A simple fix...

- Following a suggestion from Neal (1995), we connect the inputs $\mathbf{x}$ to each layer:



a) standard MLP architecture.



b) Input-connected architecture.

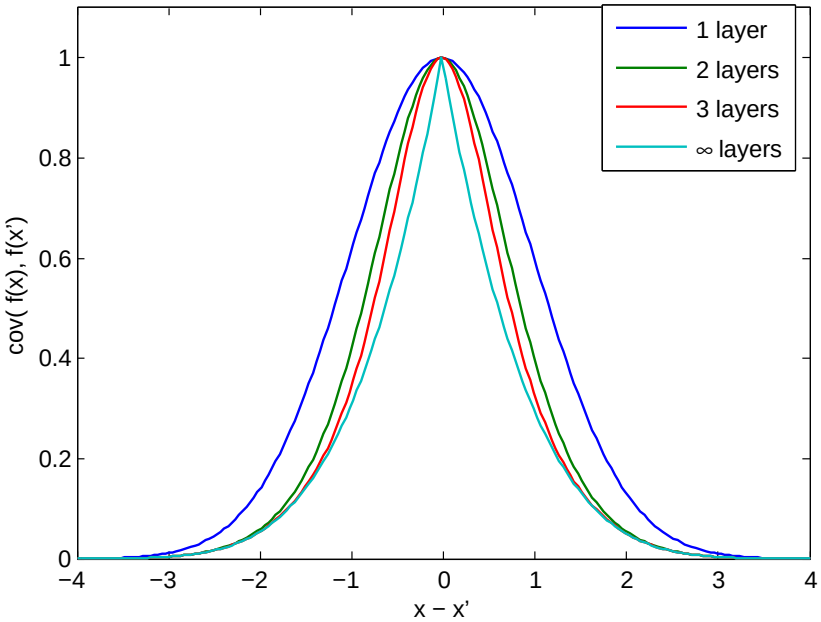
A simple fix...

- Following a suggestion from Neal (1995), we connect the inputs $\mathbf{x}$ to each layer:

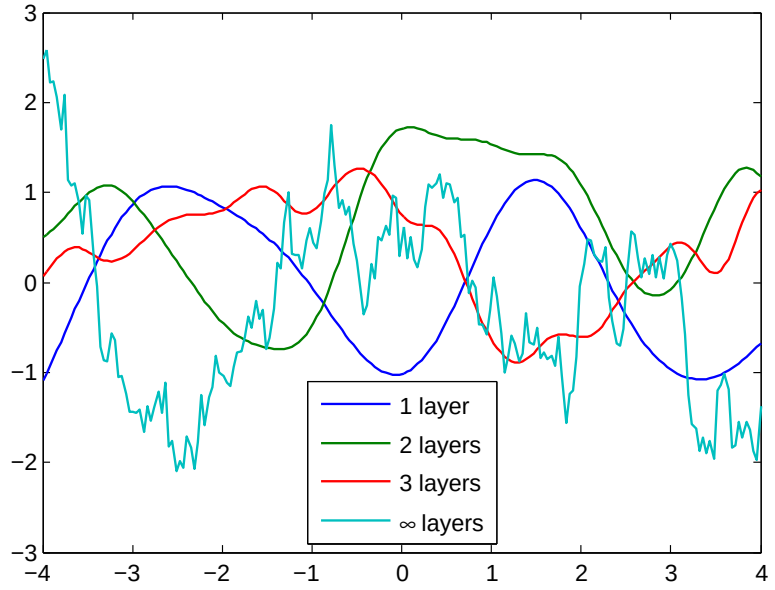
$$
\begin{aligned}
&k_{L+1}(\mathbf{x}, \mathbf{x}') = \\
&= \exp\left(-\frac{1}{2}\left|\left|\begin{bmatrix}\mathbf{h}^L(\mathbf{x}) \\ \mathbf{x}\end{bmatrix} - \begin{bmatrix}\mathbf{h}^L(\mathbf{x}') \\ \mathbf{x}'\end{bmatrix}\right|\right|_2^2\right) \\
&= \exp\left(-\frac{1}{2}\left[k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}')\right] - \frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2\right)
\end{aligned}
$$

Infinitely Deep Kernels

- What is the limit of compositions of input-connected SE features?
- $k_{L+1}(\mathbf{x}, \mathbf{x}') = \exp\left(k_L(\mathbf{x}, \mathbf{x}') - 1 - \frac{1}{2}||\mathbf{x} - \mathbf{x}'||_2^2\right).$



Kernels



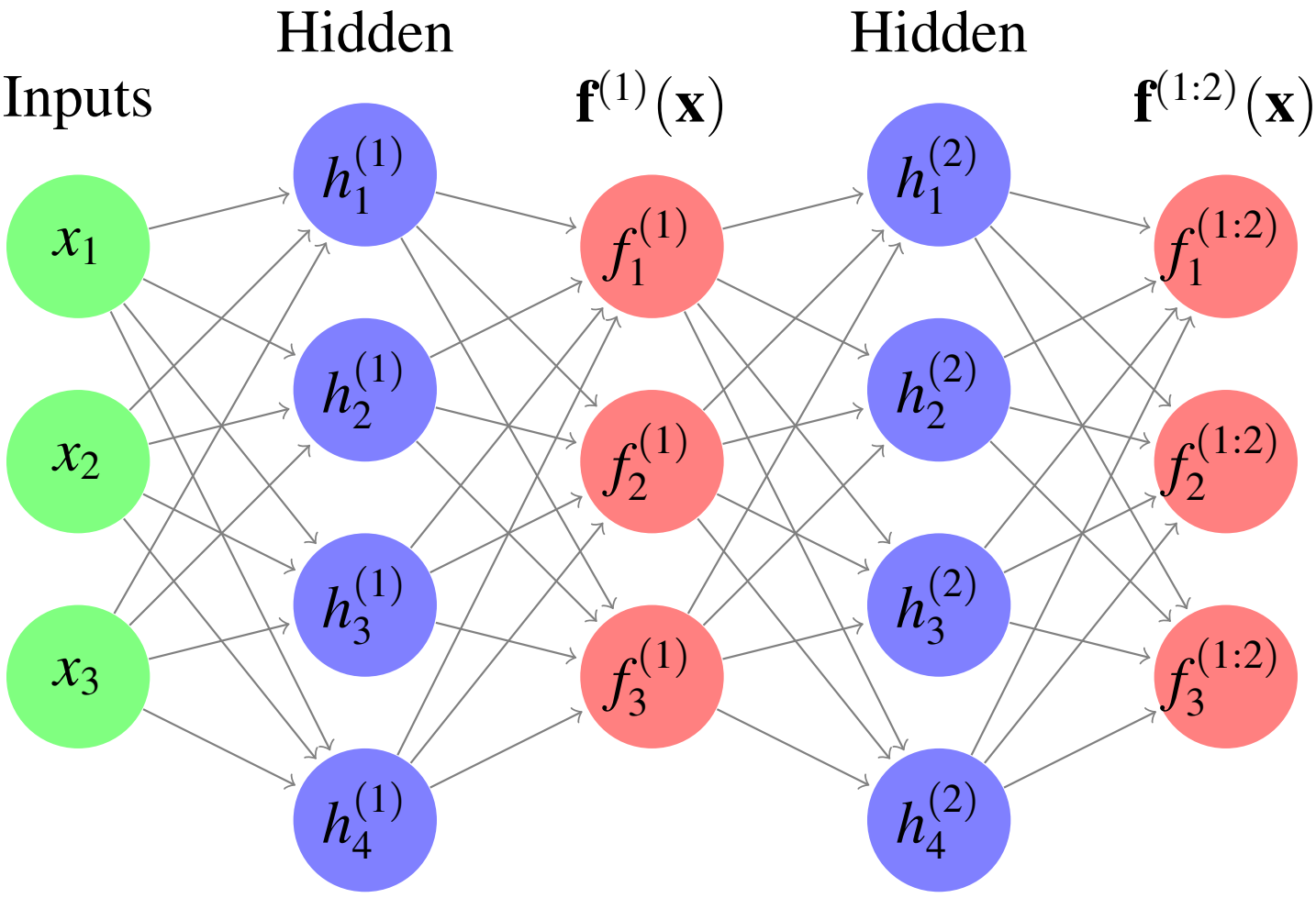
Draws from GP priors

- Like an Ornstein-Uhlenbeck process with skinny tails
- Samples are non-differentiable (fractal).

What went wrong?

- Fixed feature mapping, unlikely to be useful for anything
- power of neural nets comes from learning a custom representation
- Need to search over feature mappings!
- Can try to learn kernels, or even better, integrate over feature mappings

Deep Gaussian Processes



Deep Gaussian Processes

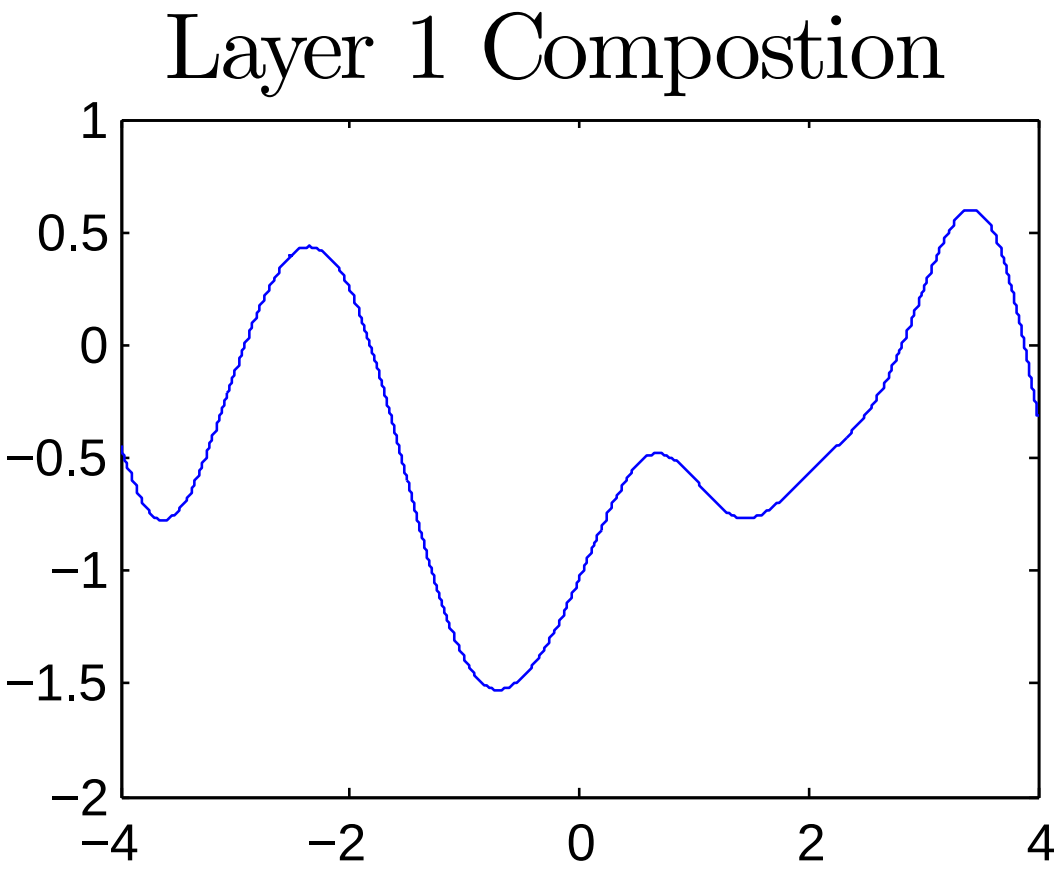
- a prior over compositions of functions:

$$\mathbf{f}^{(1:L)}(\mathbf{x}) = \mathbf{f}^{(L)}(\mathbf{f}^{(L-1)}(\ldots \mathbf{f}^{(2)}(\mathbf{f}^{(1)}(\mathbf{x}))\ldots)) \tag{1}$$

  where each $\mathbf{f}_d^{(\ell)} \overset{\text{ind}}{\sim} \mathcal{GP}\left(0, k_d^{\ell}(\mathbf{x}, \mathbf{x}')\right)$.
- Analogous to neural nets, where each neuron's activation function is an independent draw from a GP.
- inference is really hard.
- maybe we can learn something just from looking at draws?

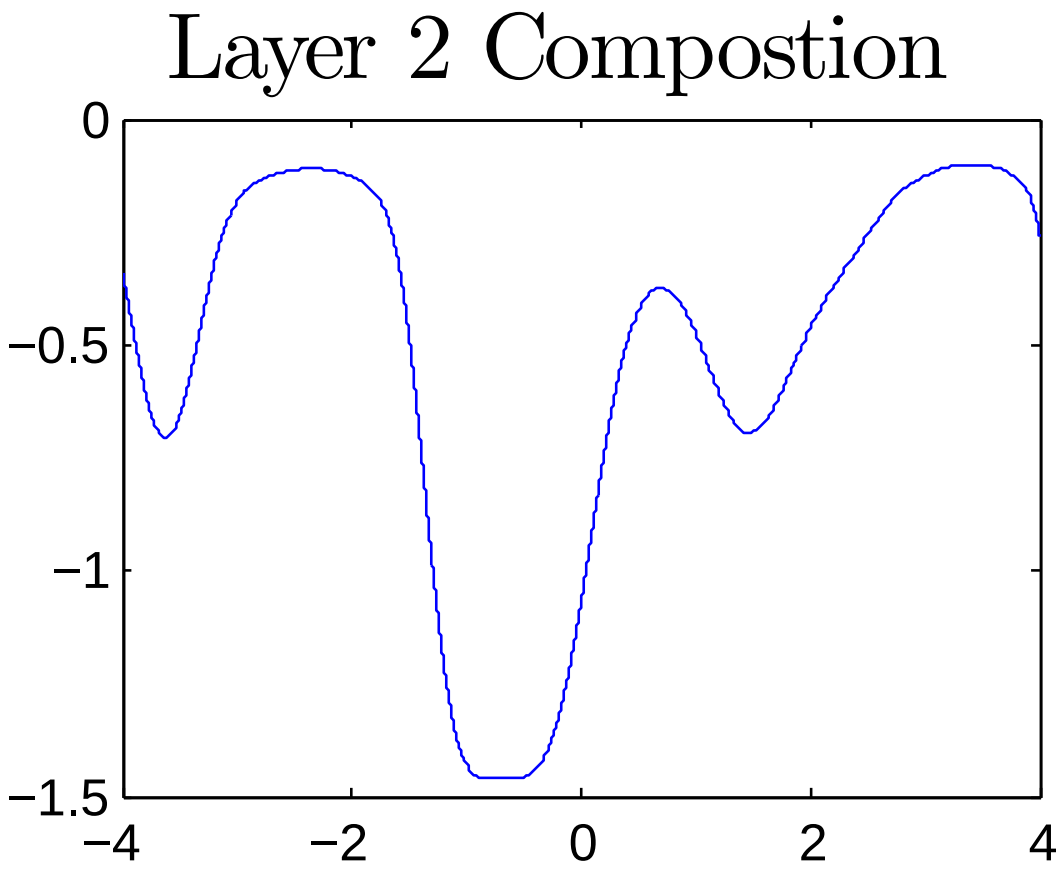
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



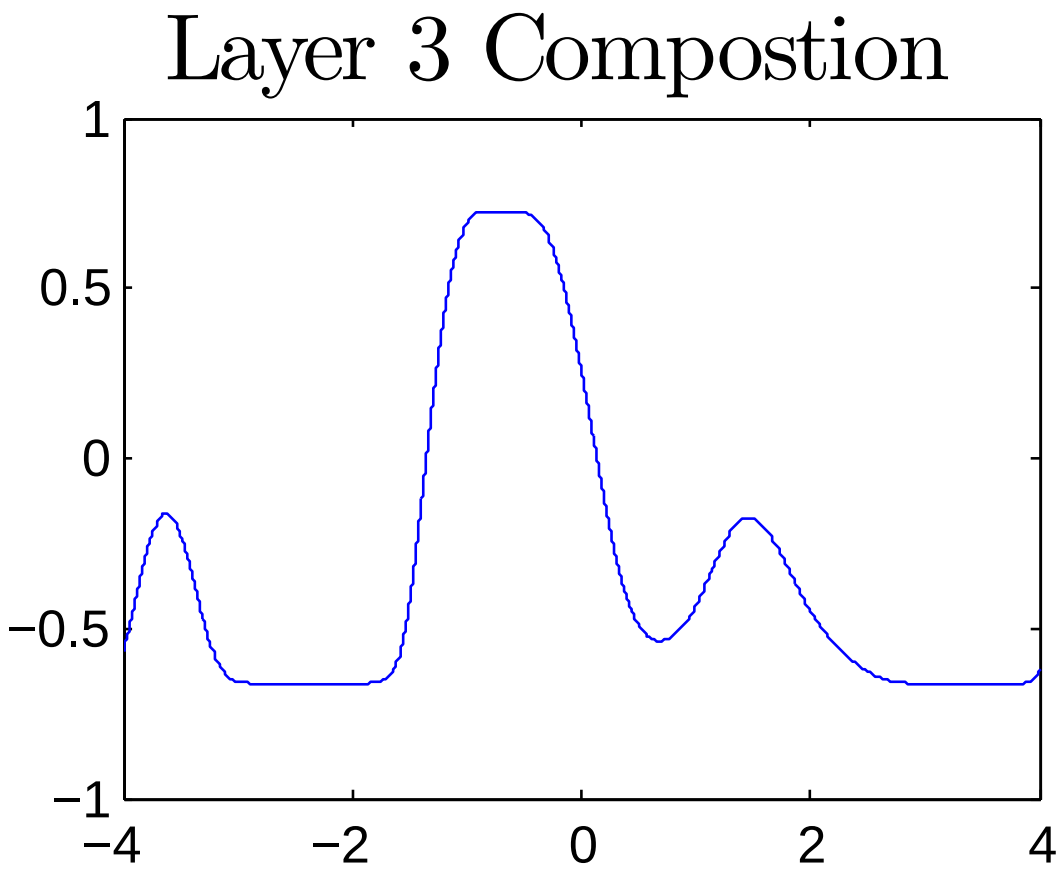
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



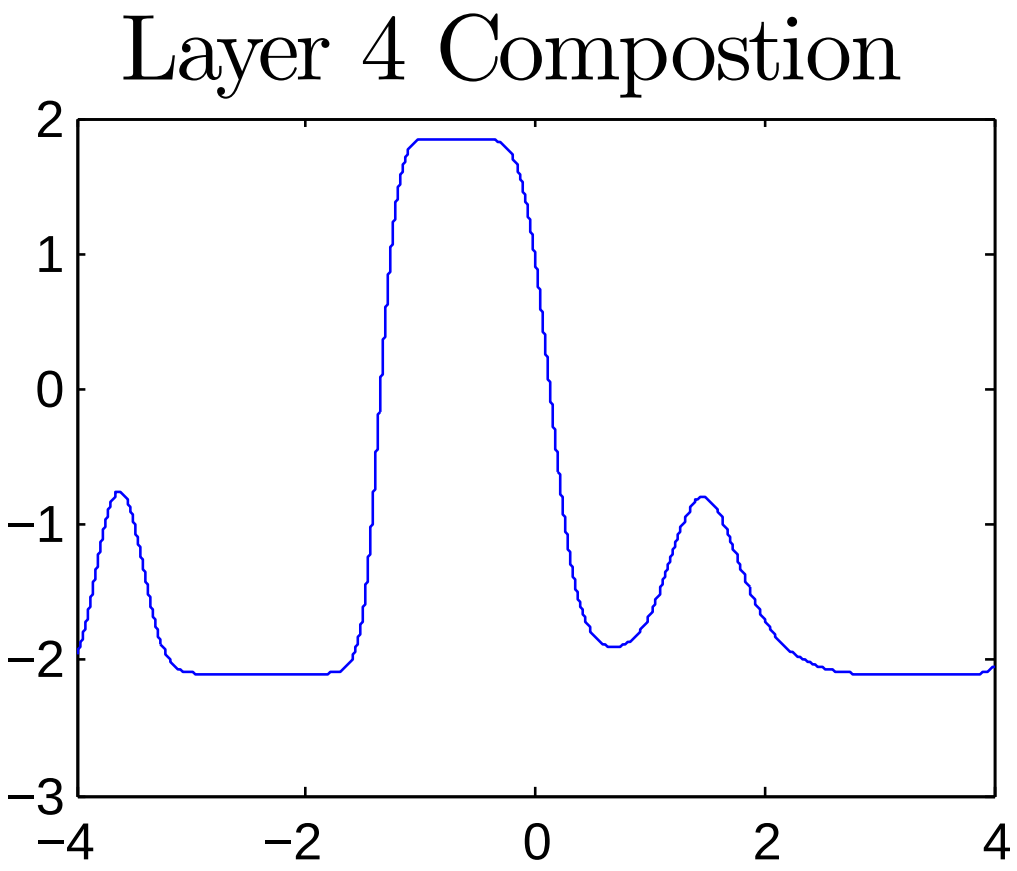
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



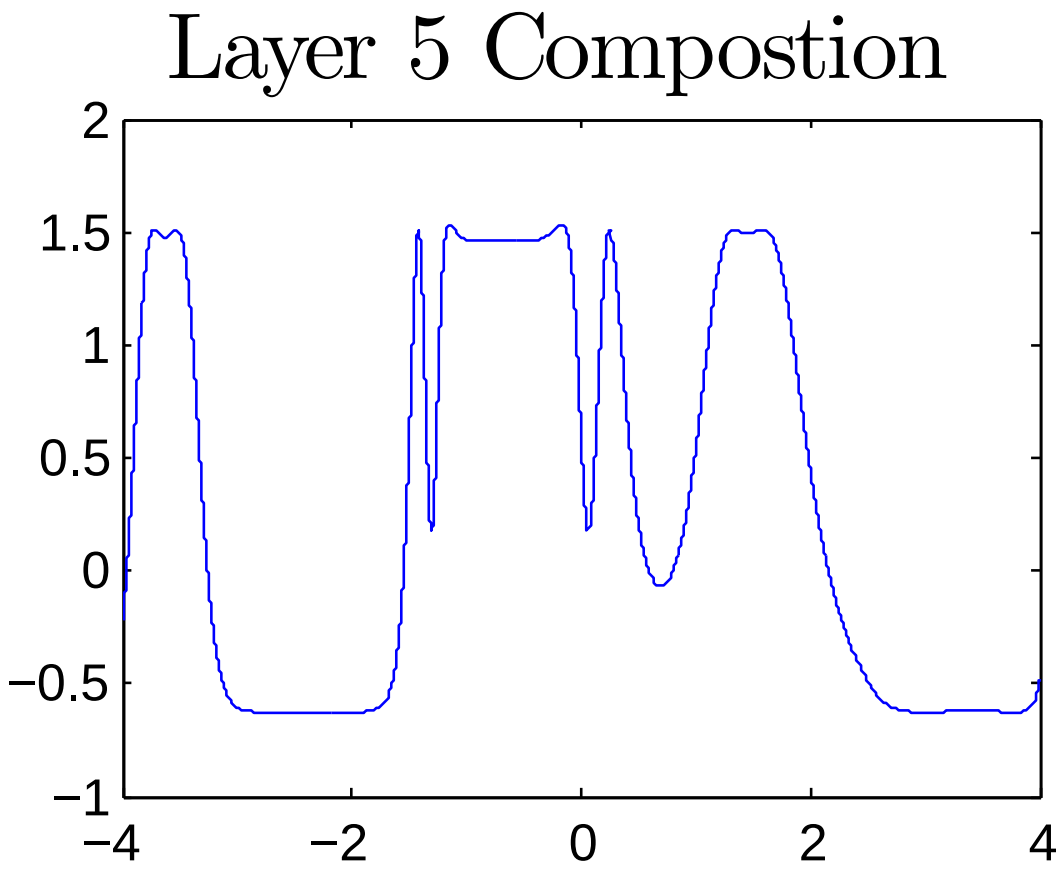
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



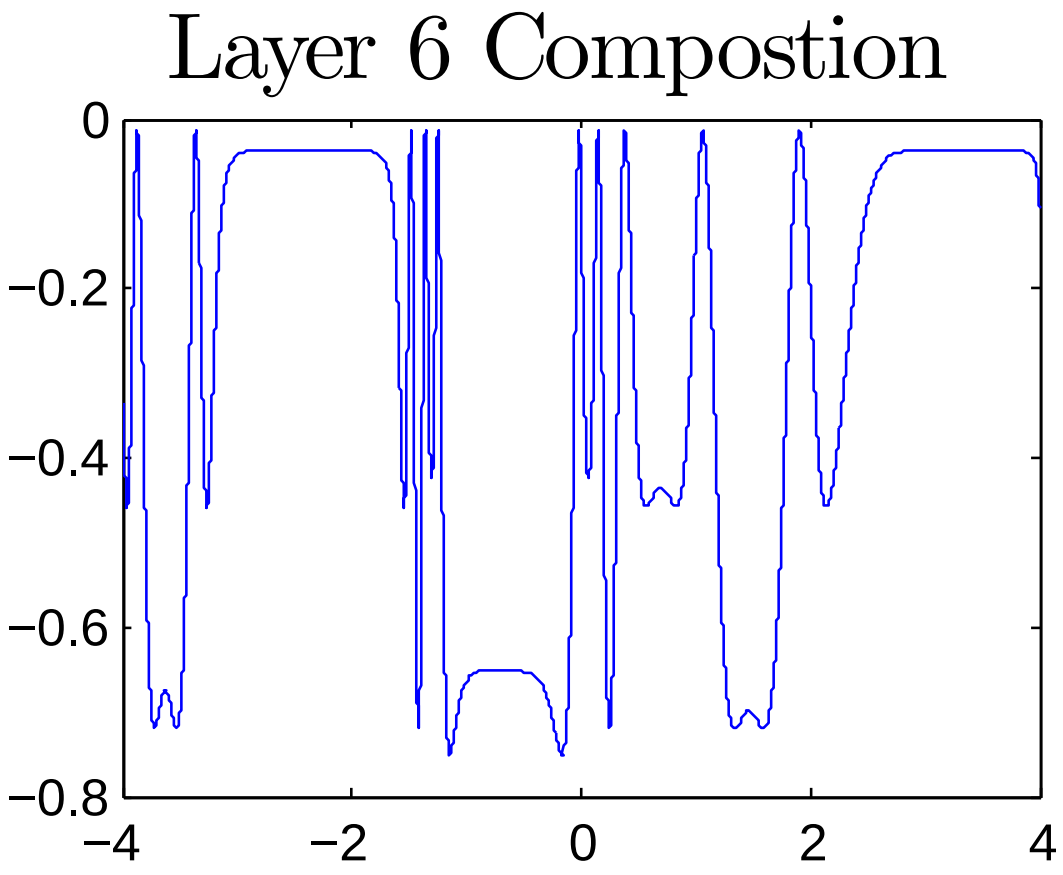
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



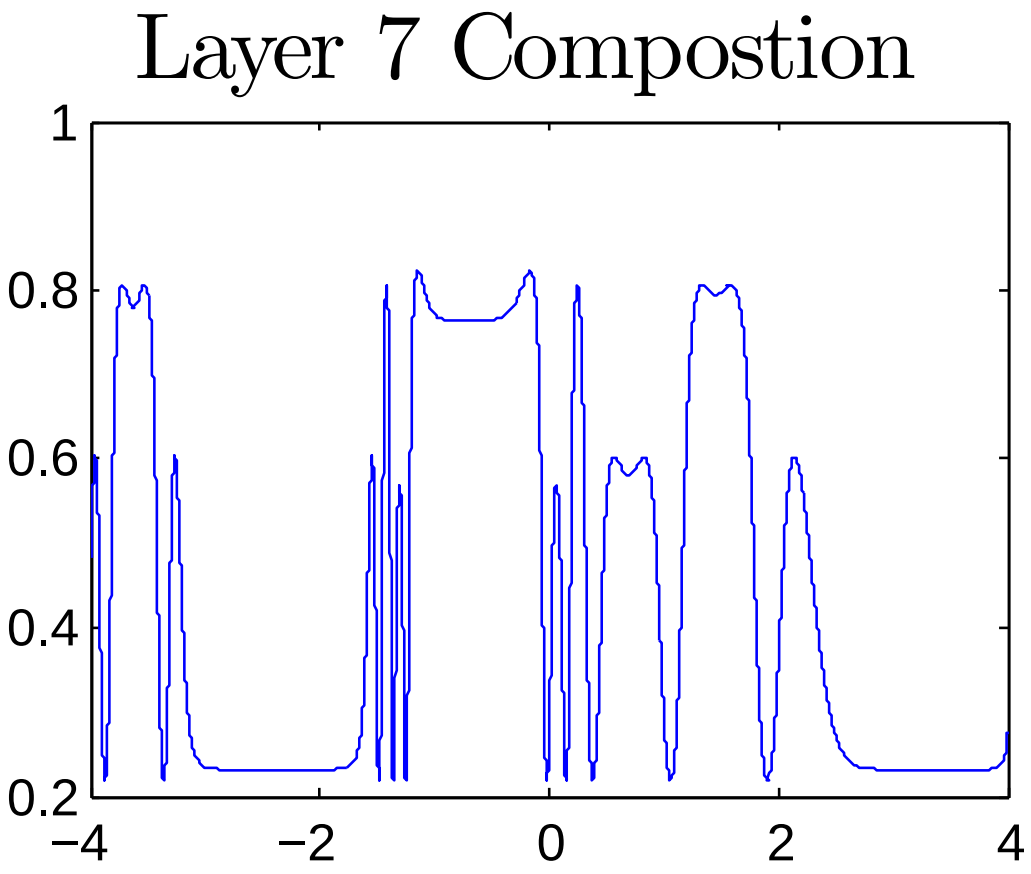
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



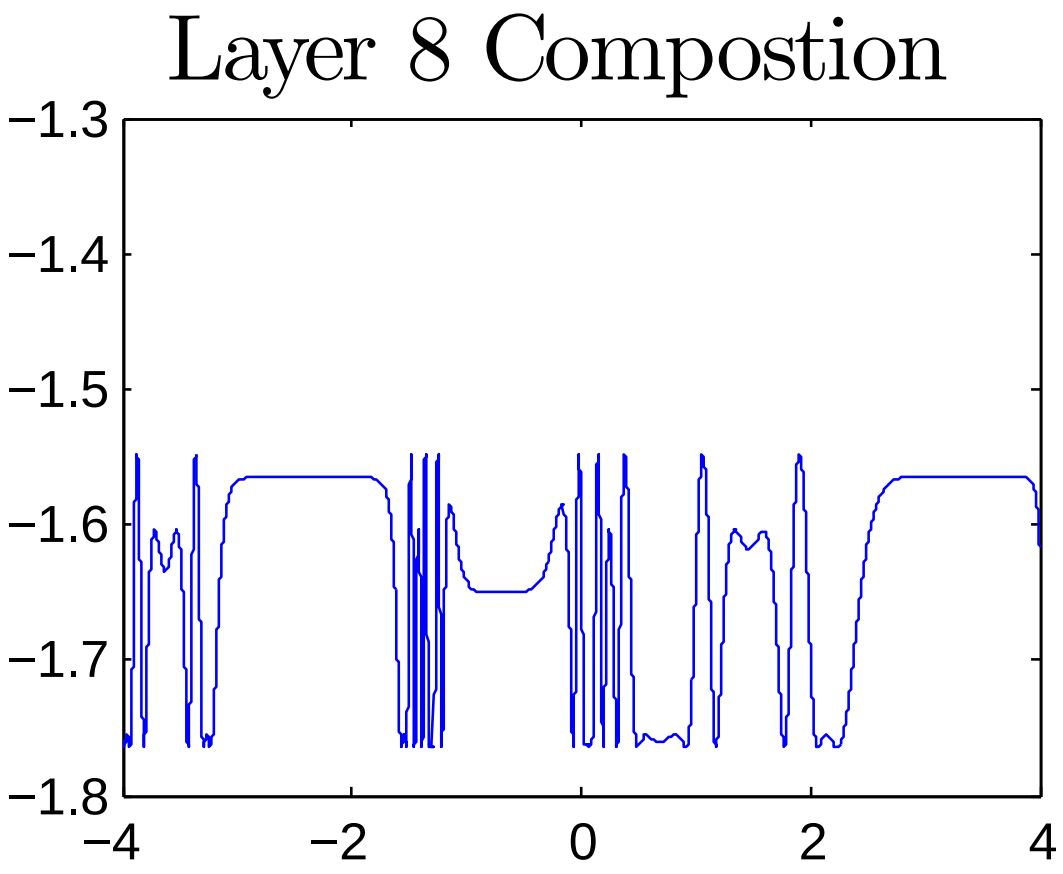
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



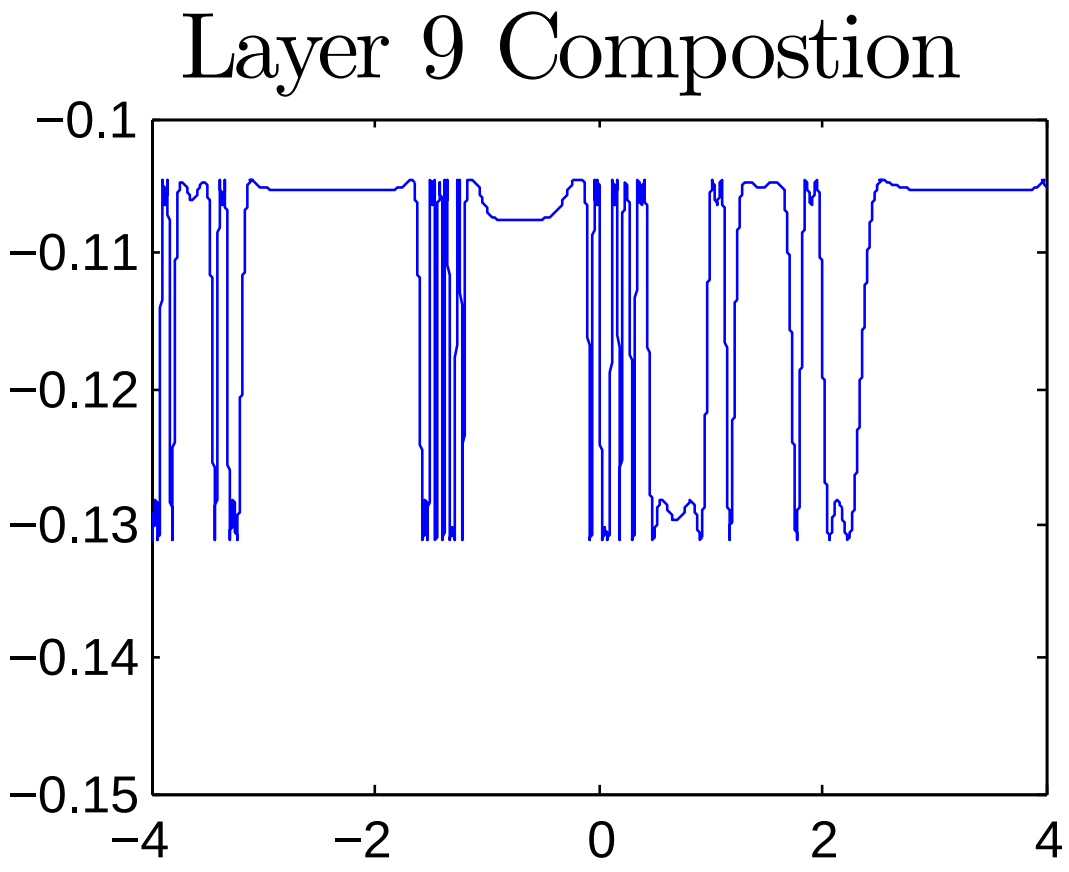
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



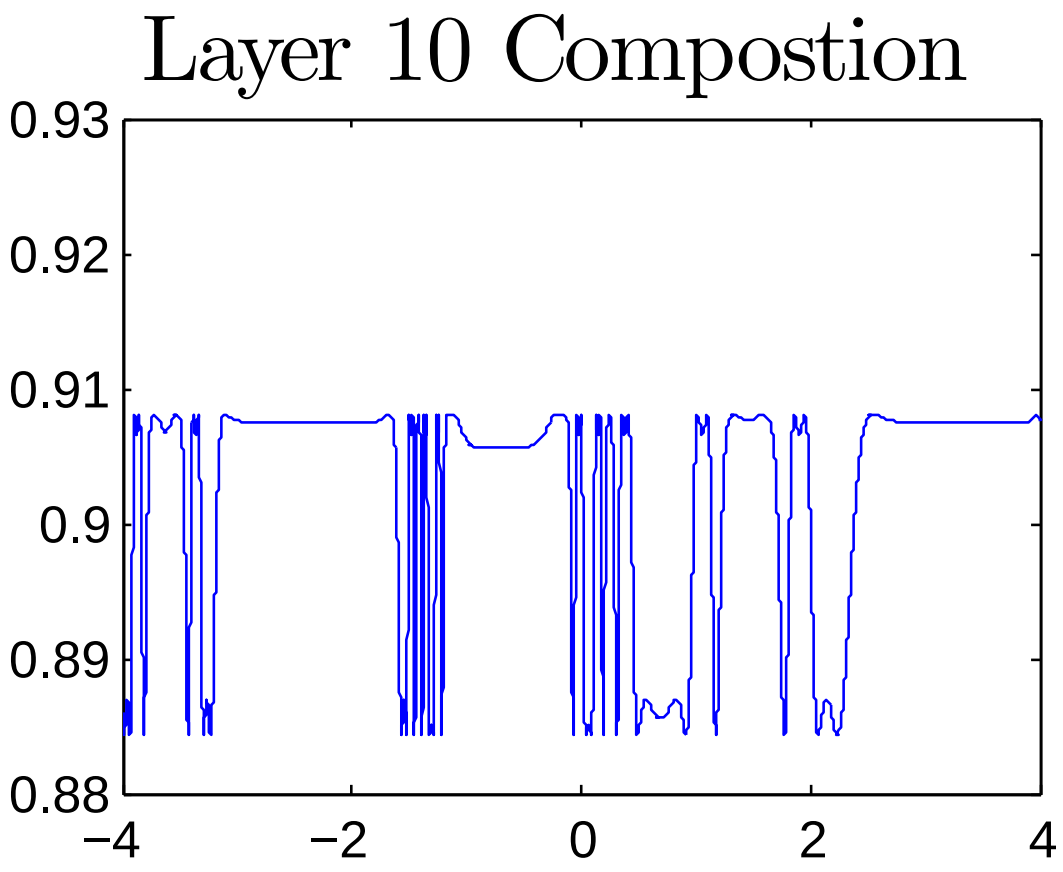
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



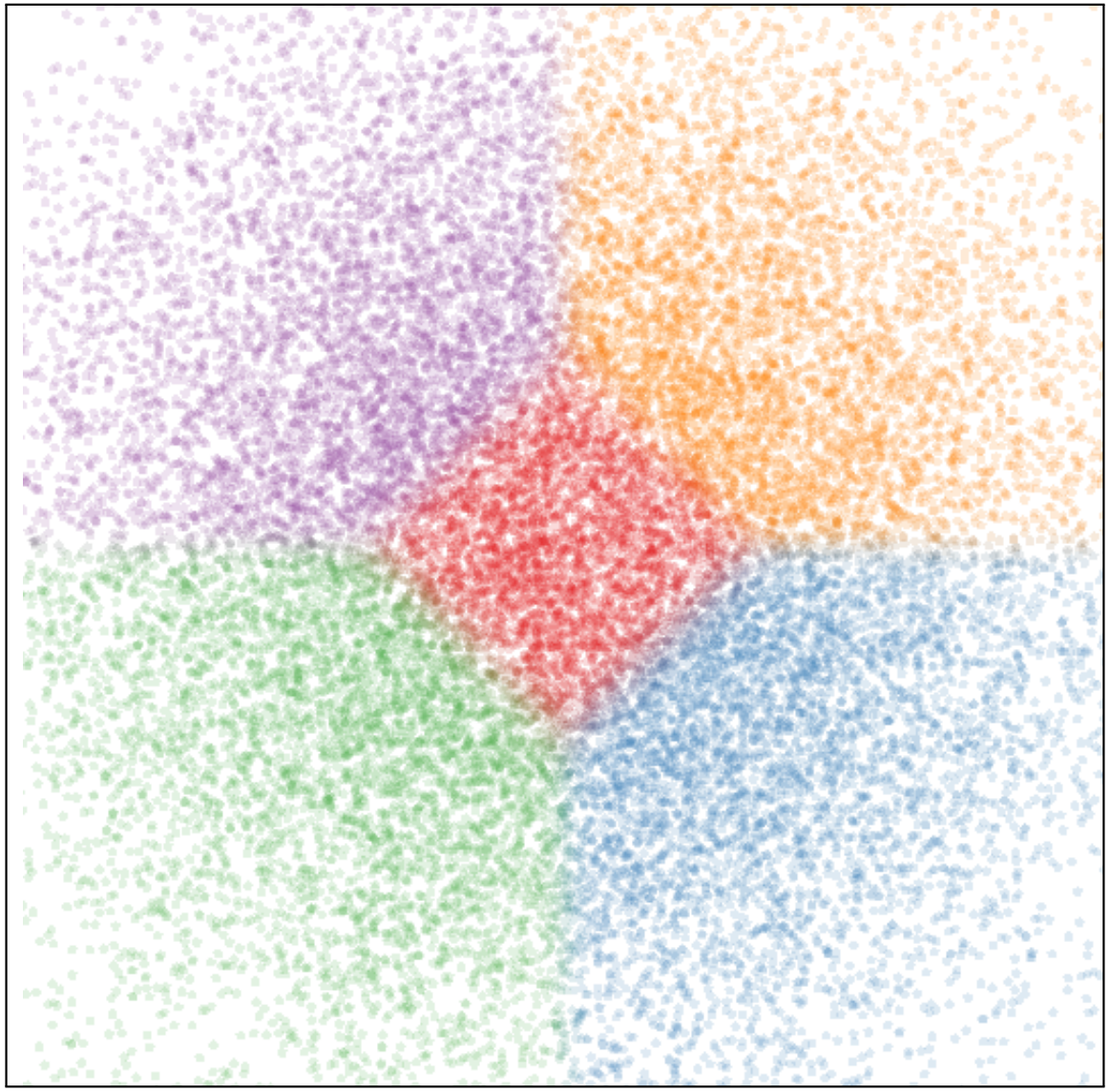
Deep Gaussian Processes

- Draws from one-dimensional deep GPs:



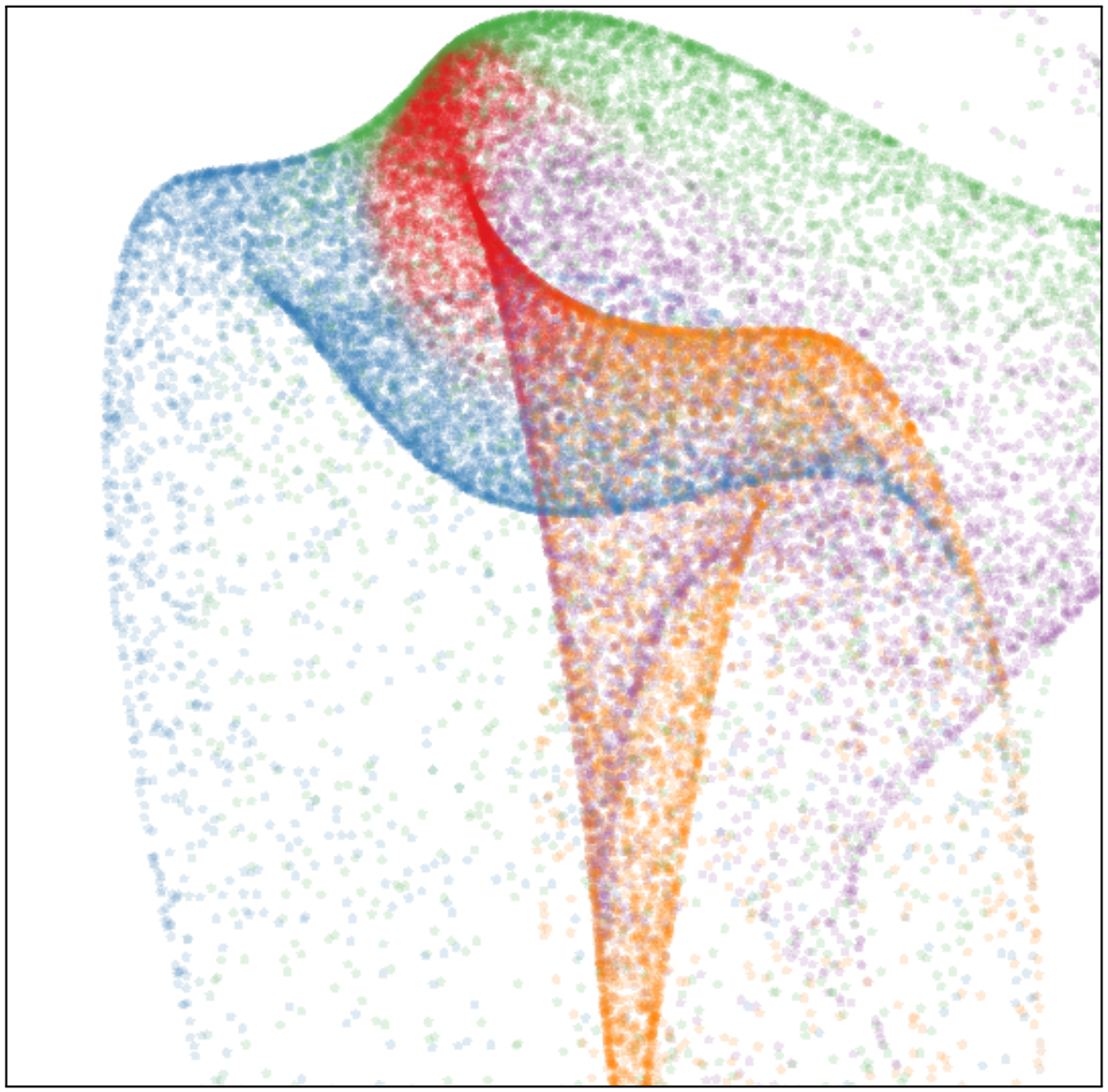
Deep Gaussian Processes

- 2D to 2D warpings of a Gaussian density:



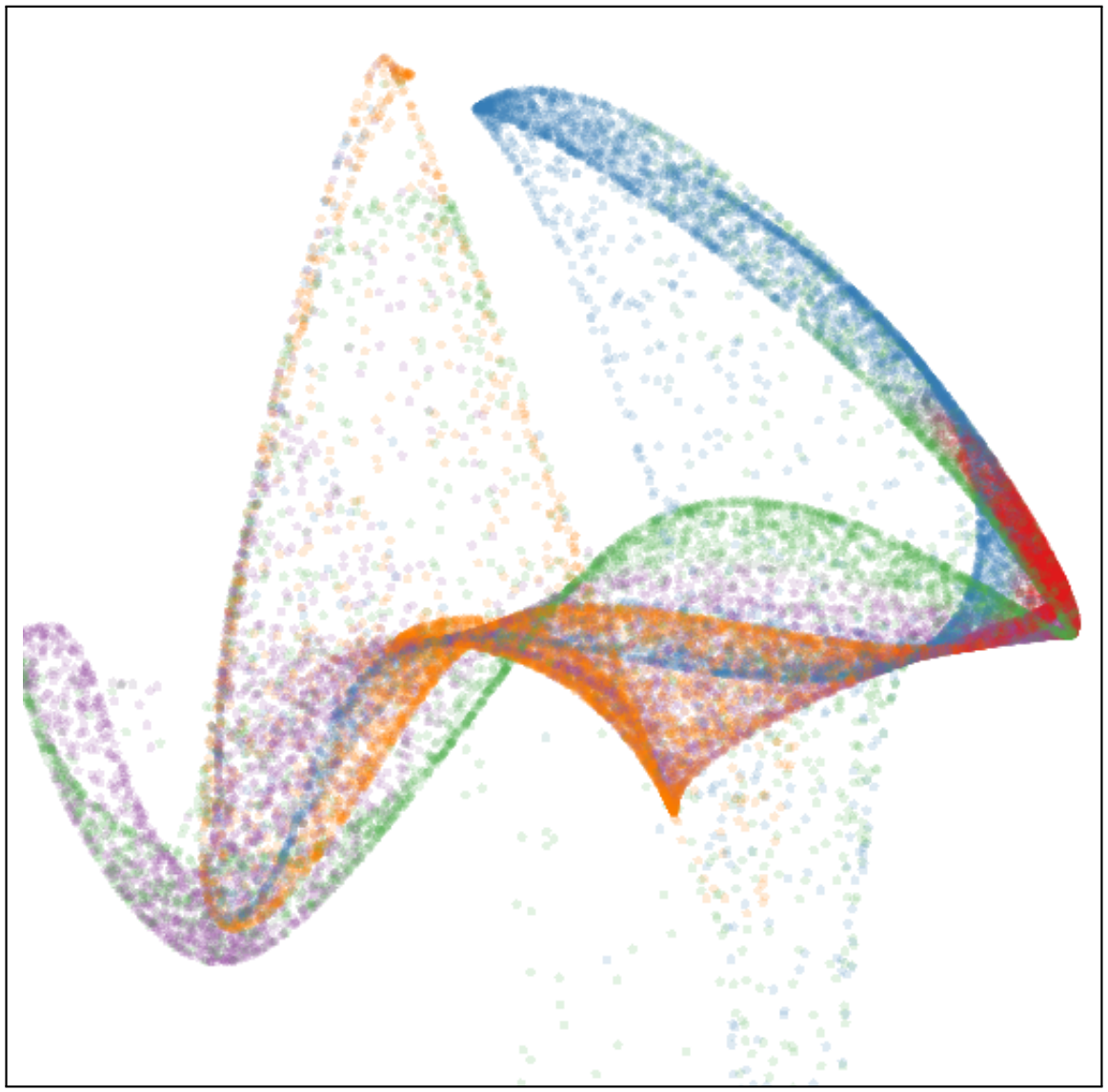
Deep Gaussian Processes

- 2D to 2D warpings of a Gaussian density:



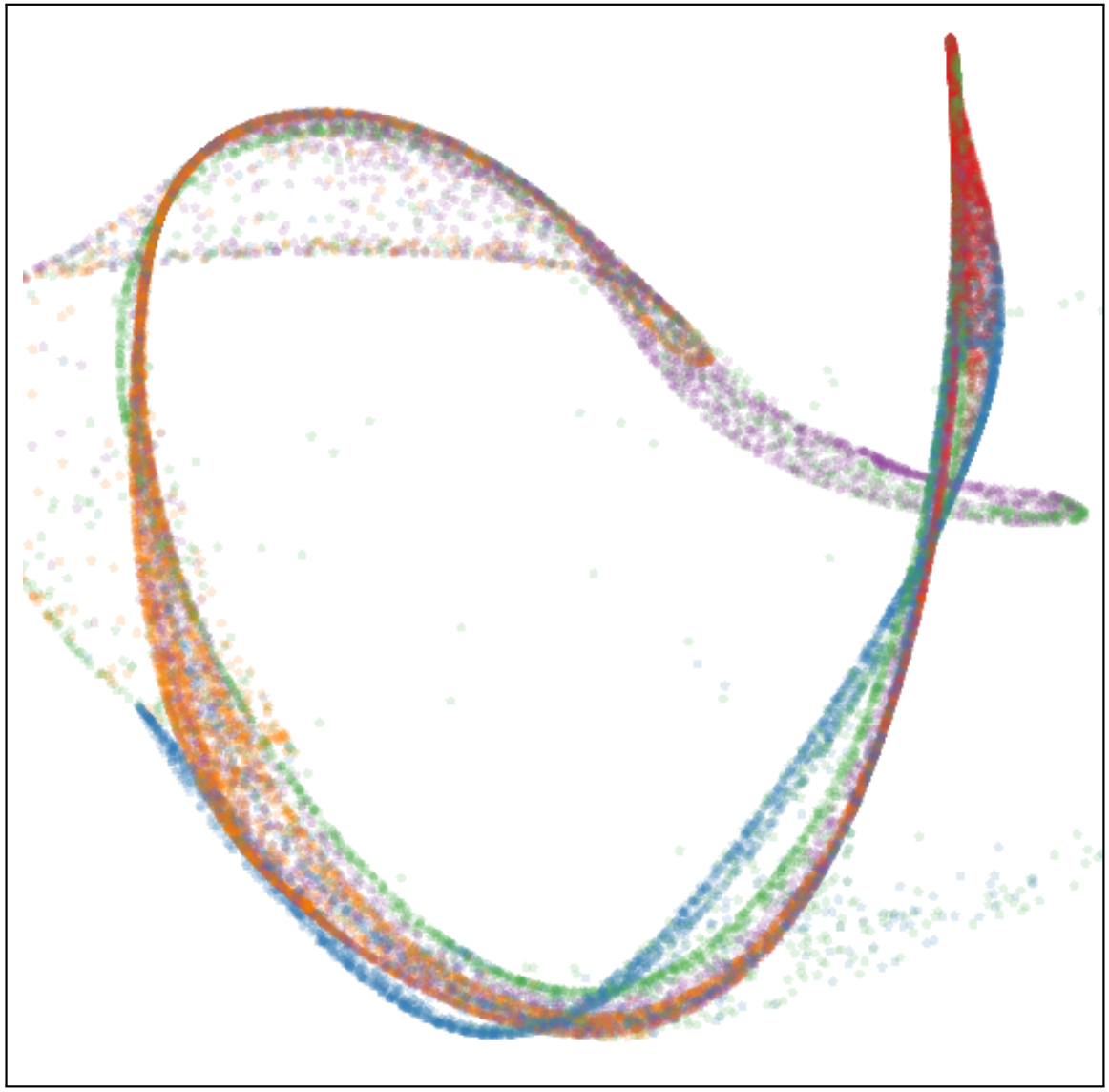
Deep Gaussian Processes

- 2D to 2D warpings of a Gaussian density:



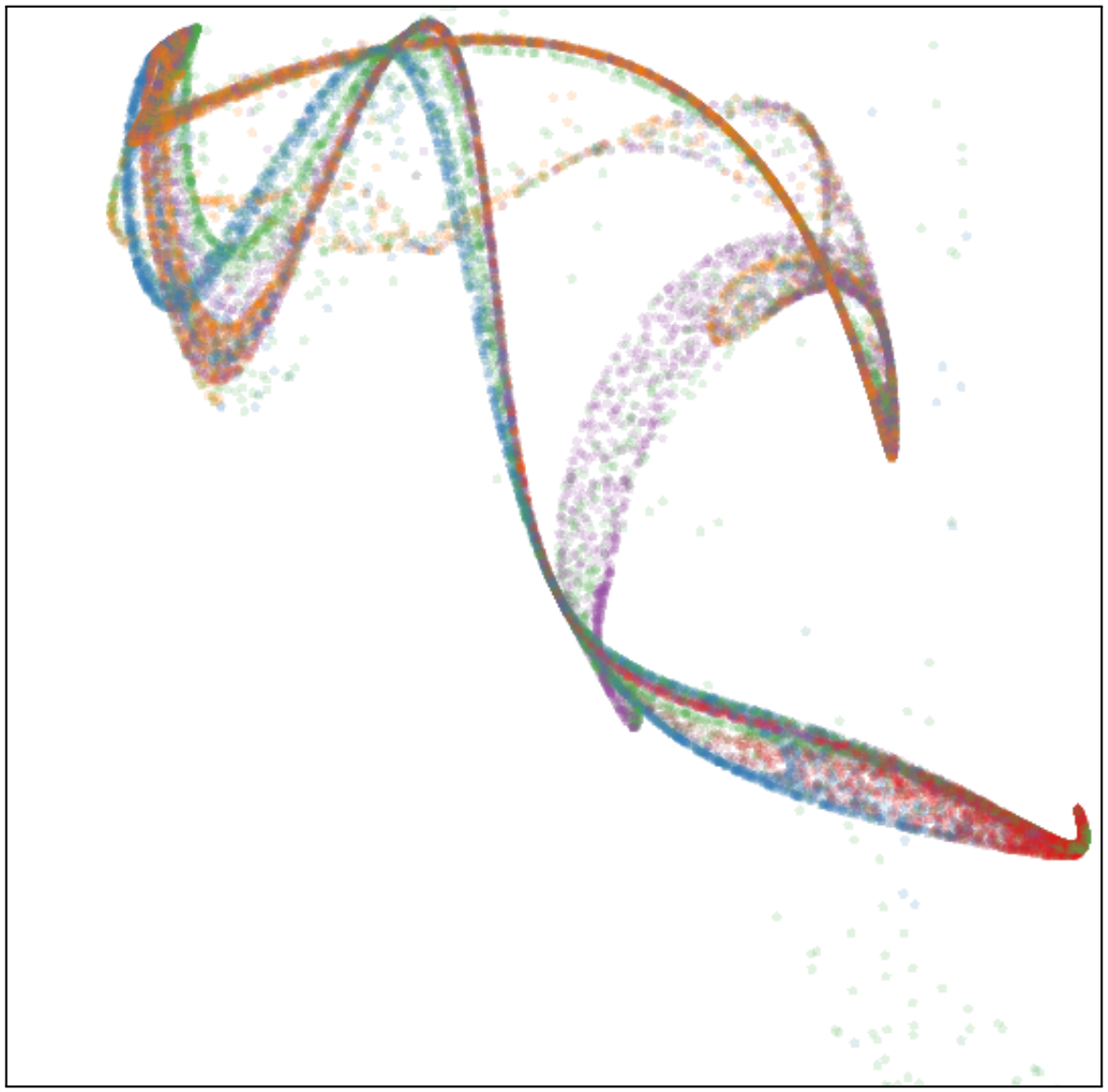
Deep Gaussian Processes

- 2D to 2D warpings of a Gaussian density:



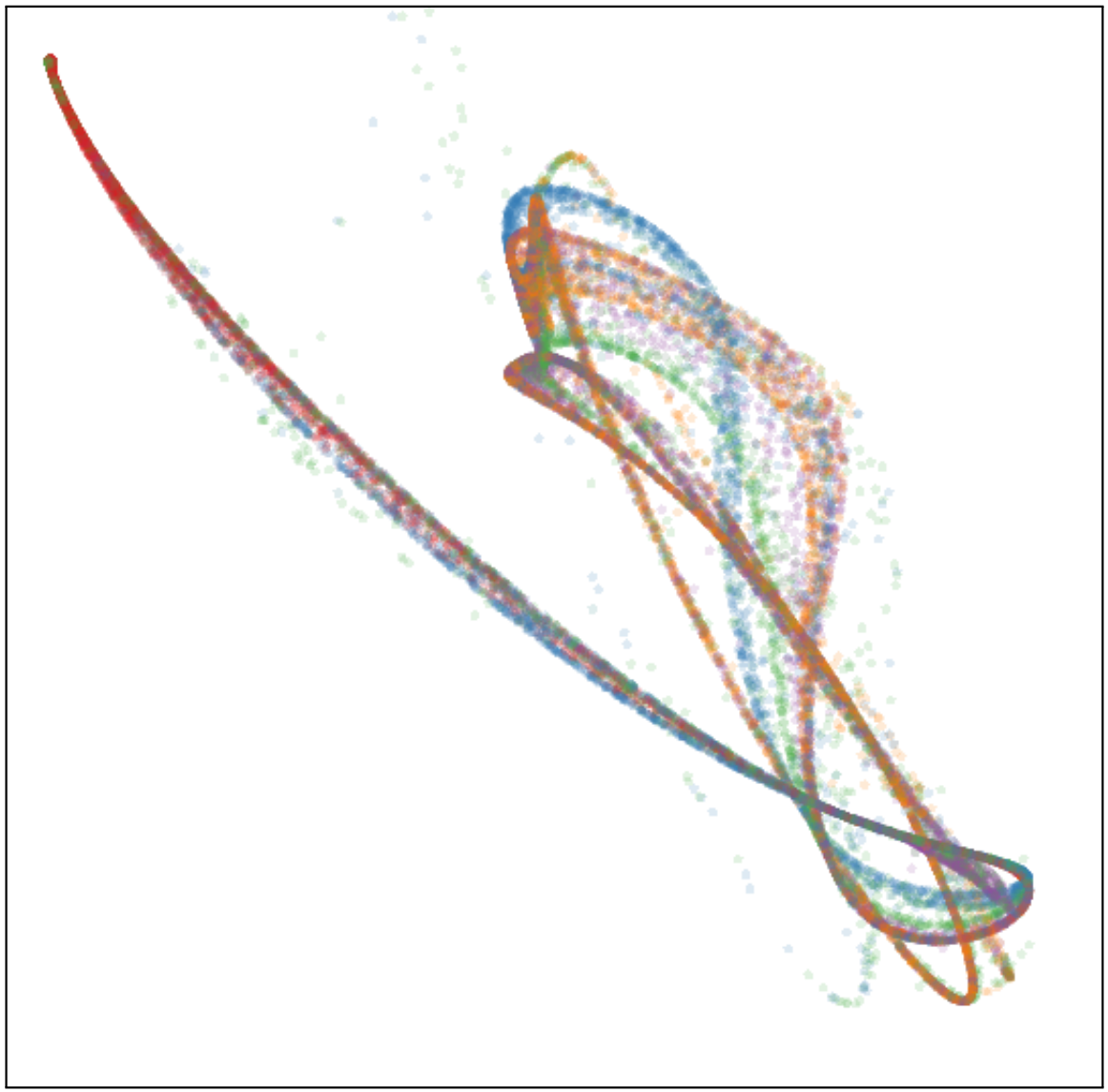
Deep Gaussian Processes

- 2D to 2D warpings of a Gaussian density:



Deep Gaussian Processes

- 2D to 2D warpings of a Gaussian density:



Deep Gaussian Processes

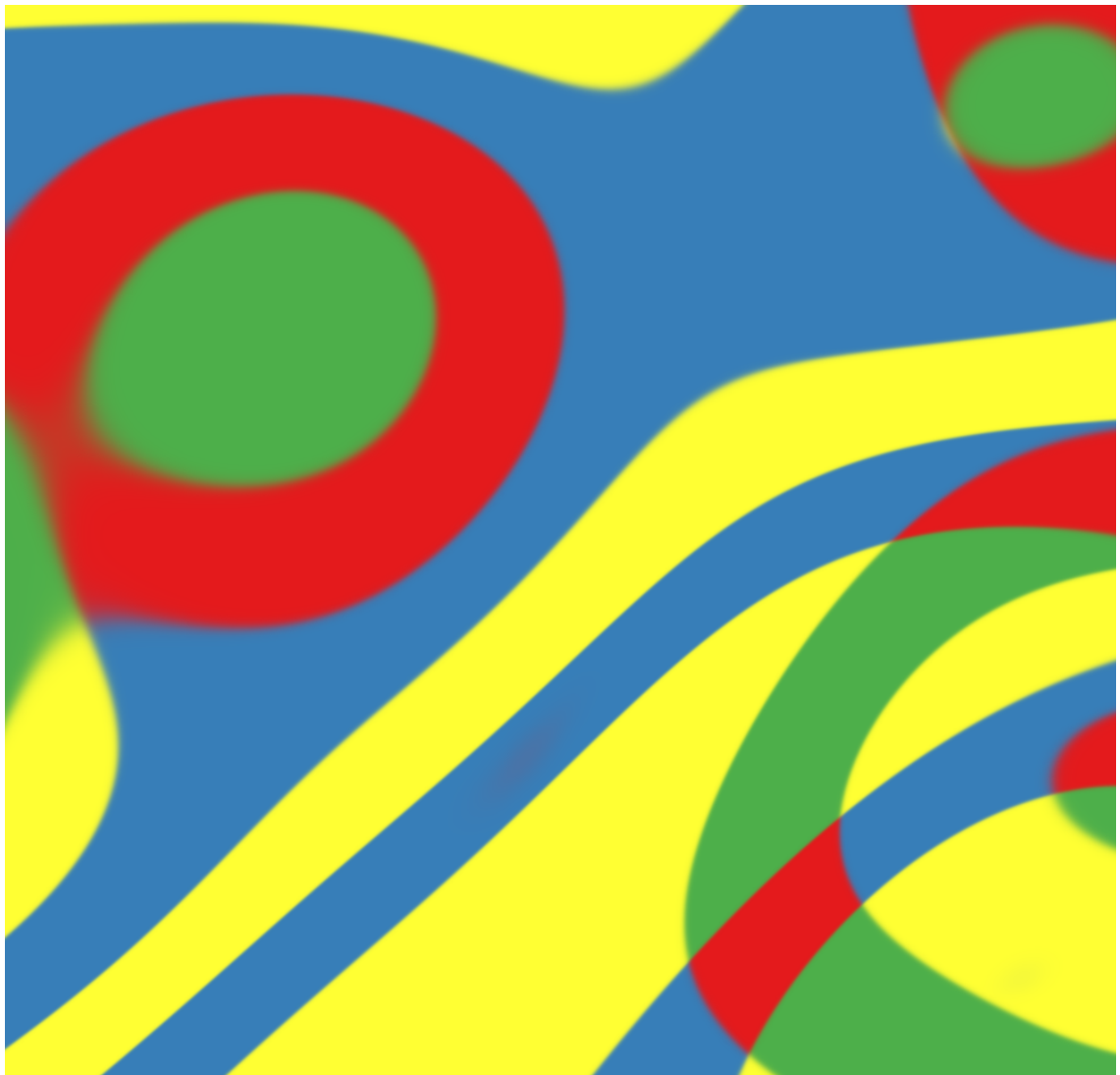
- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

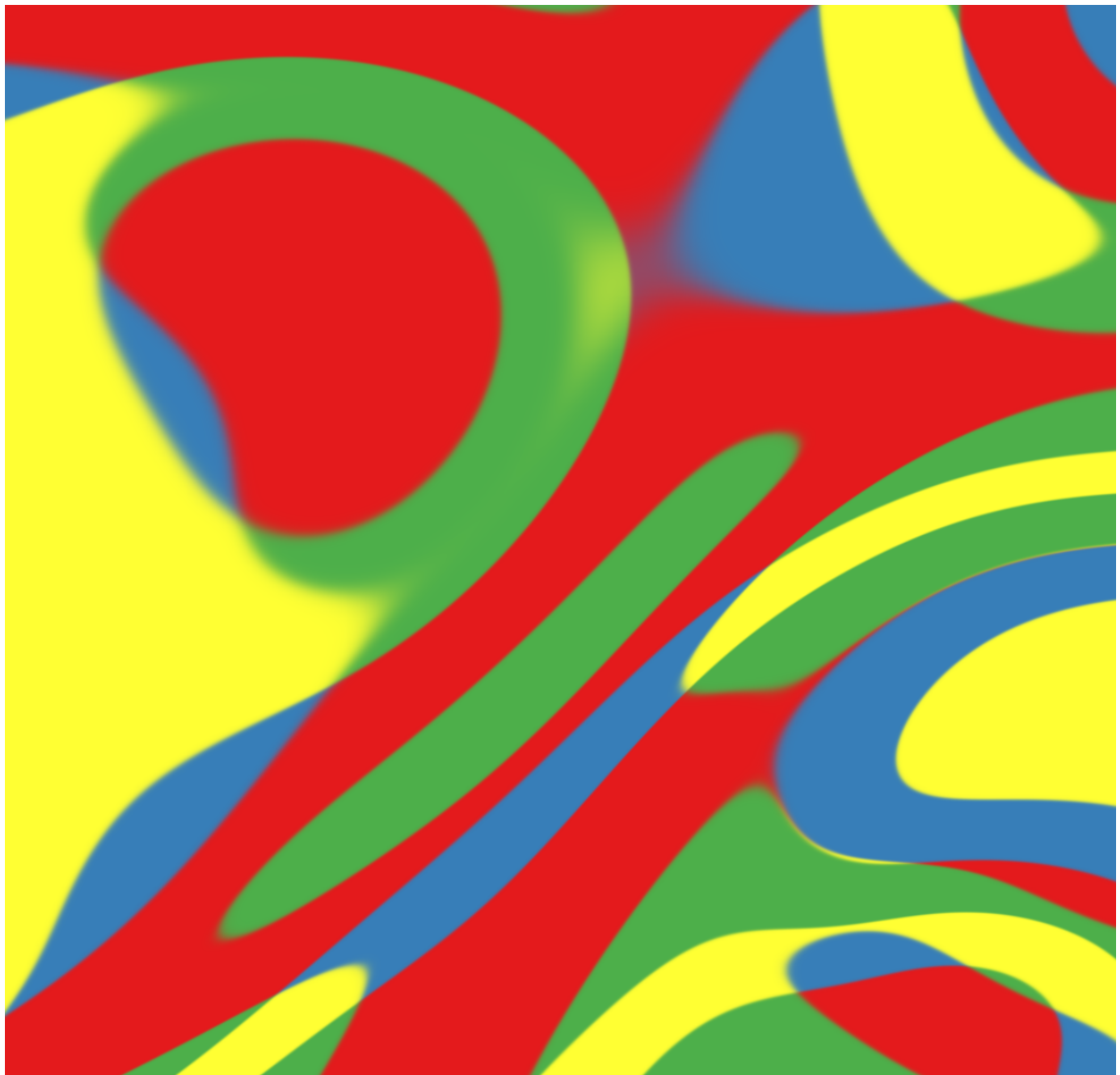


Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

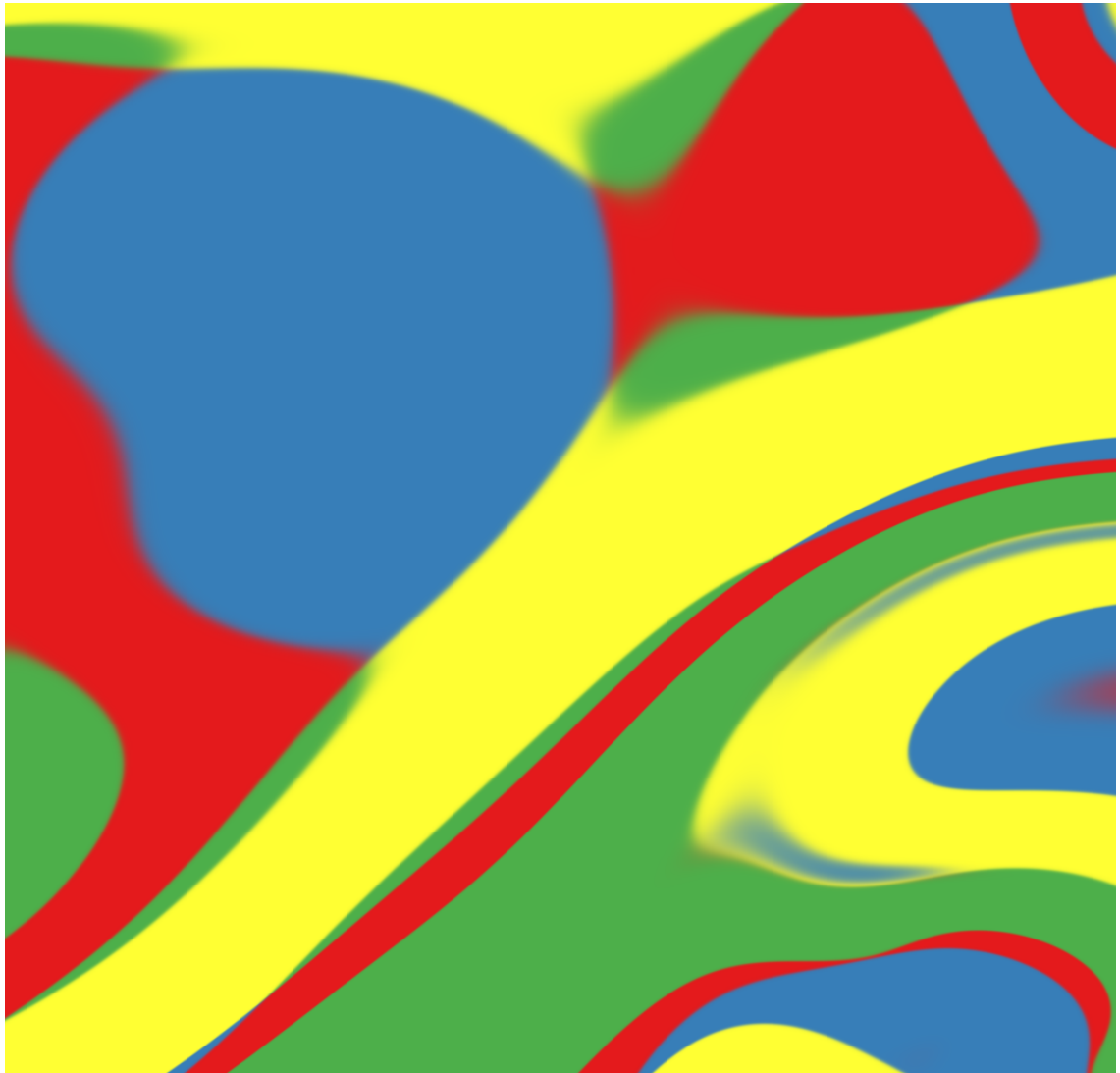
Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)



Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

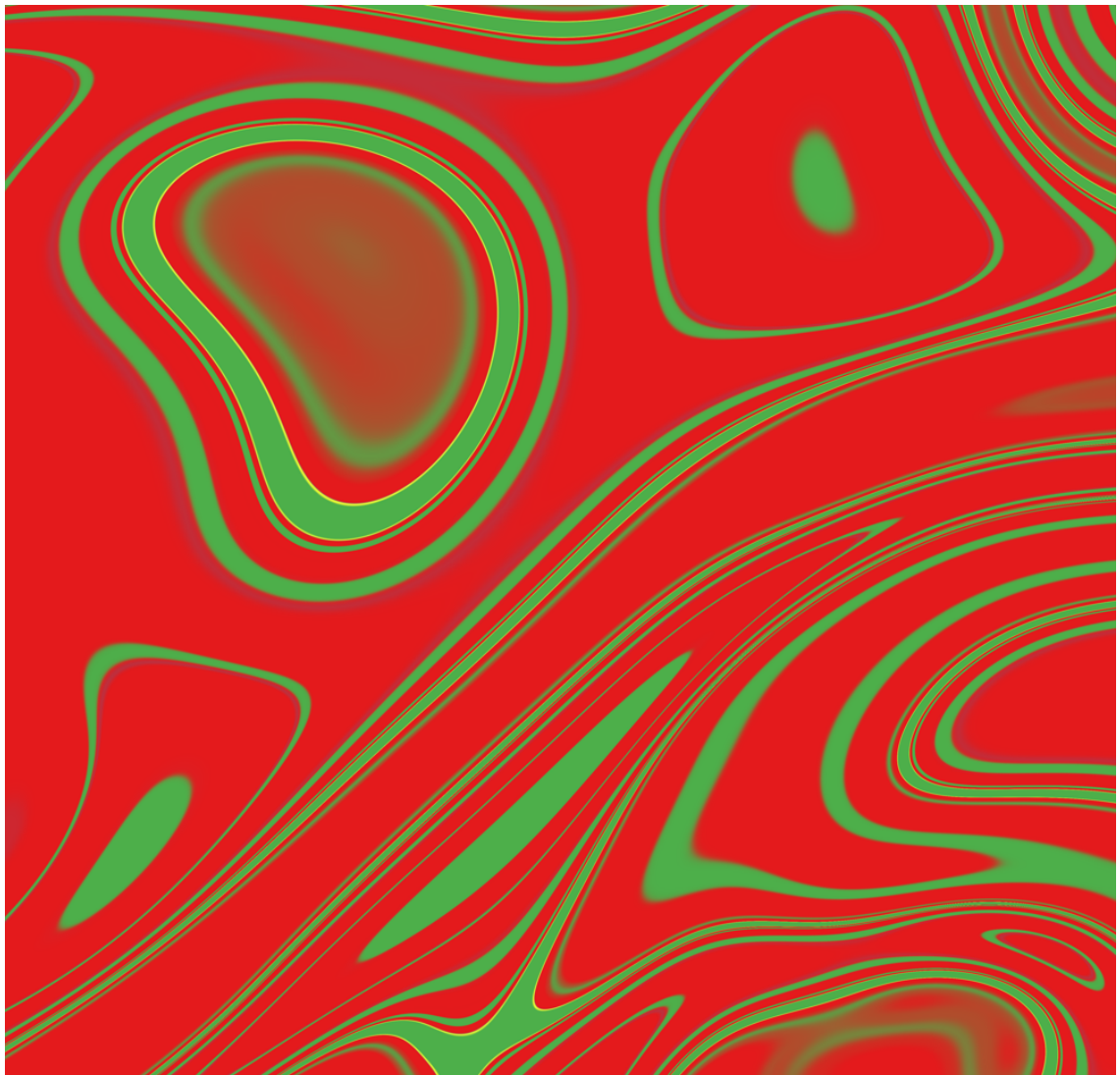


Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

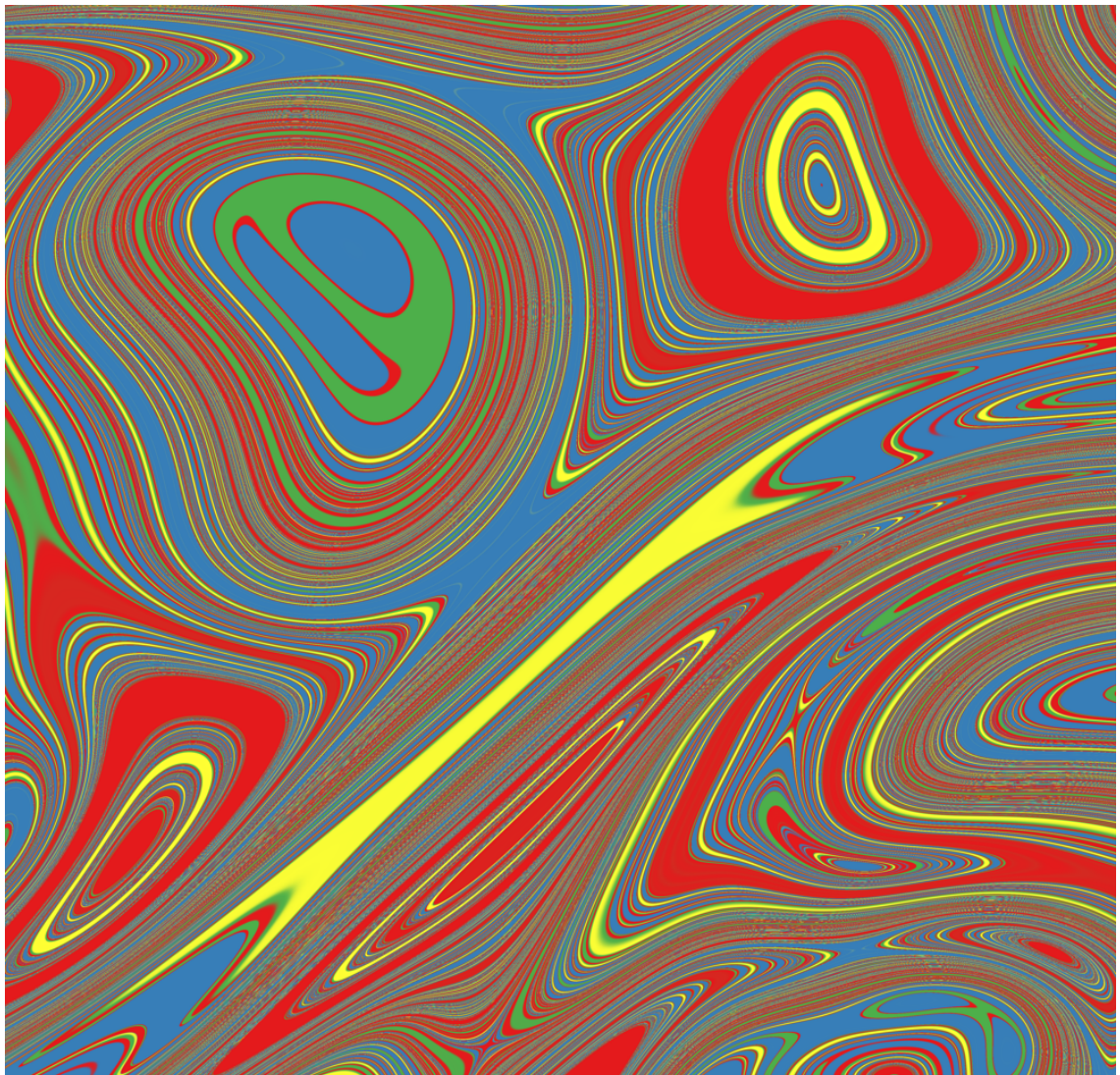
Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)



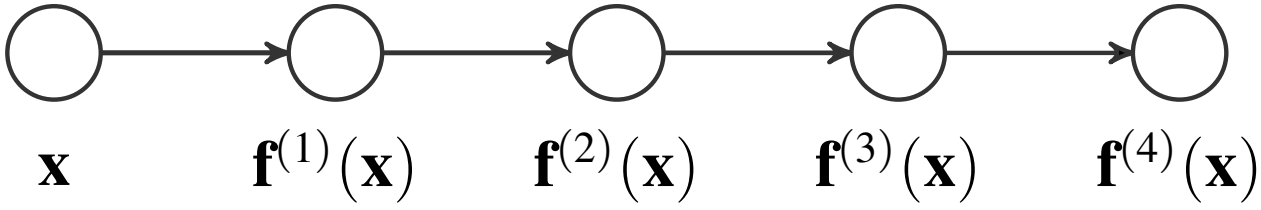
Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

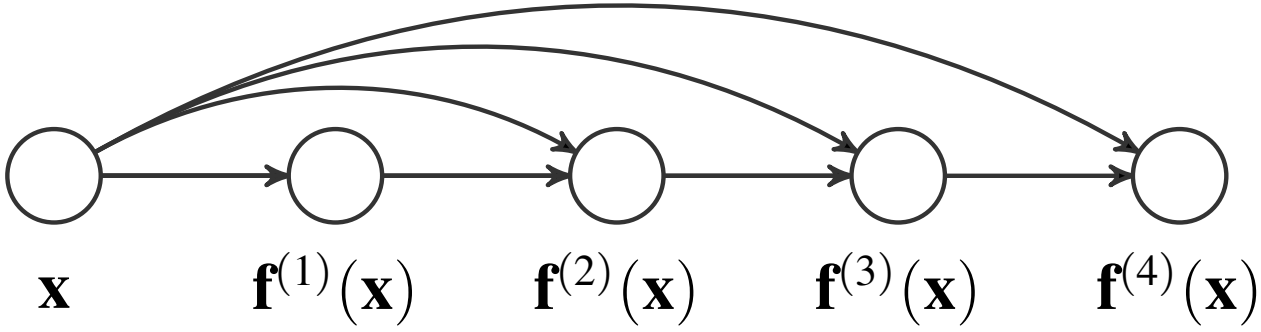


Deep Gaussian Processes

- Only one degree of freedom in x is being captured!
- Again following Radford's thesis, connect every layer to input:



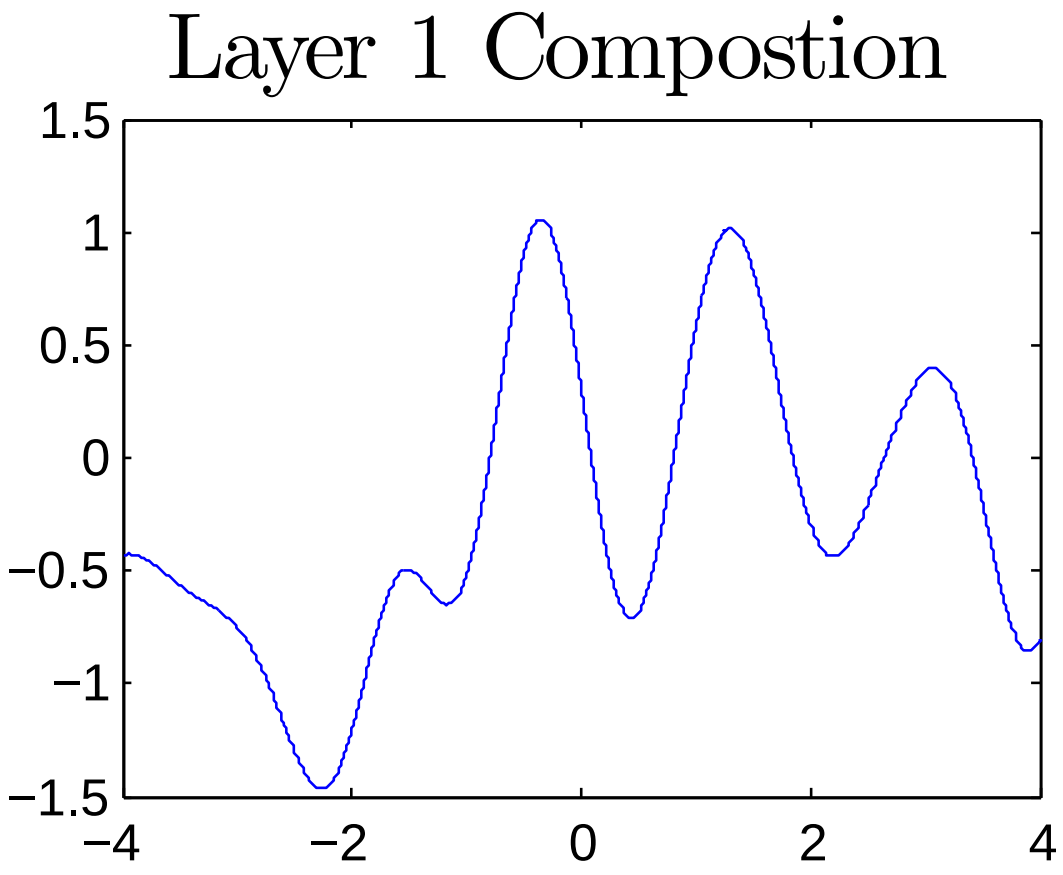
a) standard MLP architecture.



b) Input-connected architecture.

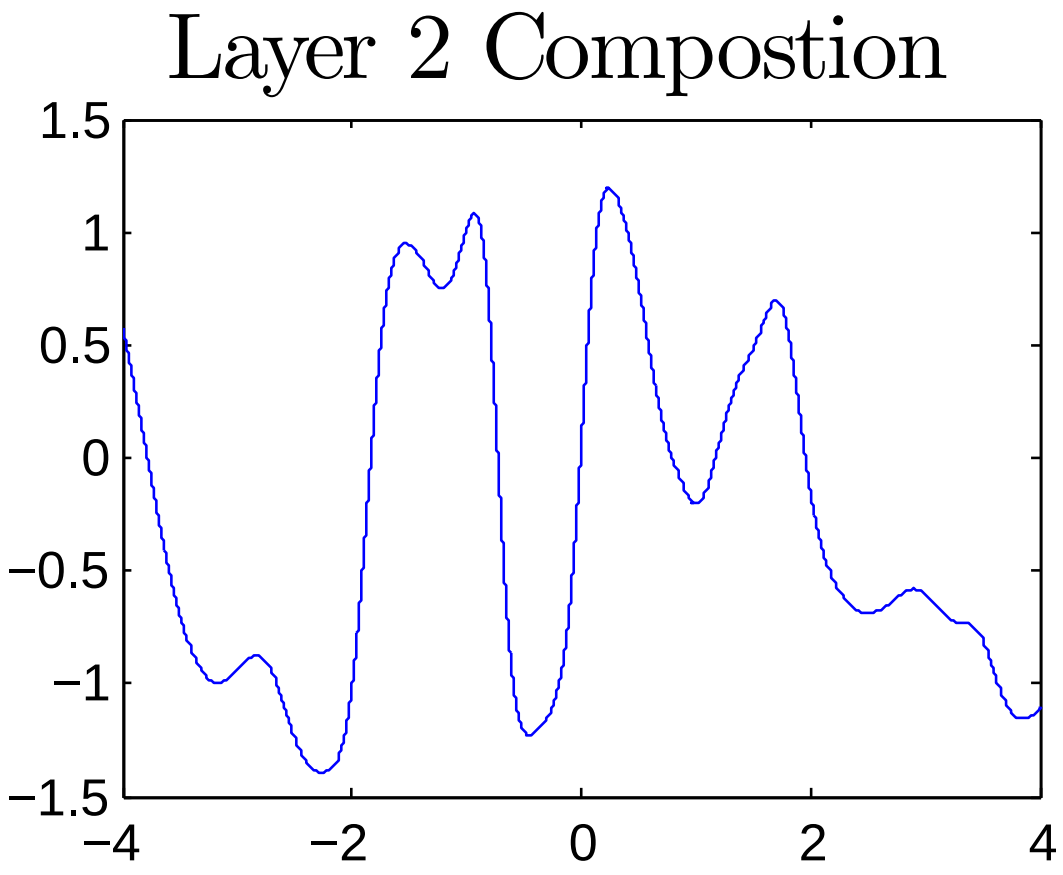
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



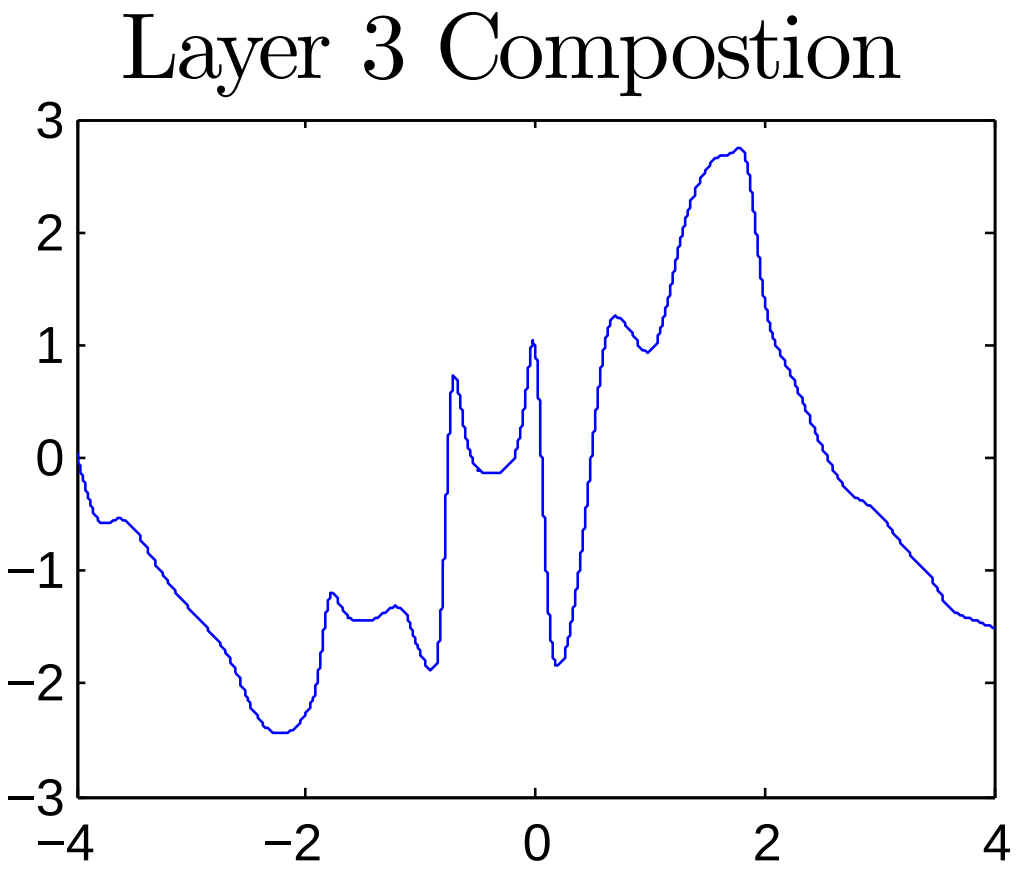
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



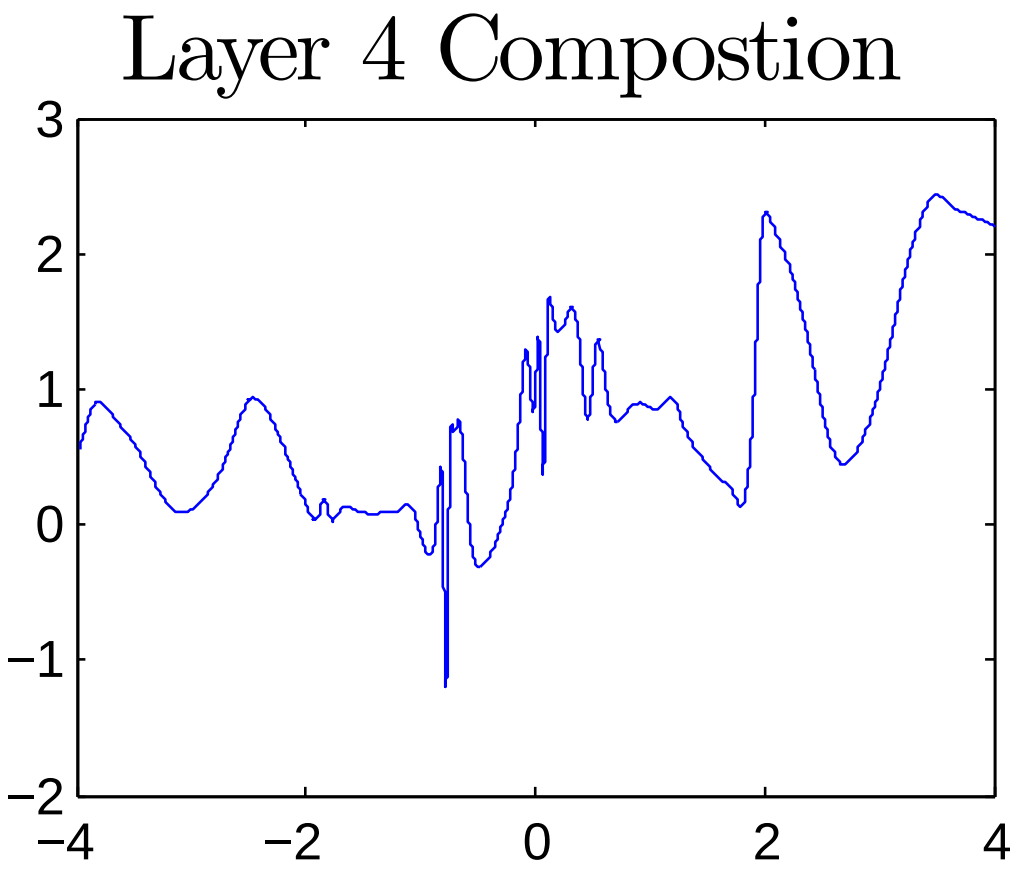
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



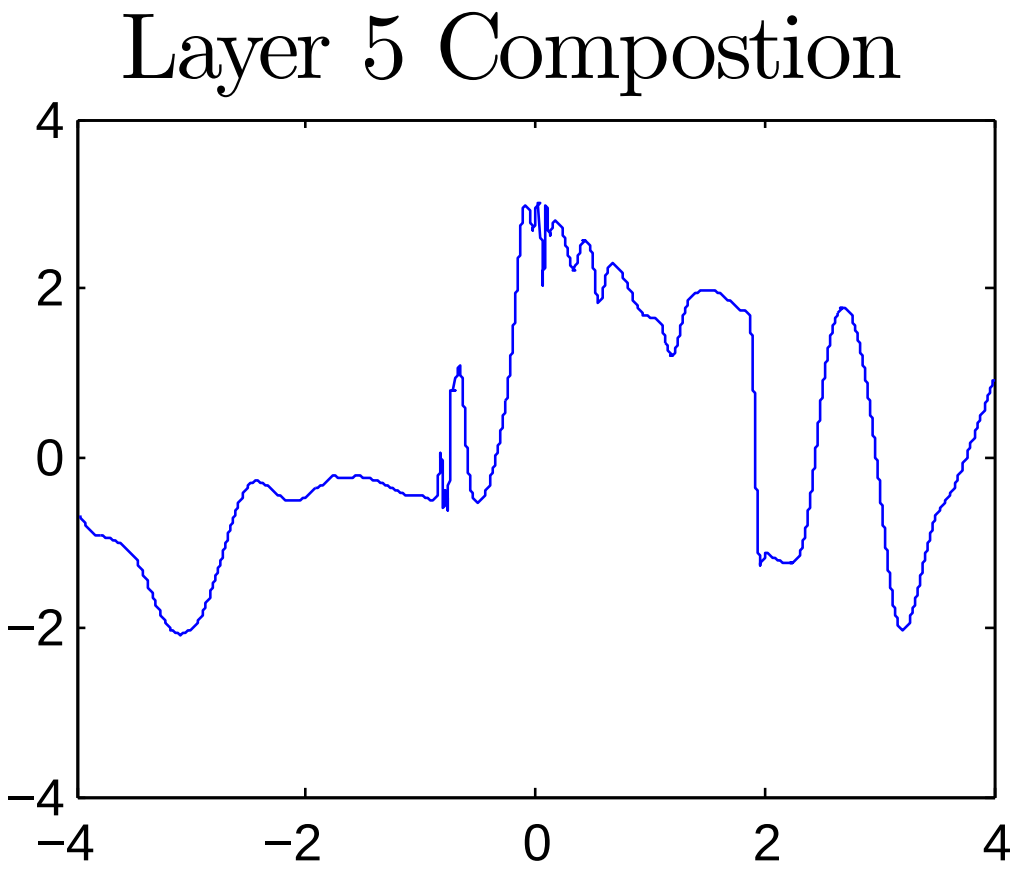
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



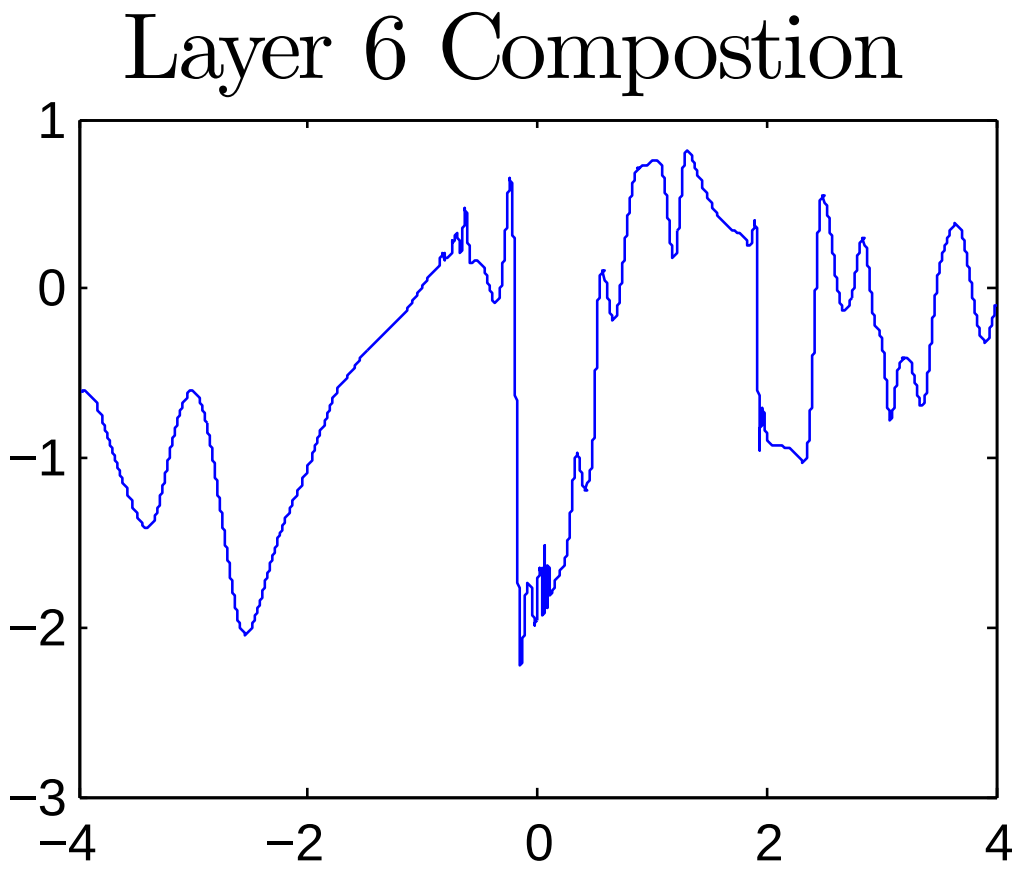
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



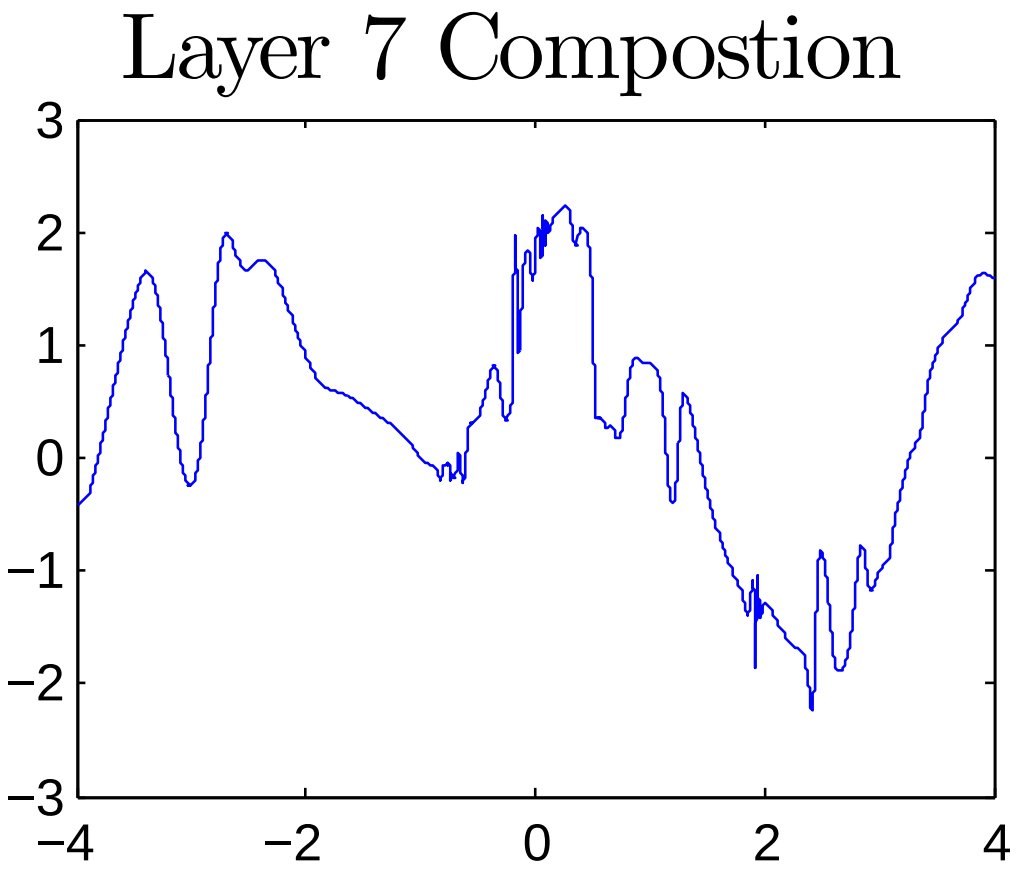
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



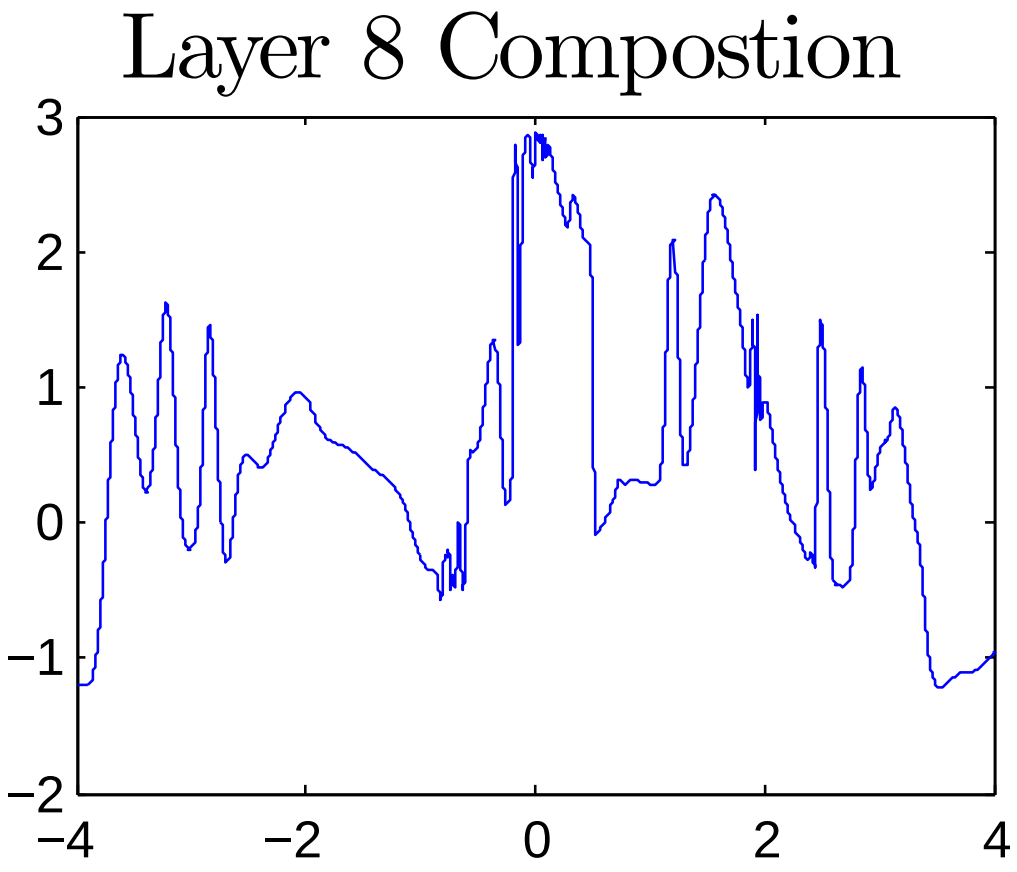
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



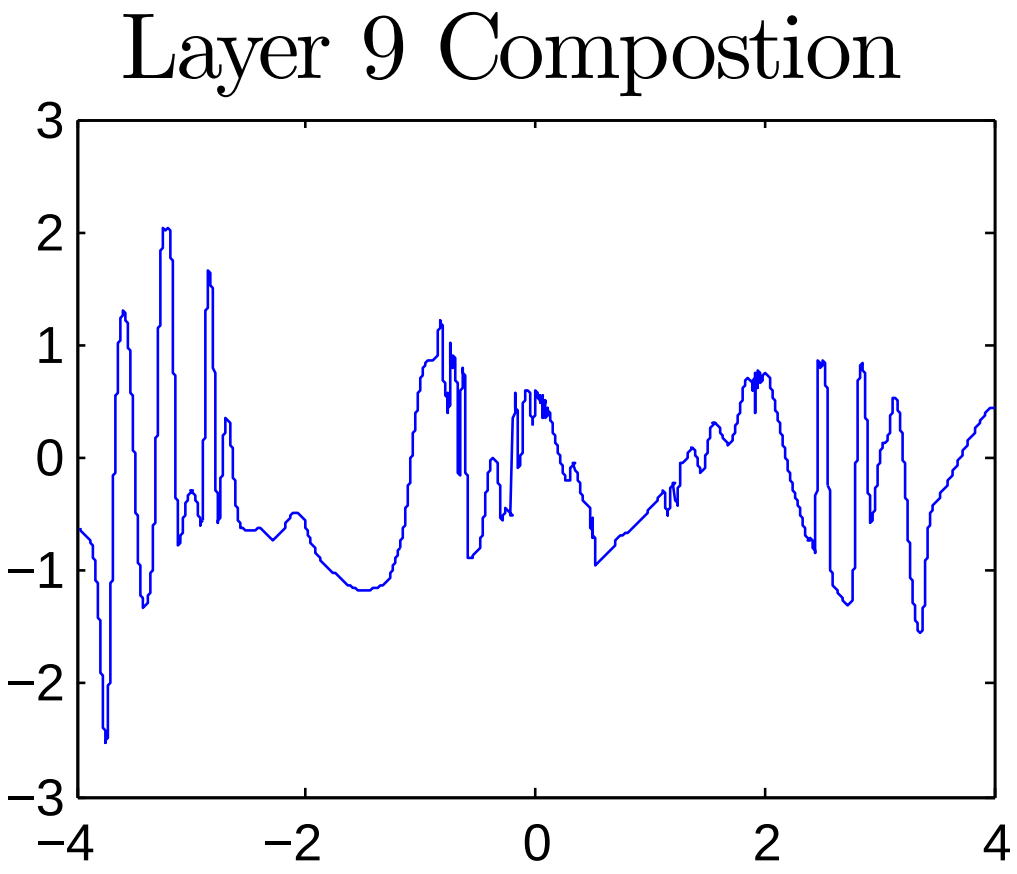
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



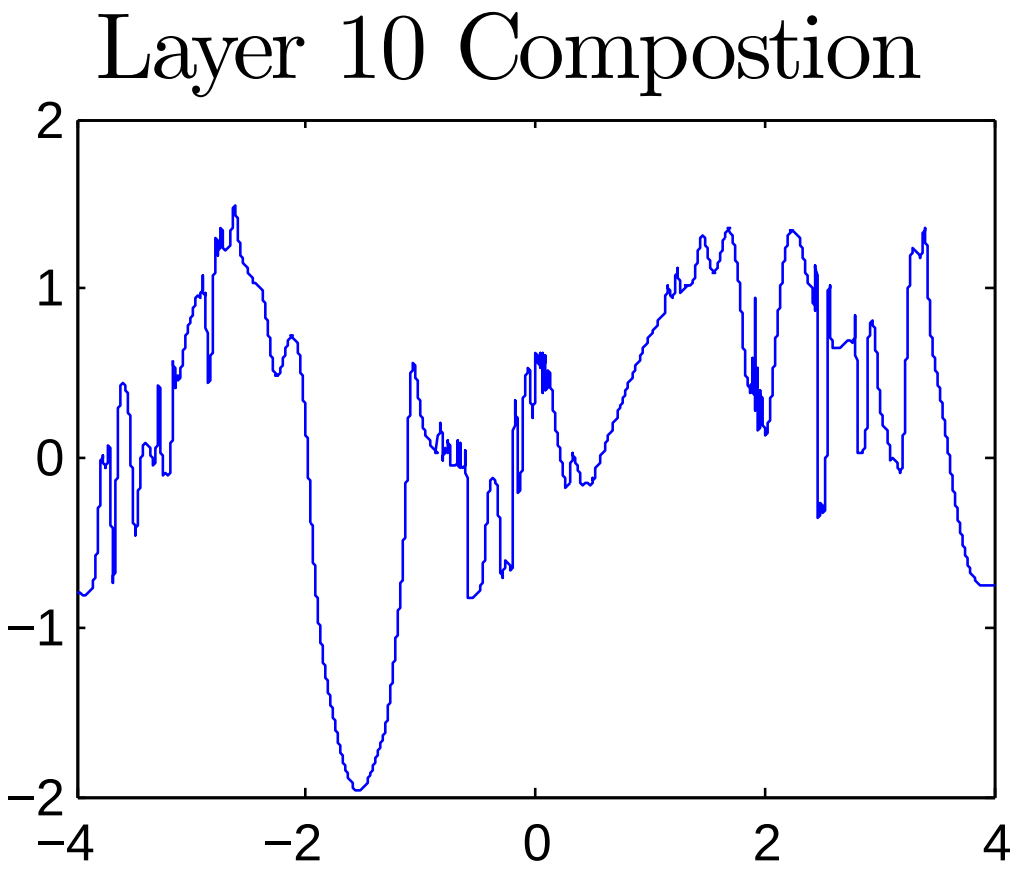
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



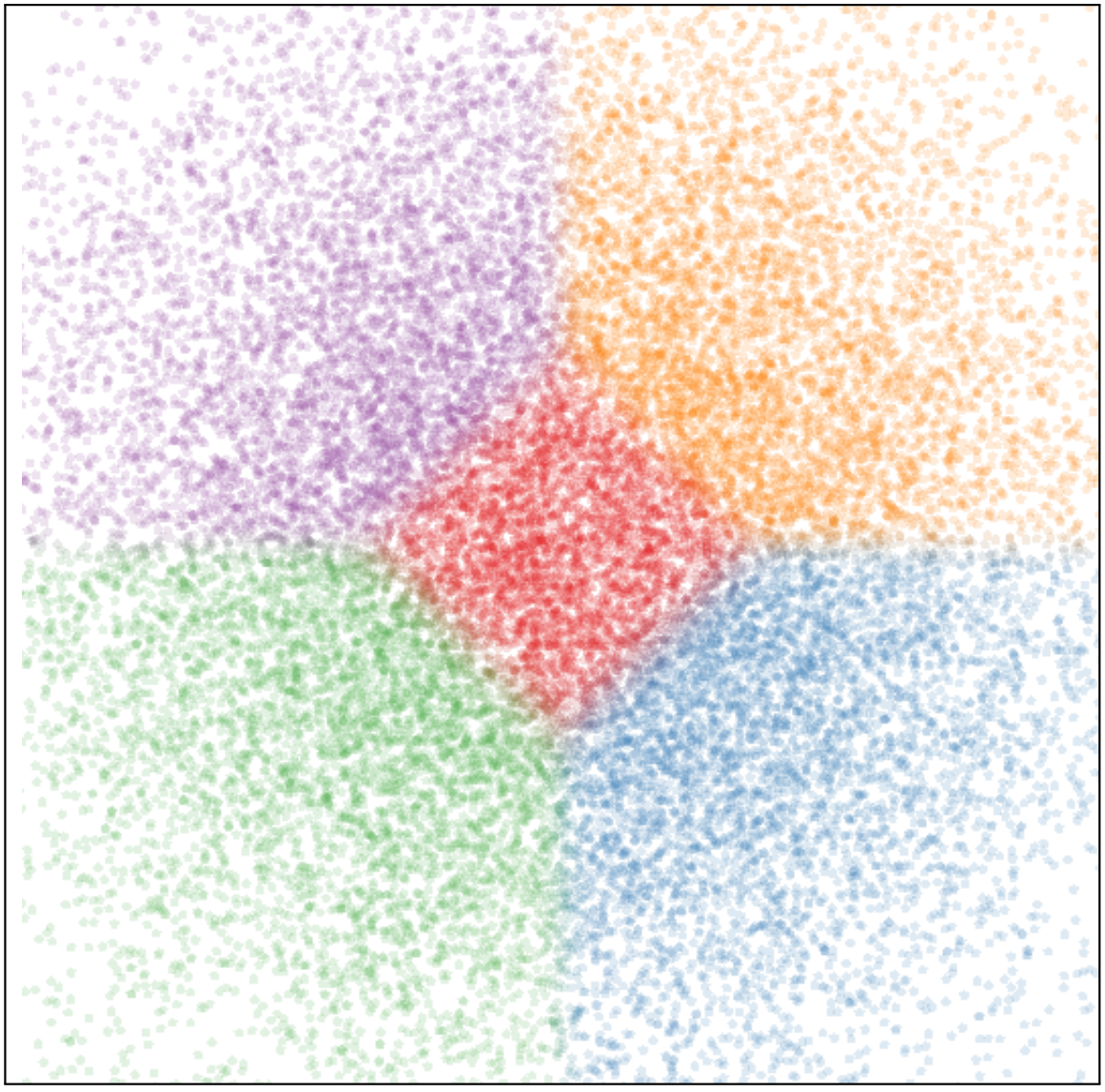
Deep Gaussian Processes

- Draws from input-connected one-dimensional deep GPs:



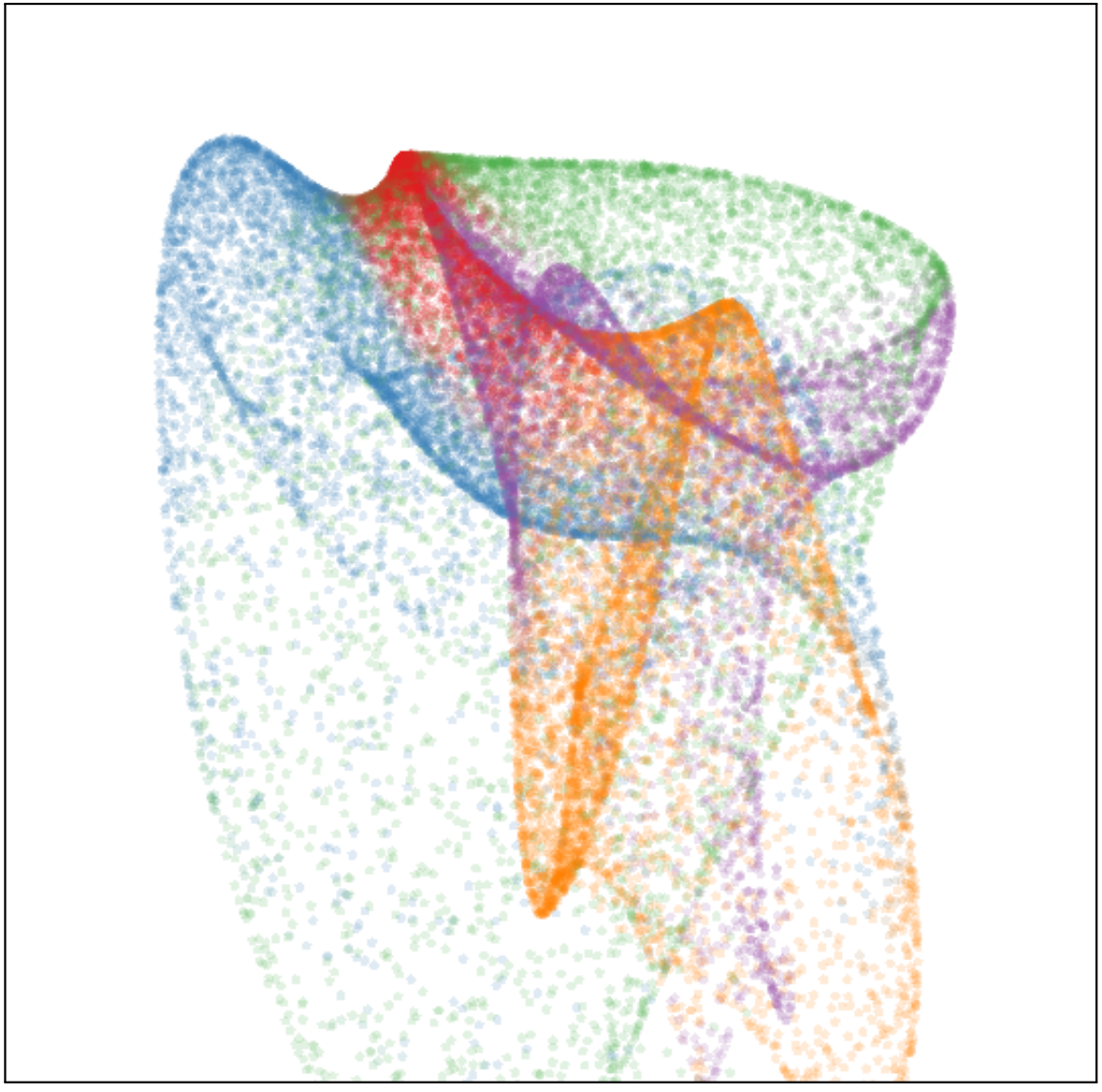
Deep Gaussian Processes

- input-connected 2D to 2D warpings of a Gaussian density:



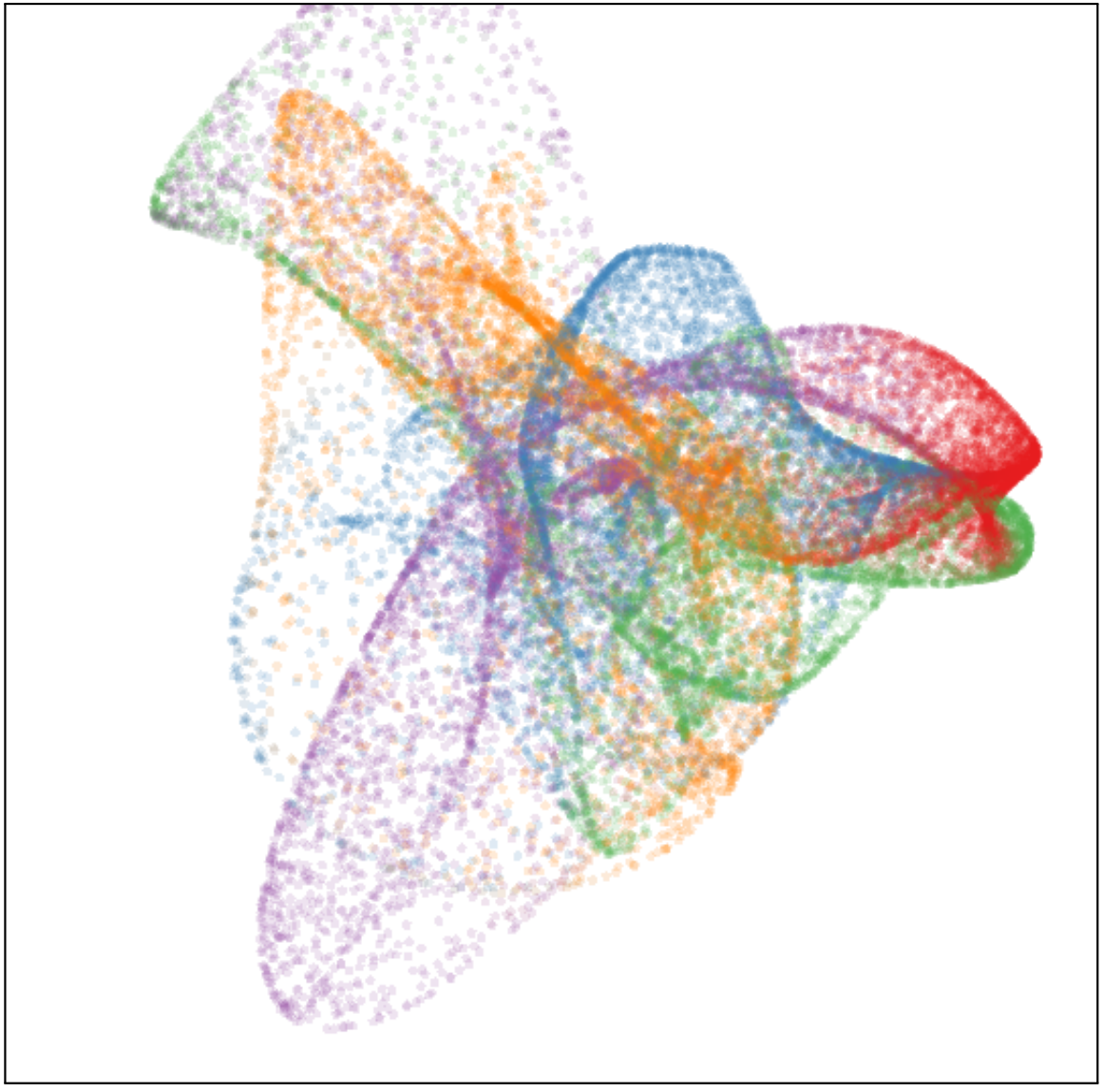
Deep Gaussian Processes

- input-connected 2D to 2D warpings of a Gaussian density:



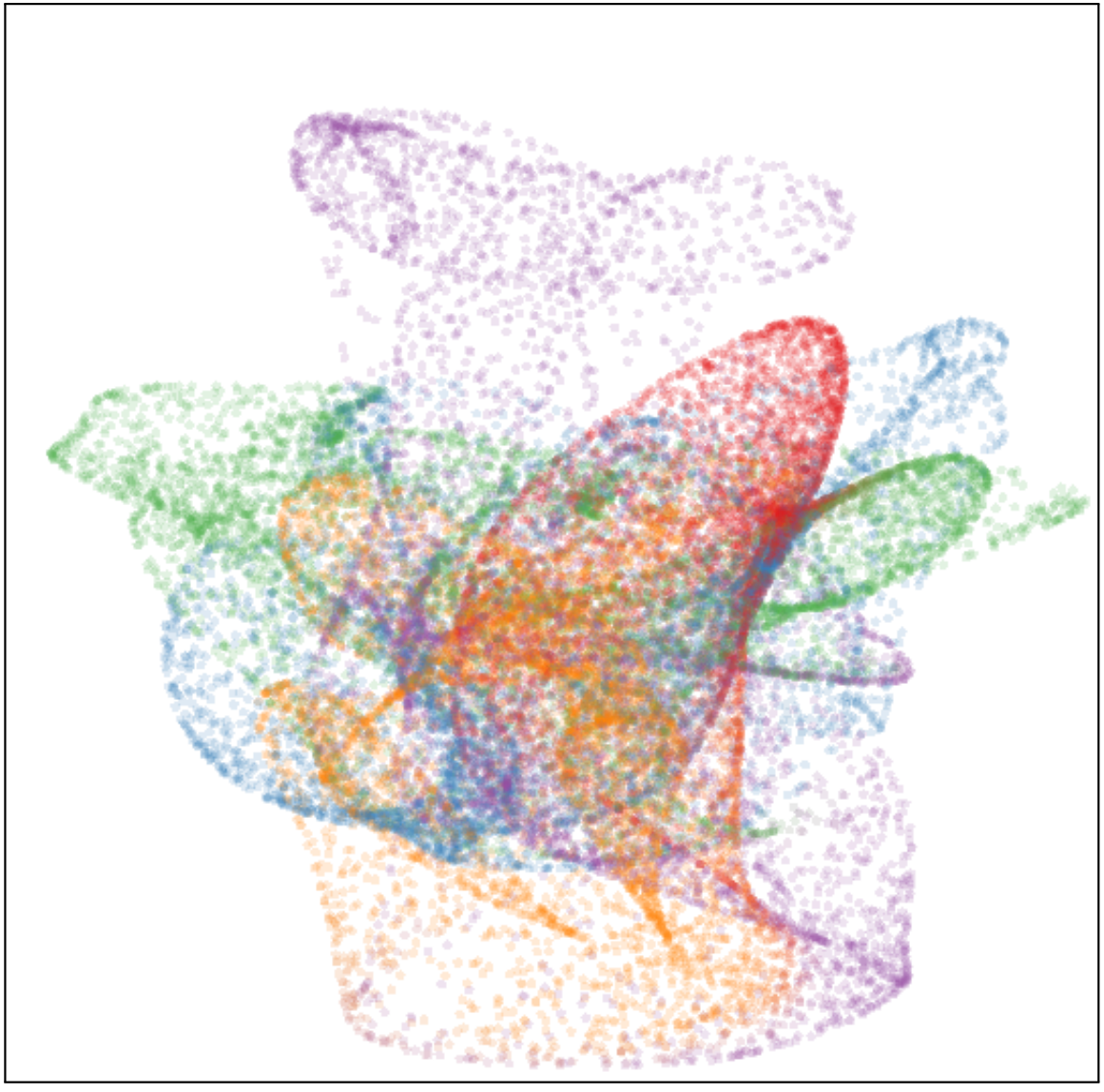
Deep Gaussian Processes

- input-connected 2D to 2D warpings of a Gaussian density:



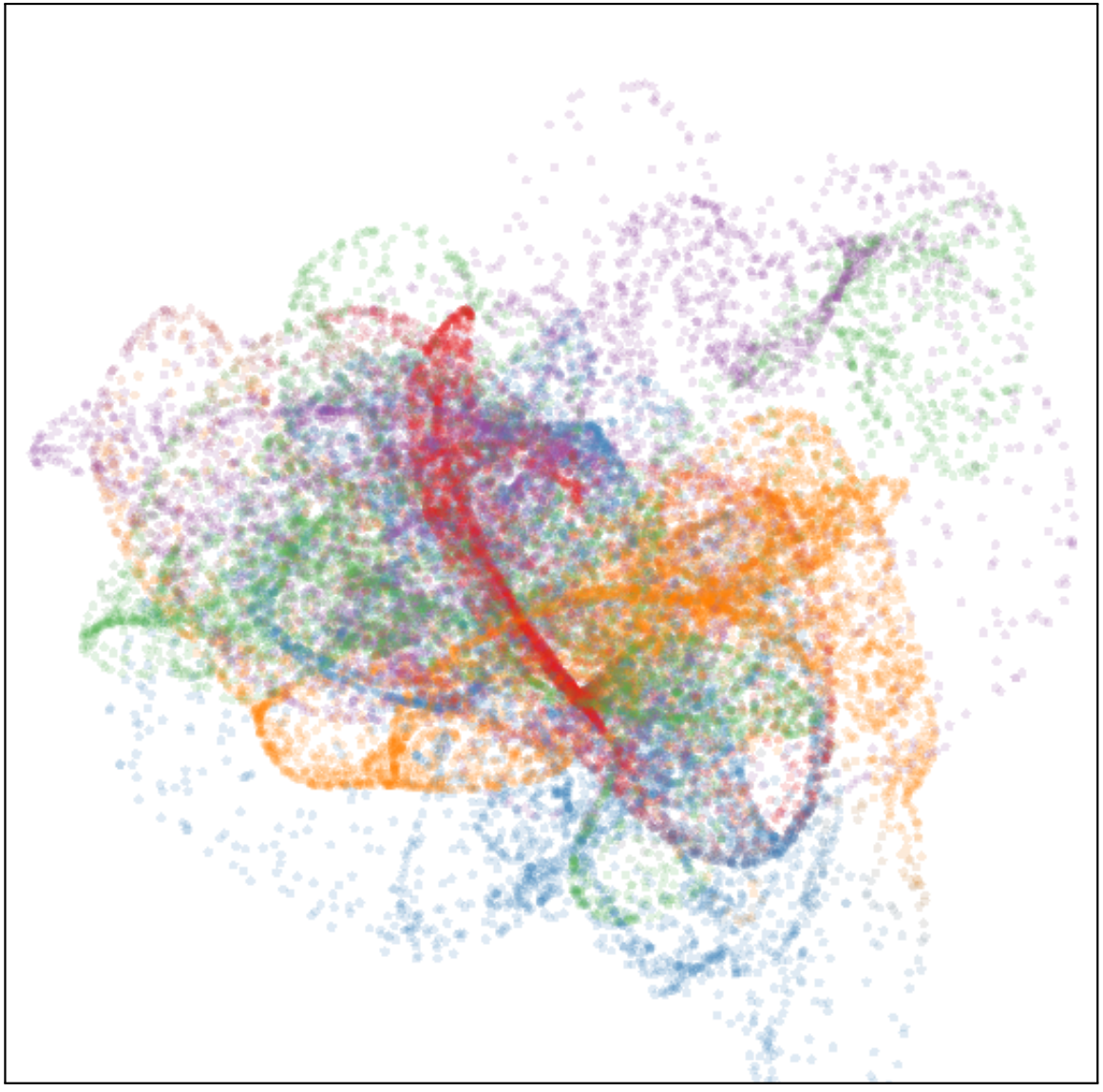
Deep Gaussian Processes

- input-connected 2D to 2D warpings of a Gaussian density:



Deep Gaussian Processes

- input-connected 2D to 2D warpings of a Gaussian density:



Deep Gaussian Processes

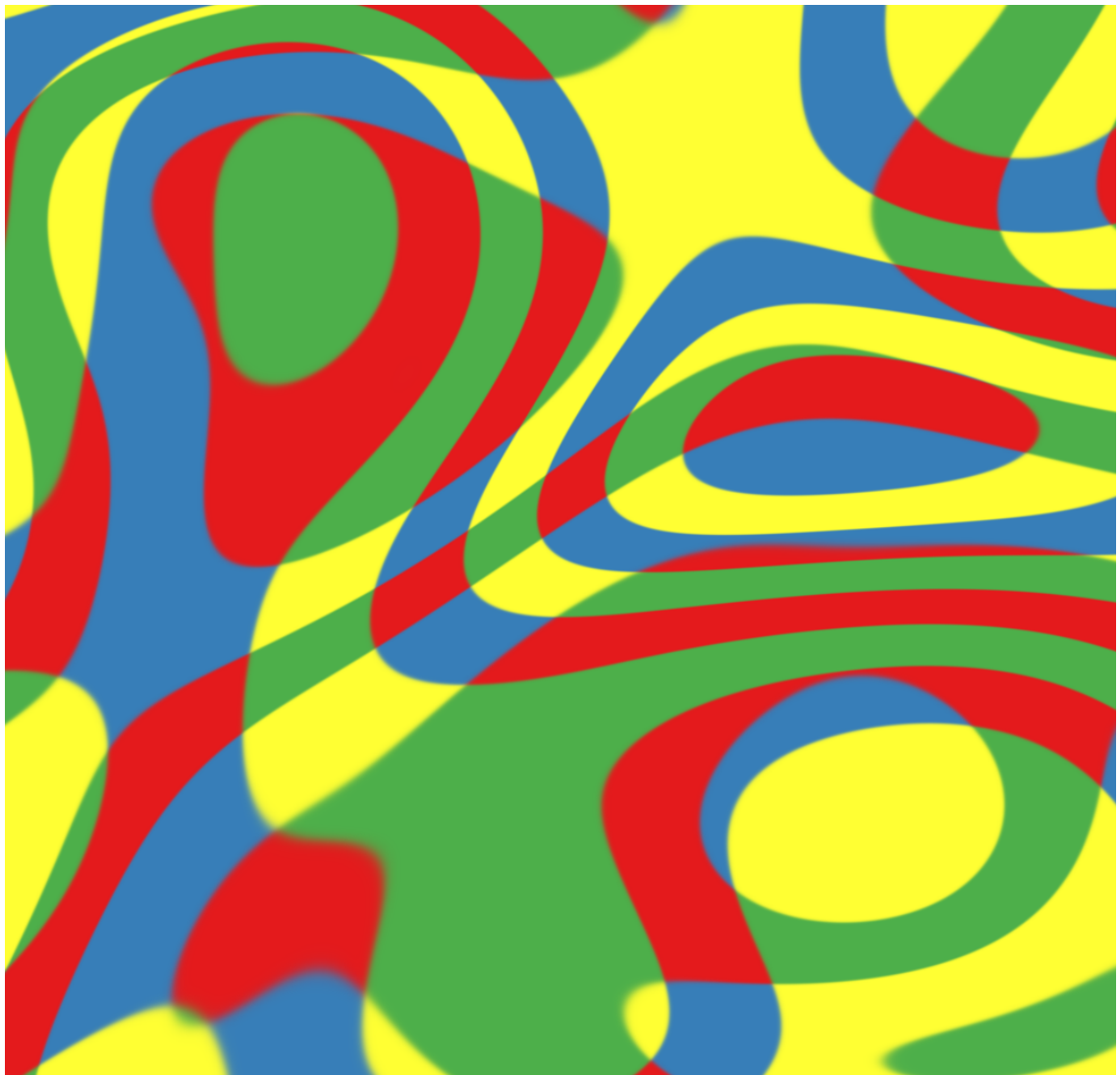
- input-connected 2D to 2D warpings of a Gaussian density:

Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)

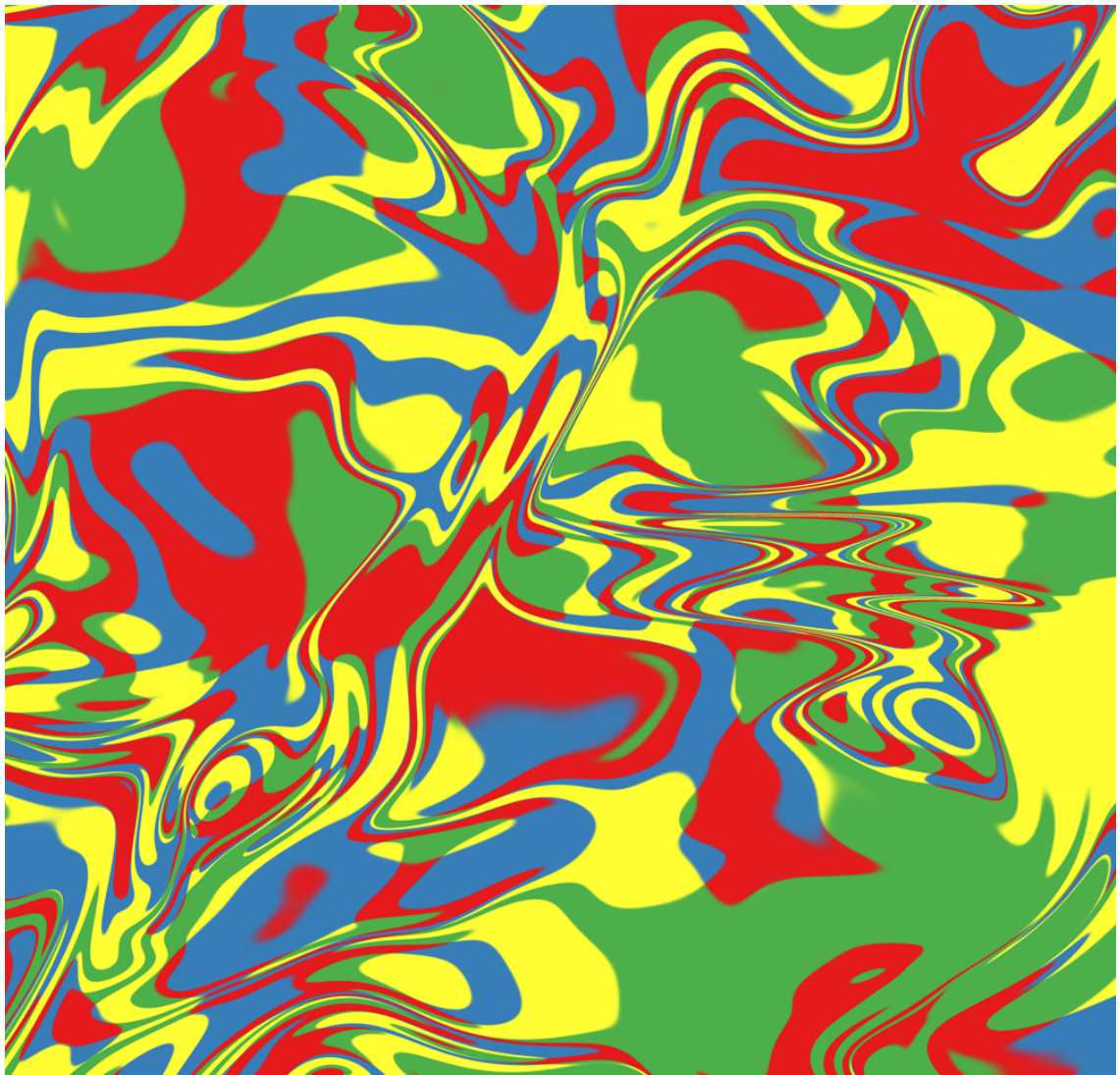
Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)



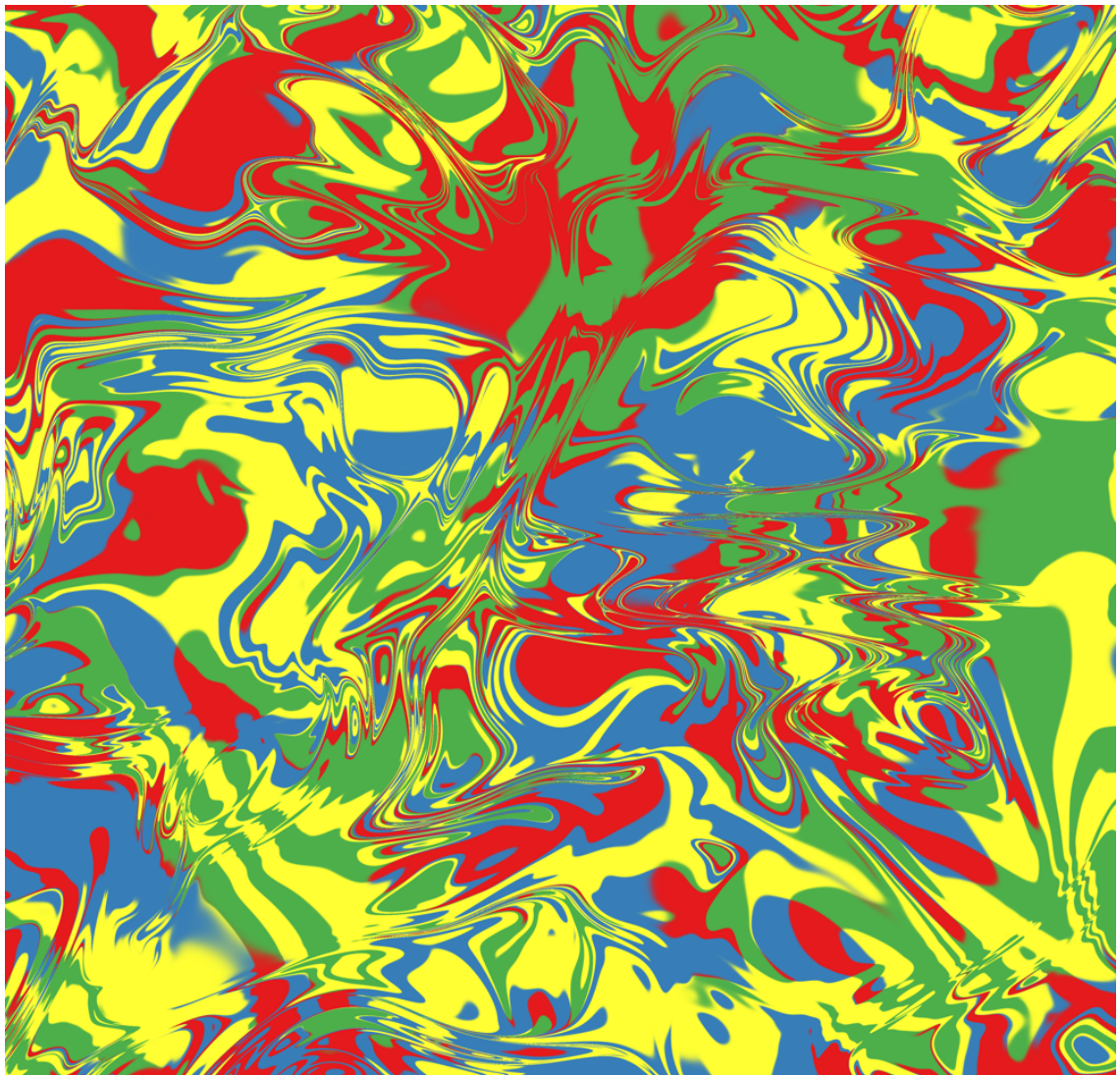
Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)



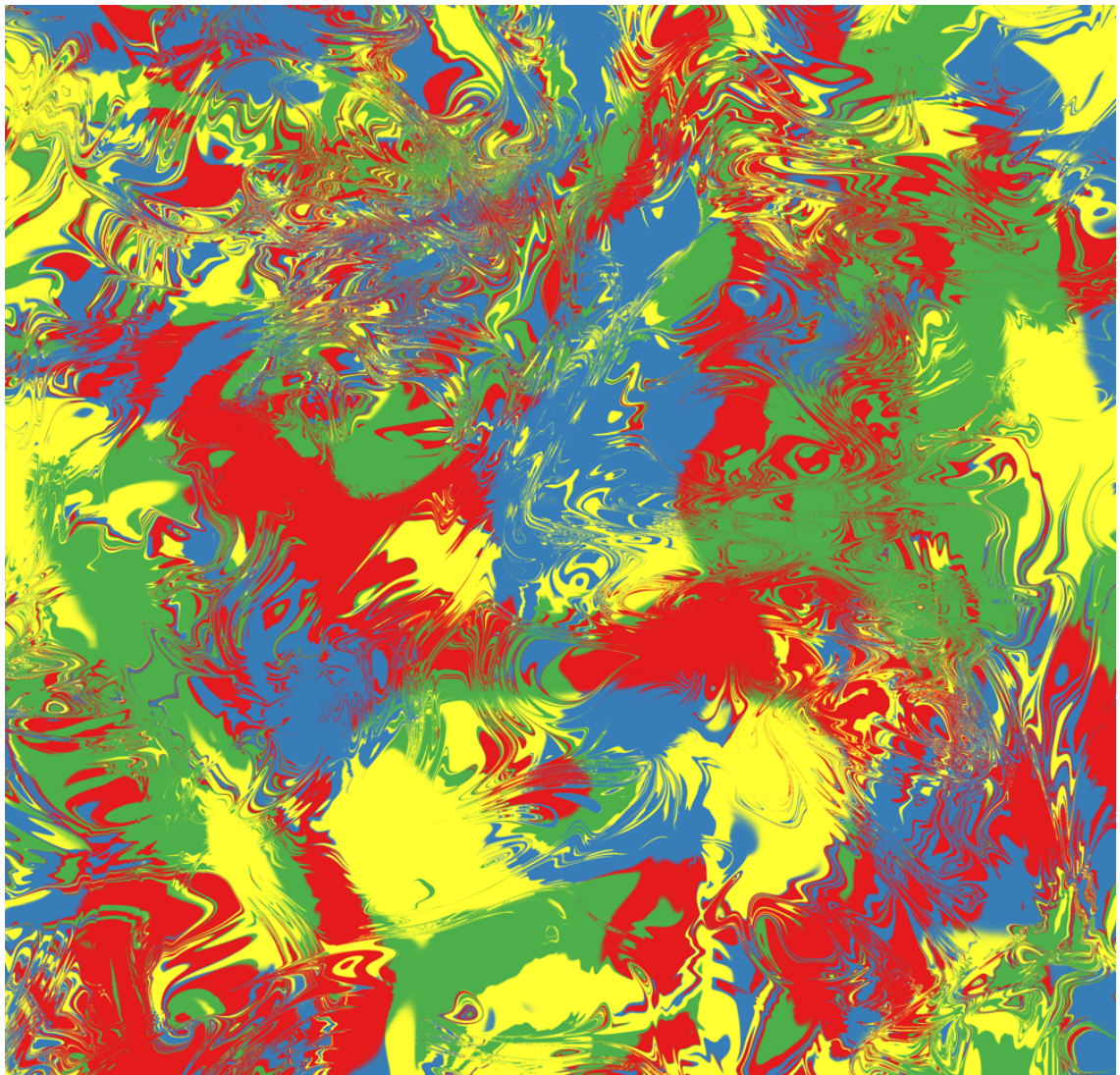
Deep Gaussian Processes

- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)



Deep Gaussian Processes

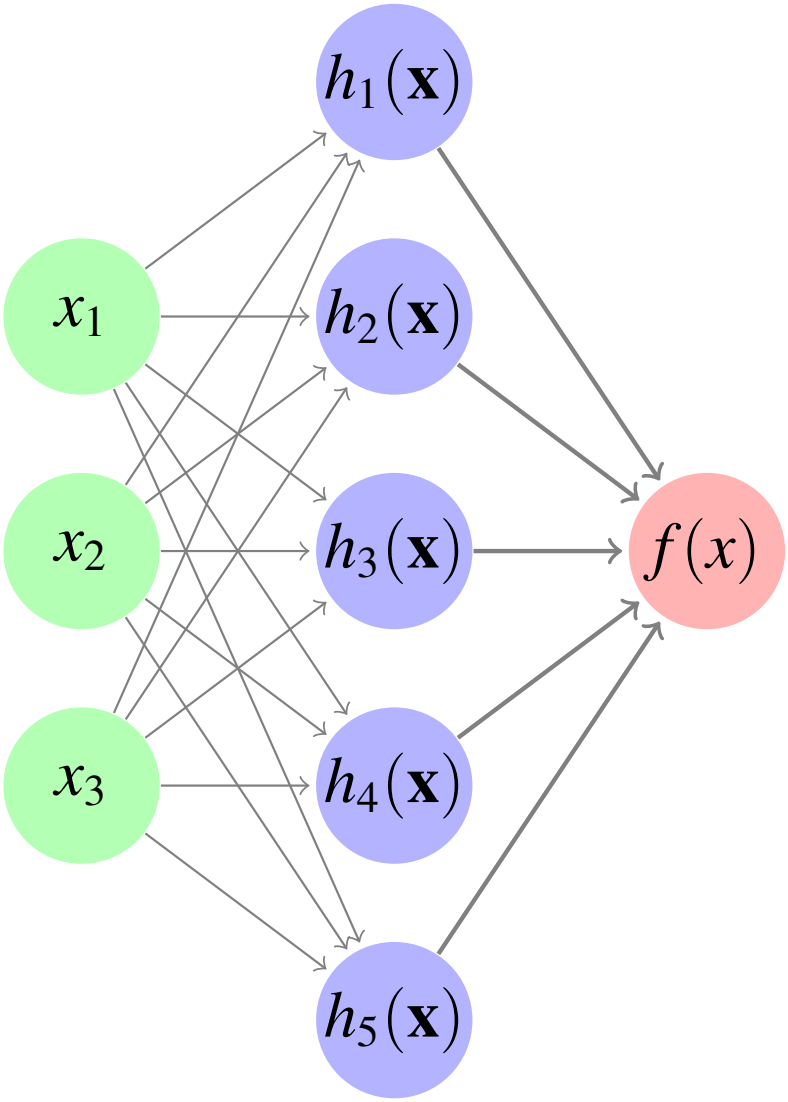
- Showing the x that gave rise to a particular y
- (i.e. decision boundaries)



Dropout

- Dropout is a method for regularizing neural networks (Hinton et al., 2012; Srivastava, 2013).
- Recipe:
  1. randomly set half of feature activations to zero
  2. Double the remaining features activations
  3. Average over all possible ways of dropping out
- Gives robustness, since neurons can't depend on one another.
- What do we get when we do dropout in GPs?

Dropout on Feature Activations



Original formulation:

$$f(\mathbf{x}) = \frac{1}{K} \sum_{i=1}^{K} w_i h_i(\mathbf{x})$$

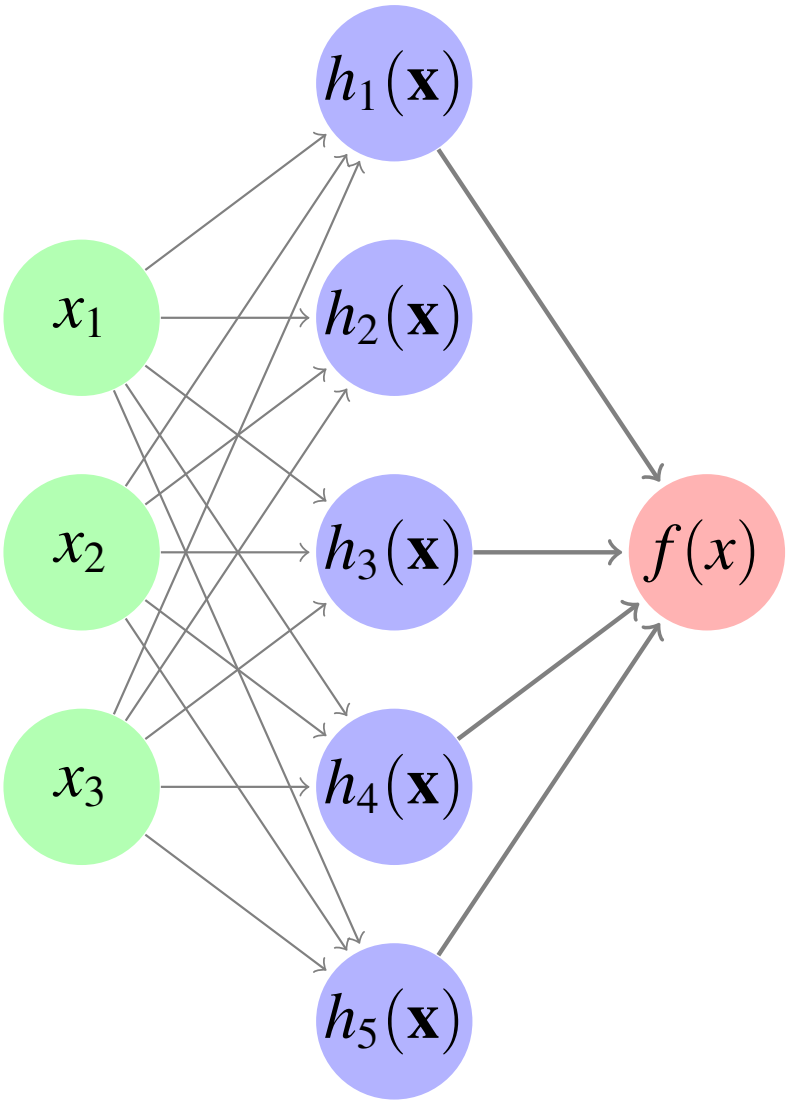
with any weight distribution,

$$\mathbb{E}[w_i] = 0, \quad \mathbb{V}[w_i] = \sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\operatorname{cov} \begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

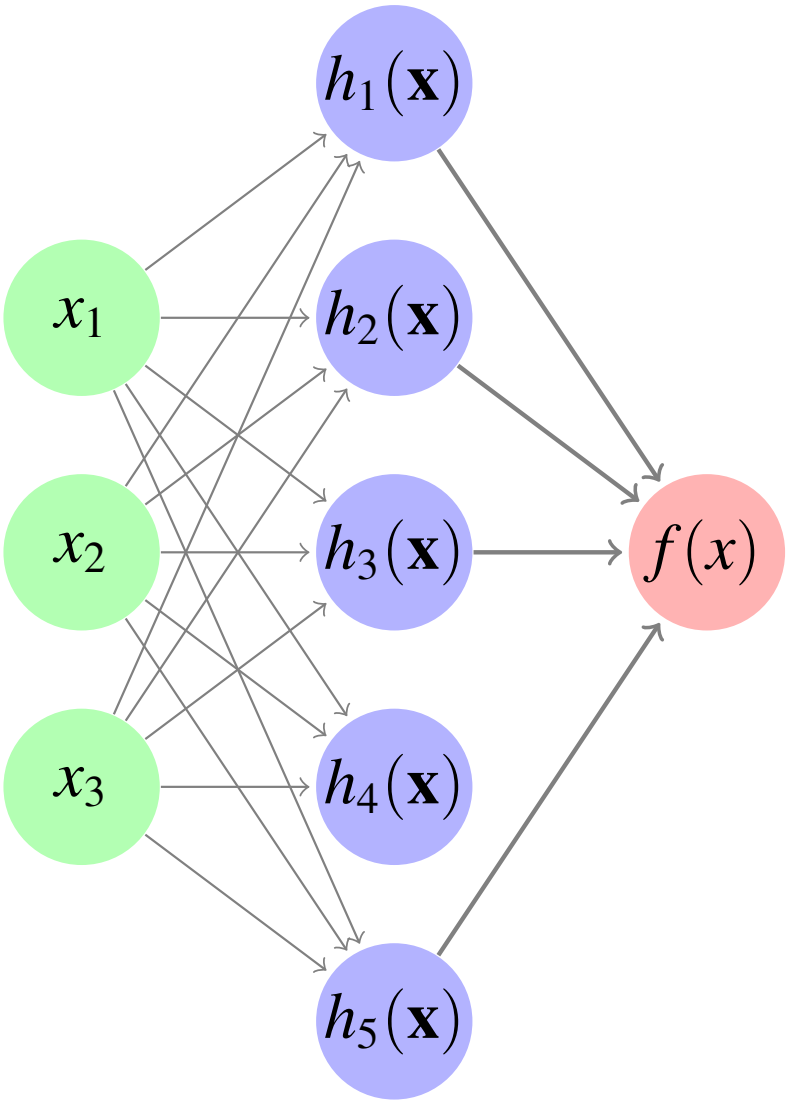
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

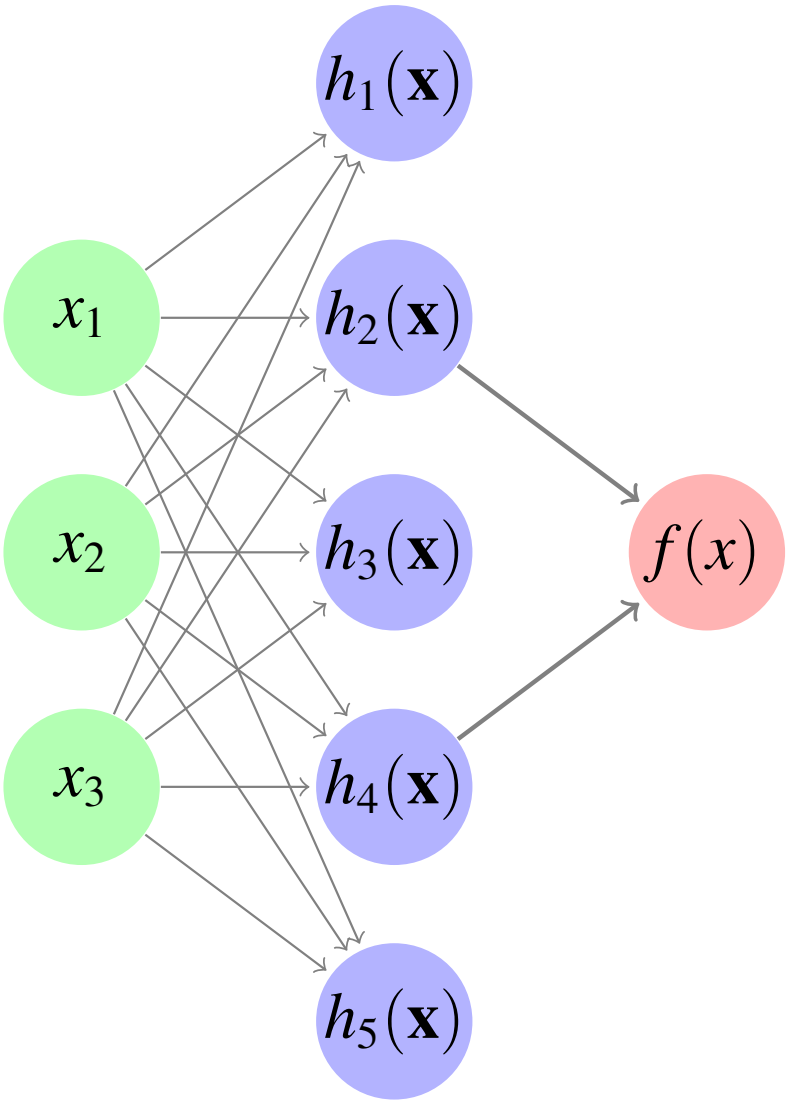
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

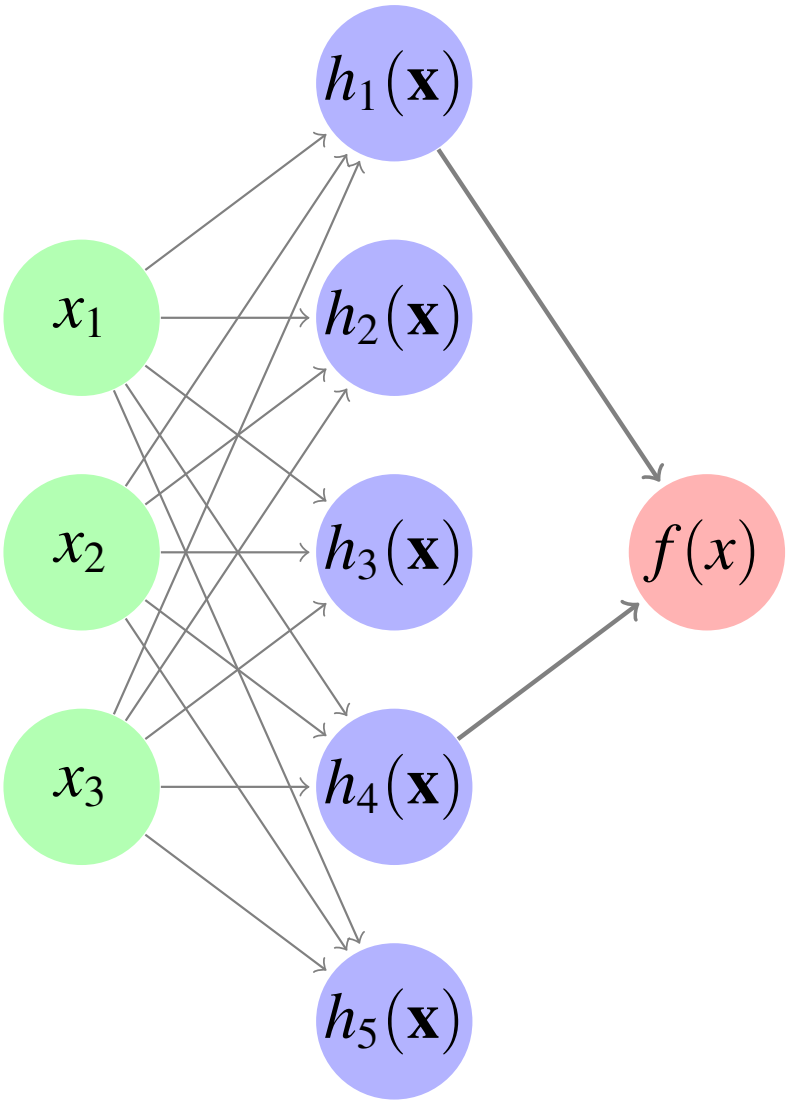
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

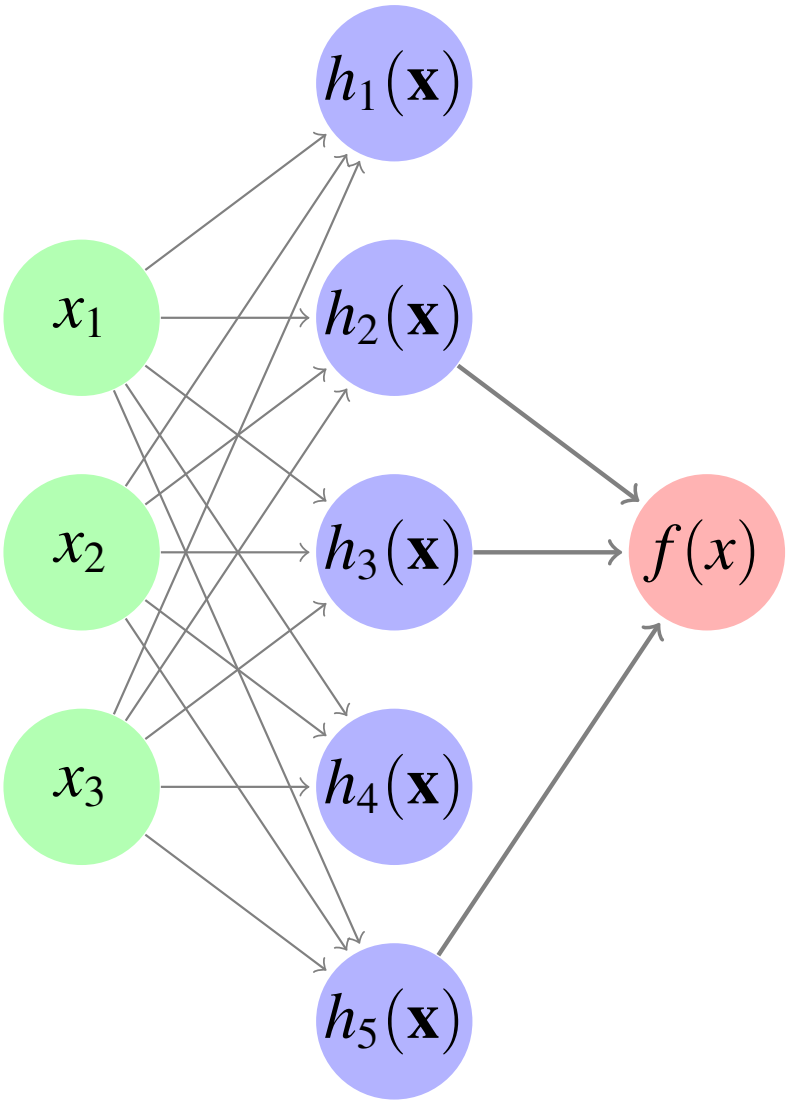
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

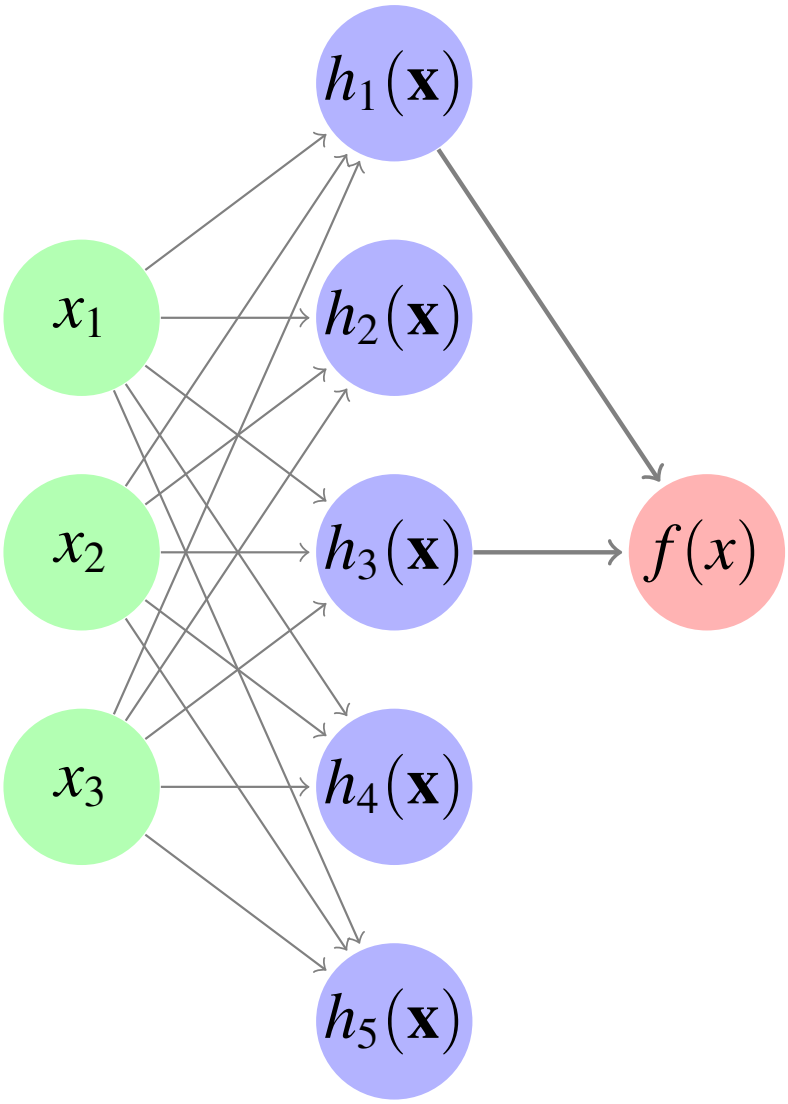
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

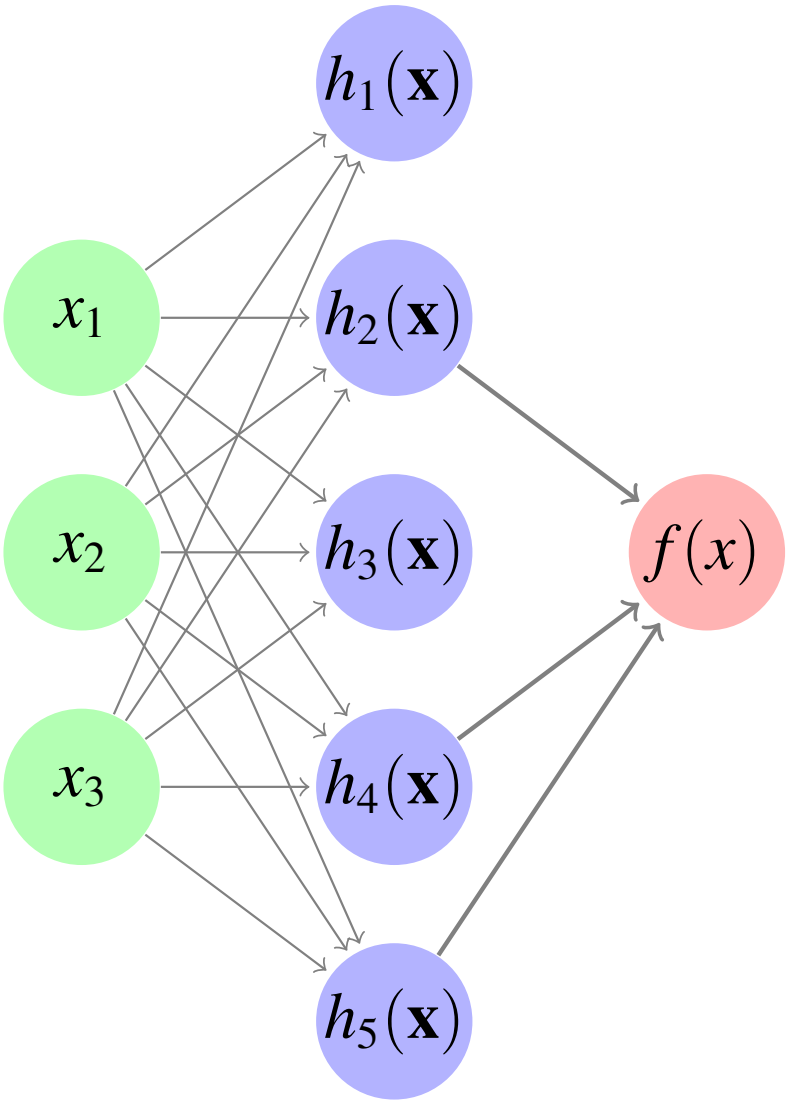
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

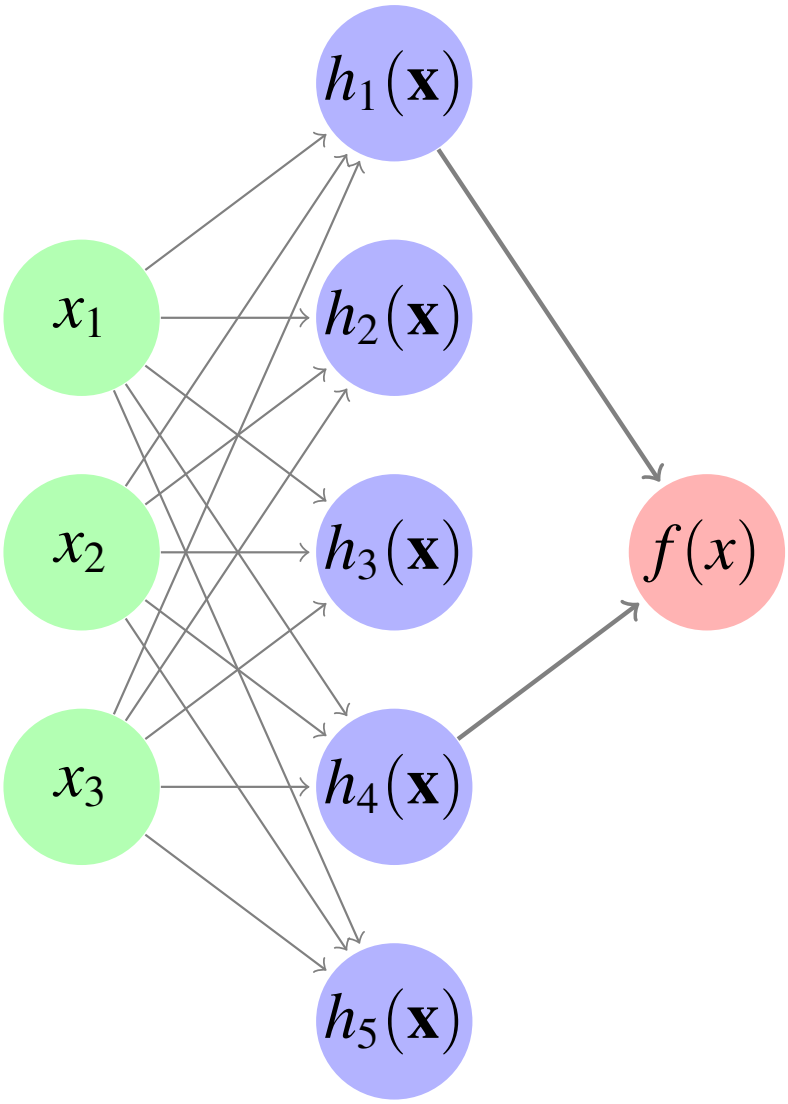
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

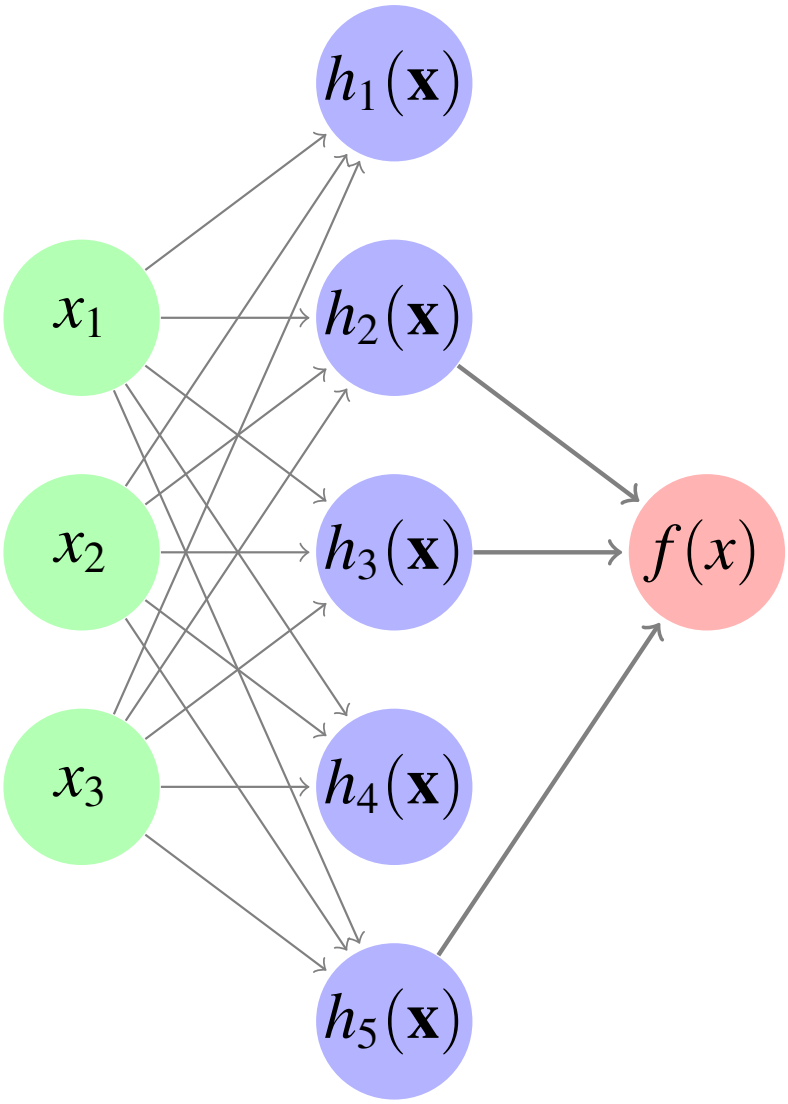
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Remove units with probability ½:

$$f(\mathbf{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

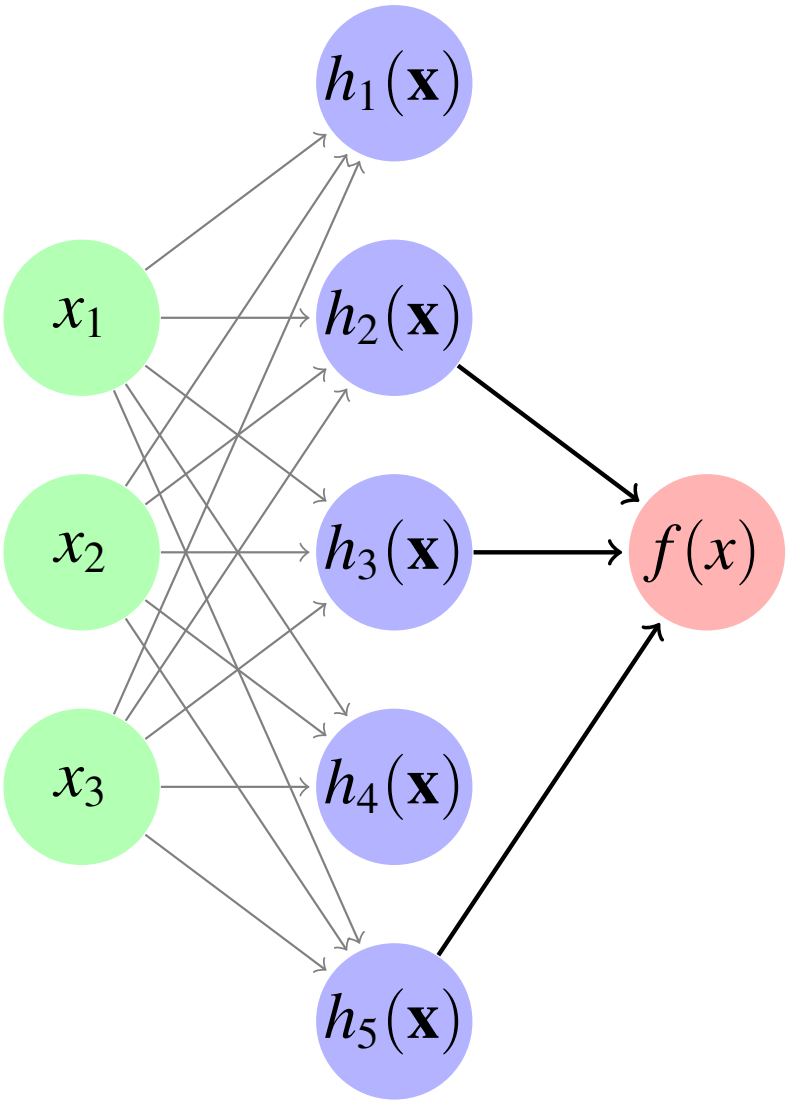
with any weight distribution,

$$\mathbb{E}\left[r_i w_i\right] = 0, \quad \mathbb{V}\left[r_i w_i\right] = \frac{1}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{1}{4}\frac{\sigma^2}{K}\sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations



Double output variance:

$$f(\mathbf{x}) = \frac{2}{K} \sum_{i=1}^{K} r_i w_i h_i(\mathbf{x}) \quad r_i \overset{\text{iid}}{\sim} \text{Ber}(1/2)$$

with any weight distribution,

$$\mathbb{E}\left[2 r_i w_i\right] = 0, \quad \mathbb{V}\left[2 r_i w_i\right] = \frac{4}{4}\sigma^2$$

by CLT, gives a GP as $K \to \infty$

$$\text{cov}\begin{bmatrix} f(\mathbf{x}) \\ f(\mathbf{x}') \end{bmatrix} \to \frac{4}{4}\frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(\mathbf{x}) h_i(\mathbf{x}')$$

Dropout on Feature Activations

- Dropout on feature activations gives same GP
  - Averaging the same model doesn't do anything
- GPs were doing dropout all along? ☺
- GPs strange because any one feature doesn't matter.
- Is there a better way to drop out features that would lead to robustness?

Dropout on Inputs

- Let $k(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)$
- Exact averaging over all dropouts is a mixture of GPs:

$$p(f(\mathbf{x}))) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \text{GP}\left(0, \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d}\right)$$

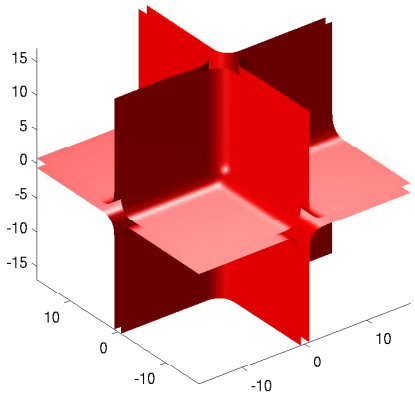
- Probably a nice model, but presumably intractable.
- In neural net literature, exponentially-many dropout models are averaged over in tractable ways.
- For instance, dropout mixture has same covariance as

$$f(\mathbf{x}) \sim \text{GP}\left(0, \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d}\right)$$

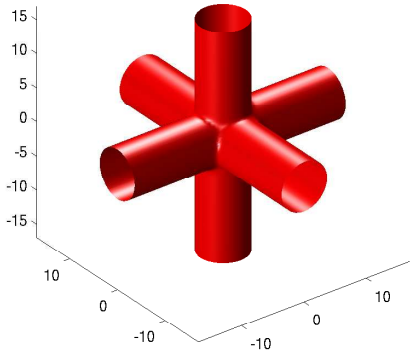
Dropout on Inputs

$$f(\mathbf{x}) \sim \mathrm{GP}\left(0, \sum_{\mathbf{r}\in\{0,1\}^D} \prod_{d=1}^{D} k_d(\mathbf{x}_d, \mathbf{x}'_d)^{r_d}\right)$$

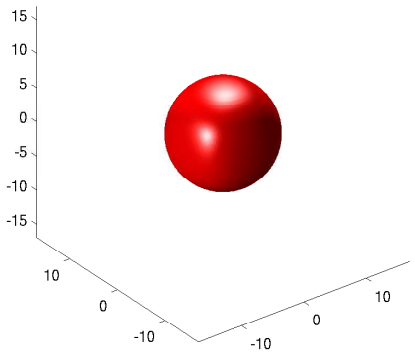
- That's exactly additive GPs! Kernel isocontours look like:



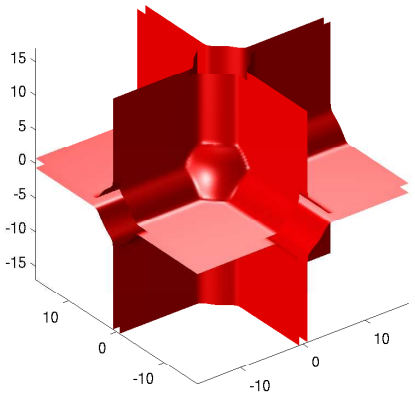
$k_1 + k_2 + k_3$



$k_1k_2 + k_2k_3 + k_1k_3$



$k_1k_2k_3$



all terms

- Can compute all 2^D terms in $\mathcal{O}(D^2)$
- lots of functions, each only depends on some of the inputs

Summary

- At least two different ways to make GPs deep:
  - composing kernels (inference easy, have to specify kernel)
  - composing GPs (inference is hard, but can learn structure)
- Kernel learning is a form of representation learning

- Building priors over functions can shed light on architectures choices or initialization strategies in a data-independent way.
- What sorts of structures do we want to be able to learn?

- We can bring tricks from the neural net literature back to GPs. (still need to try translation-invariant image kernels!)

Summary

- At least two different ways to make GPs deep:
  - composing kernels (inference easy, have to specify kernel)
  - composing GPs (inference is hard, but can learn structure)
- Kernel learning is a form of representation learning

- Building priors over functions can shed light on architectures choices or initialization strategies in a data-independent way.
- What sorts of structures do we want to be able to learn?

- We can bring tricks from the neural net literature back to GPs. (still need to try translation-invariant image kernels!)

Thanks!