Designing neural nets

- Neural nets require lots of design decisions whose implications hard to understand.
- We want to understand them without reference to a specific dataset, loss function, or training method.
- We can analyze different network architectures by looking at nets whose parameters are drawn randomly.
Why look at priors if I’m going to learn everything anyways?

- When using Bayesian neural nets:
  - Can’t learn types of networks having vanishing probability under the prior.

- Even when non-probabilistic:
  - Good prior $\rightarrow$ a good initialization strategy.
  - Good prior $\rightarrow$ a good regularization strategy.
  - Good prior $\rightarrow$ higher fraction of parameters specify reasonable models $\rightarrow$ easier optimization problem.
GPs as Neural Nets

A weighted sum of features,

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

with any weight distribution, \( \mathbb{E}[w_i] = 0, \ \mathbb{V}[w_i] = \sigma^2, \ i.i.d. \)
by CLT, gives a GP as \( K \to \infty \)

\[
\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x)h_i(x')
\]
Kernel learning as feature learning

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

- What if we make the GP neural network deep?
Example deep kernel: Periodic

Now our model is:

\[ h_1(x) = [\sin(x), \cos(x)] \]

we have "deep kernel":

\[ k_2(x, x') = \exp\left(-\frac{1}{2} (h_1(x)) - h_1(x') \right) \]
Deep nets, deep kernels

Now our model is:

\[ f(x) = \frac{1}{K} \sum_{i=1}^{K} w_i h_i^{(2)}(h^{(1)}(x)) \]

\[ = w^T h^{(2)}(h^{(1)}(x)) \]

Instead of

\[ k_1(x, x') = h^{(1)}(x)^T h^{(1)}(x') \]

we have “deep kernel”:

\[ k_2(x, x') \]

\[ = [h^{(2)}(h^{(1)}(x))]^T h^{(2)}(h^{(1)}(x')) \]
Deep Kernels

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

$$k_2(x, x') =$$

$$= \left( h^{SE}(h^1(x)) \right)^T h^{SE}(h^1(x'))$$

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

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

$$= \exp \left( -\frac{1}{2} \left[ k_1(x, x) - 2k_1(x, x') + k_1(x', x') \right] \right)$$

- A closed form... let's do it again!
Repeated Fixed Feature Mappings

\[ h^{(1)}_1 \rightarrow h^{(2)}_1 \rightarrow h^{(3)}_1 \rightarrow h^{(4)}_1 \]

\[ h^{(1)}_2 \rightarrow h^{(2)}_2 \rightarrow h^{(3)}_2 \rightarrow h^{(4)}_2 \]

\[ h^{(1)}_\infty \rightarrow h^{(2)}_\infty \rightarrow h^{(3)}_\infty \rightarrow h^{(4)}_\infty \]

\[ f(x) \]
Infinitely Deep Kernels

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

- $k_\infty(x, x') = 1$ everywhere. 😊
Following a suggestion from Neal (1995), we connect the inputs $x$ to each layer:

**Standard architecture:**

$\textbf{x} \xrightarrow{} f^{(1)}(x) \xrightarrow{} f^{(2)}(x) \xrightarrow{} f^{(3)}(x) \xrightarrow{} f^{(4)}(x)$

**Input-connected architecture:**

$\textbf{x} \xrightarrow{} f^{(1)}(x) \xrightarrow{} f^{(2)}(x) \xrightarrow{} f^{(3)}(x) \xrightarrow{} f^{(4)}(x)$
A simple fix

\[k_{L+1}(\mathbf{x}, \mathbf{x}') =
\]
\[= \exp \left( -\frac{1}{2} \| \begin{bmatrix} \mathbf{h}^L(\mathbf{x}) \\ \mathbf{x} \end{bmatrix} - \begin{bmatrix} \mathbf{h}^L(\mathbf{x}') \\ \mathbf{x}' \end{bmatrix} \|_2^2 \right) \]
\[= \exp \left( -\frac{1}{2} \left[ k_L(\mathbf{x}, \mathbf{x}) - 2k_L(\mathbf{x}, \mathbf{x}') + k_L(\mathbf{x}', \mathbf{x}') \right] - \frac{1}{2} \| \mathbf{x} - \mathbf{x}' \|_2^2 \right)\]
Infinitely deep kernels, take two

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

- Like an Ornstein-Uhlenbeck process with skinny tails
- Samples are non-differentiable (fractal).
Not very exciting...

- Fixed feature mapping, unlikely to be useful for anything
- Power of neural nets comes from learning a custom representation.
Deep Gaussian Processes

- A prior over compositions of functions:

\[ f^{(1:L)}(x) = f^{(L)}(f^{(L-1)}(\ldots f^{(2)}(f^{(1)}(x)) \ldots)) \]  

with each \( f^{(\ell)} \) \( \text{ind} \sim \mathcal{GP}(0, k^{\ell}_d(x, x')) \).

- Can be seen as a “simpler” version of Bayesian neural nets

- Two equivalent architectures.
Deep GPs as nonparametric nets

- A neural net where each neuron’s activation function is drawn from a Gaussian process prior.
- Avoids problem of unit saturation (with sigmoidal units).
- Each draw from neural net prior gives a function \( y = f(x) \).
- In this talk we only consider noiseless functions.
Deep GPs as infinitely wide parametric nets

- Infinitely-wide fixed feature maps alternating with finite linear information bottlenecks:

\[
\mathbf{h}^{(\ell)}(\mathbf{x}) = \sigma \left( \mathbf{b}^{(\ell)} + \left[ \mathbf{V}^{(\ell)} \mathbf{W}^{(\ell-1)} \right] \mathbf{h}^{(\ell-1)}(\mathbf{x}) \right)
\]
Priors on deep networks

- A draw from a one-neuron-per-layer deep GP:
A draw from a one-neuron-per-layer deep GP:
Priors on deep networks

- A draw from a one-neuron-per-layer deep GP:
A draw from a one-neuron-per-layer deep GP:
Priors on deep networks

- A draw from a one-neuron-per-layer deep GP:

\[ f(x) \]

5 Layers
A draw from a one-neuron-per-layer deep GP:

\[ f(x) \]

6 Layers
Priors on deep networks

- A draw from a one-neuron-per-layer deep GP:
Priors on deep networks

- A draw from a one-neuron-per-layer deep GP:

\[ f(x) \]

8 Layers
A draw from a one-neuron-per-layer deep GP:
Priors on deep networks

- A draw from a one-neuron-per-layer deep GP:

Size of derivative becomes log-normal distributed.
Priors on deep networks

- 2D to 2D warpings of a set of coloured points:
Priors on deep networks

- 2D to 2D warpings of a set of coloured points:
Priors on deep networks

- 2D to 2D warpings of a set of coloured points:

![Diagram showing 2D to 2D warpings of a set of coloured points with 2 Layers]
Priors on deep networks

- 2D to 2D warpings of a set of coloured points:

3 Layers
Priors on deep networks

- 2D to 2D warpings of a set of coloured points:
Priors on deep networks

- 2D to 2D warpings of a set of coloured points:

Density concentrates along filaments.
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)

No warping
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)

2 Layers
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)

3 Layers
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)

4 Layers
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)

5 Layers
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)
Priors on deep networks

Color shows $y$ that each $x$ is mapped to (decision boundary)

Representation only changes in one direction locally.
What makes a good representation?

- Good representations of data manifolds don’t change in directions orthogonal to the manifold. ([Rifai et. al. 2011](#))
- Good representations also change in directions tangent to the manifold, to preserve information.
- Representation of a $D$-dimensional manifold should change in $D$ orthogonal directions, locally.
- Our prior on functions might be too restrictive.
**Analysis of Jacobian**

The distribution of normalized singular values of the Jacobian of functions drawn from a 5-dimensional deep GP prior.

- Lemma from paper: The Jacobian of a deep GP is a product of i.i.d. random Gaussian matrices.
- Output only changes in w.r.t. one direction as net deepens.
A simple fix

Following a suggestion from Neal (1995), we connect the inputs $x$ to each layer:

**Standard architecture:**

```
   x → f^{(1)}(x) → f^{(2)}(x) → f^{(3)}(x) → f^{(4)}(x)
```

**Input-connected architecture:**

```
   x → f^{(1)}(x) → f^{(2)}(x) → f^{(3)}(x) → f^{(4)}(x)
```
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:

\[ f(x) \]
A different architecture

A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:

\[ f(x) \]
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:

\[ f(x) \]

3 layers
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:

\[ f(x) \]

8 layers
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:

\[ f(x) \]

9 layers
A different architecture

- A draw from a one-neuron-per-layer deep GP, with the input also connected to each layer:

Greater variety of derivatives.
A different architecture

- Input-connected 2D to 2D warpings of coloured points:
A different architecture

- Input-connected 2D to 2D warpings of coloured points:

[Image of coloured points with text: 1 Layer]
A different architecture

- Input-connected 2D to 2D warpings of coloured points:
A different architecture

- Input-connected 2D to 2D warpings of coloured points:
A different architecture

- Input-connected 2D to 2D warpings of coloured points:
A different architecture

- Input-connected 2D to 2D warpings of coloured points:

Density becomes more complex but remains 2D.
A different architecture (show video)

- Color shows \( y \) that each \( x \) is mapped to

No warping
A different architecture (show video)

- Color shows $y$ that each $x$ is mapped to

2 Layers
A different architecture (show video)

- Color shows that each $x$ is mapped to

10 Layers
A different architecture (show video)

- Color shows $y$ that each $x$ is mapped to

20 Layers
A different architecture (show video)

- Color shows that each \( x \) is mapped to

Representation sometimes depends on all directions.
Understanding dropout

- Dropout is a method for regularizing neural networks (Hinton et al., 2012; Srivastava, 2013).

- Recipe:
  1. Randomly set to zero (drop out) some neuron activations.
  2. Average over all possible ways of doing this.

- Gives robustness since neurons can’t depend on each other.

- How does dropout affect priors on functions?

- Related work: (Baldi and Sadowski, 2013; Cho, 2013; Wager, Wang and Liang, 2013)
Dropout on Feature Activations

Original formulation:

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

with any weight distribution,

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

by CLT, gives a GP as \( K \to \infty \)

\[ \text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x)h_i(x') \]
Dropout on Feature Activations

Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

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

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

$$\text{cov} \begin{bmatrix} f(x) \\ f(x') \end{bmatrix} \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x) h_i(x')$$
Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

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

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

$$\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x) h_i(x')$$
Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

$$\mathbb{E}[r_i w_i] = 0, \quad \forall [r_i w_i] = \frac{1}{2} \sigma^2$$

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

$$\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x)h_i(x')$$
Dropout on Feature Activations

Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

$$\mathbb{E}[r_i w_i] = 0, \quad \forall [r_i w_i] = \frac{1}{2} \sigma^2$$

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

$$\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x) h_i(x')$$
Dropout on Feature Activations

Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

$$\mathbb{E} [r_i w_i] = 0, \quad \forall [r_i w_i] = \frac{1}{2} \sigma^2$$

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

$$\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \rightarrow \frac{1}{2} \sigma^2 \frac{1}{K} \sum_{i=1}^{K} h_i(x)h_i(x')$$
Dropout on Feature Activations

Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

$$\mathbb{E}[r_iw_i] = 0, \quad \forall [r_iw_i] = \frac{1}{2}\sigma^2$$

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

$$\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x)h_i(x')$$
Dropout on Feature Activations

Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

$$\mathbb{E}[r_i w_i] = 0, \quad \forall [r_i w_i] = \frac{1}{2}\sigma^2$$

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

$$\text{cov} \left[ f(x), f(x') \right] \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x)h_i(x')$$
Dropout on Feature Activations

Remove units with probability \( \frac{1}{2} \):

\[
f(x) = \frac{1}{K} \sum_{i=1}^{K} r_i w_i h_i(x) \quad r_i \sim \text{iid Ber}(\frac{1}{2})
\]

with any weight distribution,

\[
\mathbb{E} [r_i w_i] = 0, \quad \forall [r_i w_i] = \frac{1}{2} \sigma^2
\]

by CLT, gives a GP as \( K \to \infty \)

\[
\text{cov} \begin{bmatrix} f(x) \\ f(x') \end{bmatrix} \rightarrow \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x) h_i(x')
\]
Dropout on Feature Activations

Remove units with probability $\frac{1}{2}$:

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

with any weight distribution,

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

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

$$\text{cov} \left[ \begin{array}{c} f(x) \\ f(x') \end{array} \right] \to \frac{1}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x) h_i(x')$$
Dropout on Feature Activations

Double output variance:

\[ f(x) = \frac{2}{K} \sum_{i=1}^{K} r_i w_i h_i(x) \quad r_i \sim \text{iid Ber}(\frac{1}{2}) \]

with any weight distribution,

\[ \mathbb{E}[\sqrt{2}r_i w_i] = 0, \quad \forall \left[\sqrt{2}r_i w_i\right] = \frac{2}{2} \sigma^2 \]

by CLT, gives a GP as \( K \to \infty \)

\[ \text{cov} \left[ f(x), f(x') \right] \to \frac{2}{2} \frac{\sigma^2}{K} \sum_{i=1}^{K} h_i(x)h_i(x') \]
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 are strange because any one feature doesn’t matter.
- Is there a better way to drop out features that would lead to robustness?
Dropout on GP inputs

Each function only depends on some input dimensions.

Given prior covariance \( \text{cov} [f(x), f(x')] = k(x, x') \), exact dropout gives a mixture of GPs:

\[
p(f(x)) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \text{GP} \left( 0, k(\mathbf{r}^T \mathbf{x}, \mathbf{r}^T \mathbf{x}') \right)
\]

Can be viewed as spike-and-slab ARD prior.
Dropout on GP inputs

Each function only depends on some input dimensions.

Given prior covariance \( \text{cov} [f(x), f(x')] = k(x, x') \), exact dropout gives a mixture of GPs:

\[
p(f(x)) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0,1\}^D} \text{GP} \left( 0, k(\mathbf{r}^T \mathbf{x}, \mathbf{r}^T \mathbf{x}') \right)
\]

Can be viewed as spike-and-slab ARD prior.
Dropout on GP inputs

Each function only depends on some input dimensions.

Given prior covariance $\text{cov} [f(\mathbf{x}), f(\mathbf{x}')] = k(\mathbf{x}, \mathbf{x}')$, exact dropout gives a mixture of GPs:

$$p(f(\mathbf{x})) = \frac{1}{2^D} \sum_{\mathbf{r} \in \{0, 1\}^D} \text{GP}(0, k(\mathbf{r}^T \mathbf{x}, \mathbf{r}^T \mathbf{x}'))$$

Can be viewed as spike-and-slab ARD prior.
Covariance before and after dropout

Original squared-exp:
\[
\text{cov} [f(x), f(x')] = k(x, x')
\]

After dropout:
\[
\text{cov} [f(x), f(x')] = \sum_{r \in \{0, 1\}^D} k(r^T x, r^T x')
\]

- Sum of many functions, each depends only on a subset of inputs.
- Output similar even if some input dimensions change a lot.
Summary

- Priors on functions can shed light on design choices in a data-independent way.
- Example 1: Increasing depth makes net outputs change in fewer input directions.
- Example 2: Dropout makes output similar even if some inputs change a lot.
- What sorts of structures do we want to be able to learn?
Priors on functions can shed light on design choices in a data-independent way.

Example 1: Increasing depth makes net outputs change in fewer input directions.

Example 2: Dropout makes output similar even if some inputs change a lot.

What sorts of structures do we want to be able to learn?

Thanks!