Deep kernel process and machines

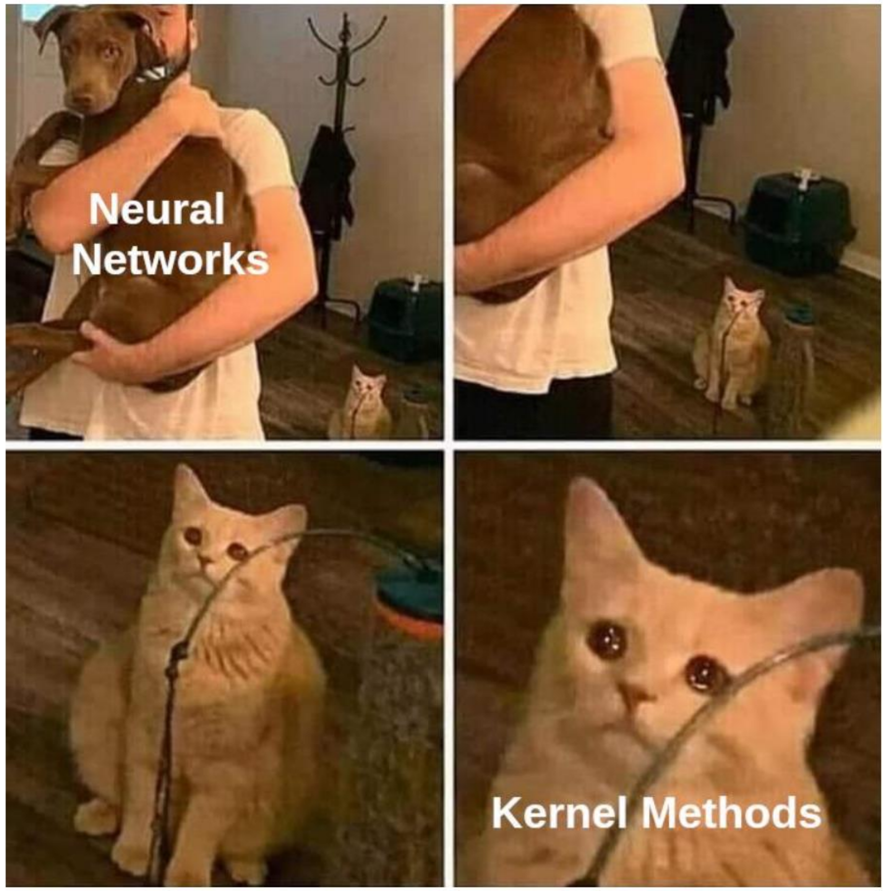
Laurence Aitchison

Sebastian Ober

Adam Yang

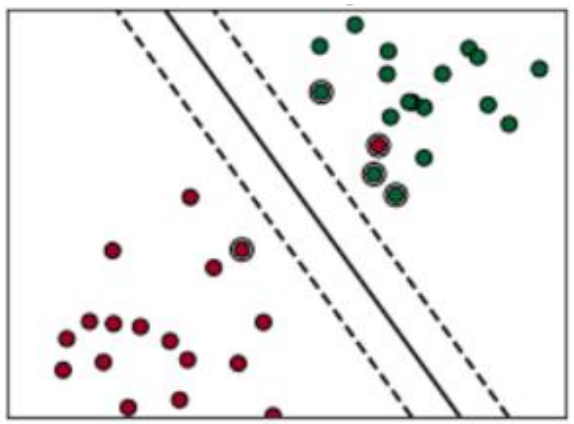
Edward Milsom

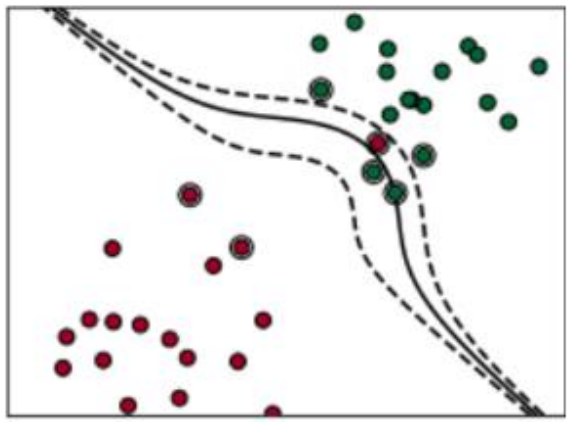
Ben Anson

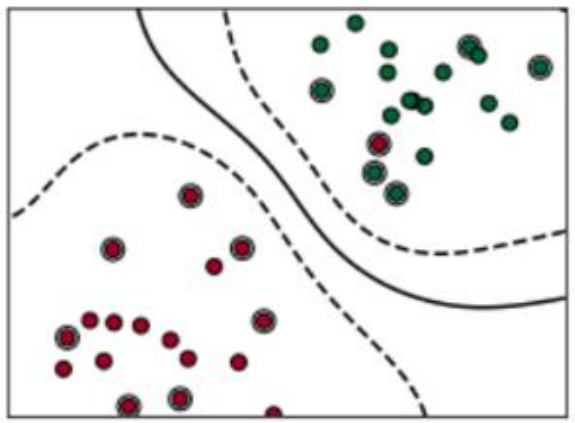

Neural Networks
Kernel Methods

	shallow	deep
feature	linear regression	
kernel	kernel ridge regression	

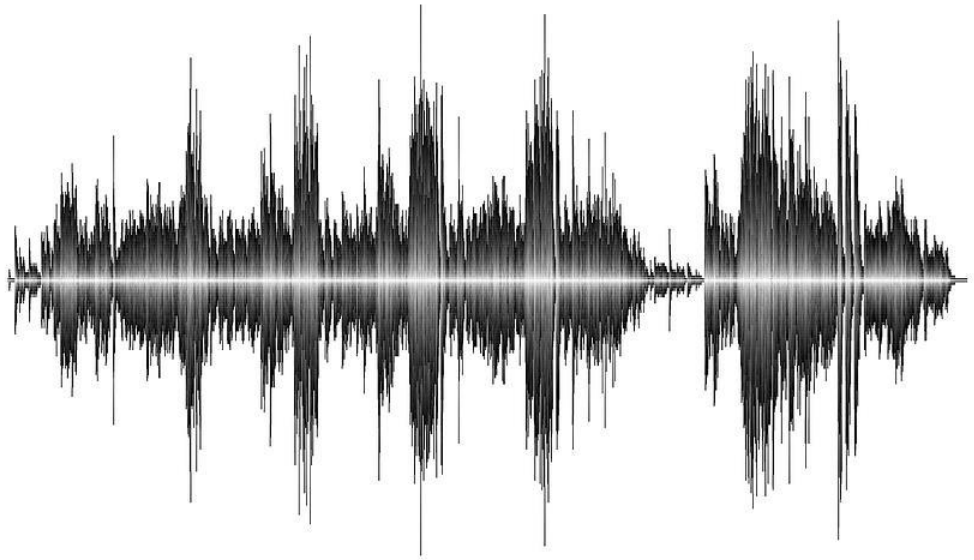
For good performance, we need to choose a choosing a good feature extractor / kernel


linear kernel


polynomial kernel


RBF kernel

Problem: can’t choose a good feature extractor/kernel for complex data like images



	shallow
feature	linear regression
kernel	kernel ridge regression

	shallow		deep
feature	linear regression	➡	neural net
kernel	kernel ridge regression		

Multiple layers
Flexibility at each layer

	shallow		deep
feature	linear regression	➡	neural net
		Multiple layers Flexibility at each layer	
kernel	kernel ridge regression	➡	**deep kernel methods**

Summary

$$\begin{tikzcd}
 & \text{deep Gaussian processes} \arrow[dl, "\text{rewrite prior in terms of Gram matrices}"'] \arrow[dr, "\text{infinite-width limit}"] & \\
\text{deep kernel } \textbf{processes} & & \text{deep kernel } \textbf{machines}
\end{tikzcd}$$

Part 1

deep Gaussian processes

rewrite prior in terms of Gram matrices

Infinite-width limit

deep kernel **processes**

deep kernel **machines**

P = number of datapoints
N_ℓ = width of layer ℓ

In deep-kernel methods, we switch to working entirely with Gram matrices

What are Gram matrices?

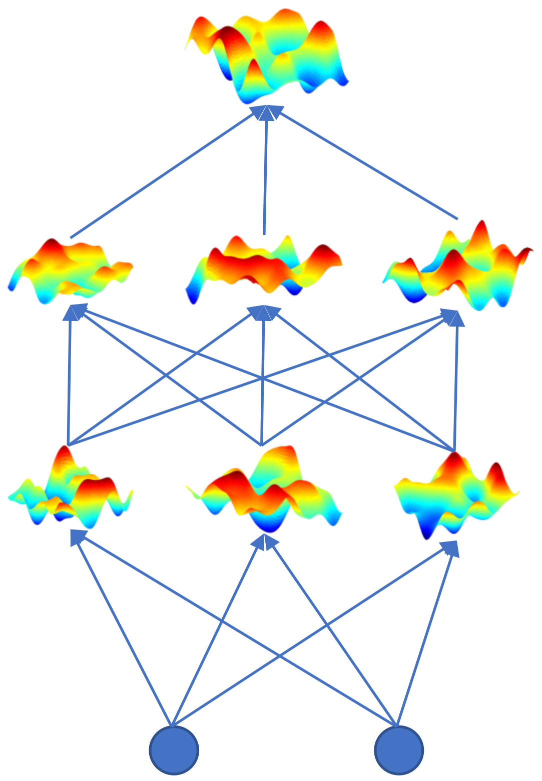
- same shape as the kernel
- Just like the kernel, Gram matrices describe similarities of training points
- Gram matrix = "representation"
- Gram matrices centered on kernel
- Gram matrices have "noise"
- So Gram matrices represent prior variability in representations!

Gram matrices

$$\overset{P\times P}{\boldsymbol{G}_2} = \frac{1}{N_2}\sum_{\lambda=1}^{N_2} \boldsymbol{f}_\lambda^2 \left(\boldsymbol{f}_\lambda^2\right)^T$$
$$E[\boldsymbol{G}_2] = \boldsymbol{K}_{\mathrm{f}}(\boldsymbol{F}_1)$$

$$\overset{P\times P}{\boldsymbol{G}_1} = \frac{1}{N_1}\sum_{\lambda=1}^{N_1} \boldsymbol{f}_\lambda^1 \left(\boldsymbol{f}_\lambda^1\right)^T$$
$$E[\boldsymbol{G}_1] = \boldsymbol{K}_{\mathrm{f}}(\boldsymbol{X})$$

$$\overset{P\times P}{\boldsymbol{G}_0} = \frac{1}{N_0}\sum_{\lambda=1}^{N_0} \boldsymbol{x}_\lambda \boldsymbol{x}_\lambda^T$$



DGP

outputs,
$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \underset{P\times P}{\boldsymbol{K}_{\mathrm{f}}}(\underset{P\times N_2}{\boldsymbol{F}_2}) + \sigma^2 \boldsymbol{I}\right)$$

hiddens,
$$\underset{P\times 1}{\boldsymbol{f}_2^\lambda} \sim \mathcal{N}\left(\mathbf{0}, \underset{P\times P}{\boldsymbol{K}_{\mathrm{f}}}(\underset{P\times N_1}{\boldsymbol{F}_1})\right)$$

hiddens,
$$\underset{P\times 1}{\boldsymbol{f}_1^\lambda} \sim \mathcal{N}\left(\mathbf{0}, \underset{P\times P}{\boldsymbol{K}_{\mathrm{f}}(\boldsymbol{X})}\right)$$

batch of
input vectors, $\underset{P\times N_X}{\boldsymbol{X}}$

GPs from Duvenaud et al. (2014)

P = number of datapoints
N_ℓ = width of layer ℓ

In deep-kernel methods, we switch to working entirely with Gram matrices

Gram matrices

DGP

What are Gram matrices?

- same shape as the kernel
- Just like the kernel, Gram matrices describe similarities of training points
- Gram matrix = "representation"
- Gram matrices centered on kernel
- Gram matrices have "noise"
- So Gram matrices represent prior variability in representations!

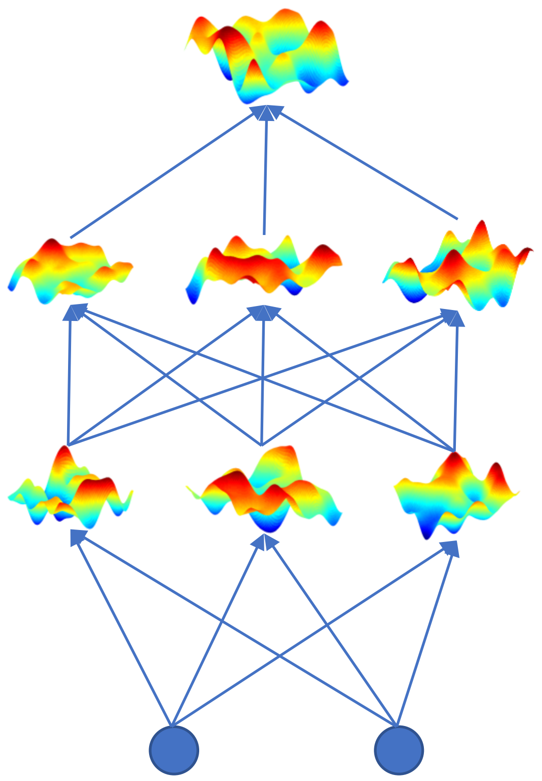
$$\overset{P\times P}{\mathbf{G}_2} = \frac{1}{N_2}\mathbf{F}_2\mathbf{F}_2^T$$

$$E[\mathbf{G}_2] = \mathbf{K}_{\mathrm{f}}(\mathbf{F}_1)$$

$$\overset{P\times P}{\mathbf{G}_1} = \frac{1}{N_1}\mathbf{F}_1\mathbf{F}_1^T$$

$$E[\mathbf{G}_1] = \mathbf{K}_{\mathrm{f}}(\mathbf{X})$$

$$\overset{P\times P}{\mathbf{G}_0} = \frac{1}{N_0}\mathbf{X}\mathbf{X}^T$$



outputs,
$$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \underset{P\times P}{\mathbf{K}_{\mathrm{f}}}(\underset{P\times N_2}{\mathbf{F}_2}) + \sigma^2\mathbf{I})$$

hiddens,
$$\underset{P\times 1}{\mathbf{f}_2^\lambda} \sim \mathcal{N}(\mathbf{0}, \underset{P\times P}{\mathbf{K}_{\mathrm{f}}}(\underset{P\times N_1}{\mathbf{F}_1}))$$

hiddens,
$$\underset{P\times 1}{\mathbf{f}_1^\lambda} \sim \mathcal{N}(\mathbf{0}, \underset{P\times P}{\mathbf{K}_{\mathrm{f}}}(\mathbf{X}))$$

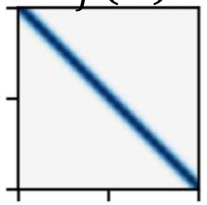
batch of
input vectors, $\underset{P\times N_X}{\mathbf{X}}$

GPs from Duvenaud et al. (2014)

A Gram matrix view on sampling from the prior in a DGPs

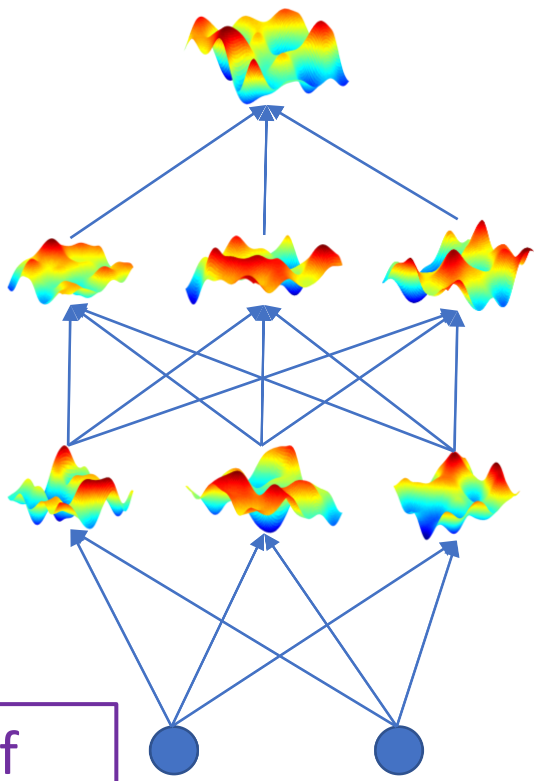
P = number of datapoints
N_ℓ = width of layer ℓ

$\boldsymbol{K}_f(\boldsymbol{X})$



$\boldsymbol{G}_1 = \boldsymbol{F}_1 \boldsymbol{F}_1^T / N_1$ $\boldsymbol{G}_2 = \boldsymbol{F}_2 \boldsymbol{F}_2^T / N_1$

DGP



outputs,
$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \underbrace{\boldsymbol{K}_{\mathrm{f}}(\underbrace{\boldsymbol{F}_2}_{P \times N_2})}_{P \times P} + \sigma^2 \boldsymbol{I})$

hiddens,
$\underbrace{\boldsymbol{f}_2^\lambda}_{P \times 1} \sim \mathcal{N}(\mathbf{0}, \underbrace{\boldsymbol{K}_{\mathrm{f}}(\underbrace{\boldsymbol{F}_1}_{P \times N_1})}_{P \times P})$

hiddens,
$\underbrace{\boldsymbol{f}_1^\lambda}_{P \times 1} \sim \mathcal{N}(\mathbf{0}, \underbrace{\boldsymbol{K}_{\mathrm{f}}(\boldsymbol{X})}_{P \times P})$

batch of
input vectors, $\underbrace{\boldsymbol{X}}_{P \times N_X}$

So can we write the DGP prior entirely in terms of Gram matrices? Yes! But we need two tricks.

GPs from Duvenaud et al. (2014)

Trick 1: most kernels of interest can be computed from the Gram matrix

- True for e.g. arccos kernels used in infinite NNs (Cho and Saul 2009)
- Also true for standard GP kernels that only depend on distance between datapoints i and j, because we can recover distance from the Gram matrix, (Duvenaud et al. 2014)

$$\begin{aligned} R_{ij}(\mathbf{G}) &= \tfrac{1}{N}\textstyle\sum_{\lambda=1}^{N}\left(F_{i\lambda} - F_{j\lambda}\right)^2 \\ &= \tfrac{1}{N}\textstyle\sum_{\lambda=1}^{N}\left(\left(F_{i\lambda}\right)^2 - 2F_{i\lambda}F_{j\lambda} + \left(F_{j\lambda}\right)^2\right) \\ &= G_{ii} - 2G_{ij} + G_{jj} \end{aligned}$$

- Overall:

$$\boldsymbol{K}_f(\boldsymbol{F}_\ell) = \boldsymbol{K}(\boldsymbol{G}_\ell)$$

Trick 2: Gram matrices are Wishart distributed

To get next Gram matrix, we first sample a bunch of features,

$$\boldsymbol{F}_\ell \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_{\ell-1})\right)$$

And then compute the Gram matrix

$$\boldsymbol{G}_\ell = \frac{1}{N_\ell} \boldsymbol{F}_\ell \boldsymbol{F}_\ell^T$$

But this exactly matches the definition of the Wishart distribution!

$$\boldsymbol{G}_\ell \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_{\ell-1})/N_\ell, N_\ell)$$

(e.g. see Wikipedia for pdf, moments etc.)

In deep-kernel methods, we switch to working entirely with Gram matrices

P = number of datapoints
N_ℓ = width of layer ℓ

DKP

outputs,
$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{K}(\boldsymbol{G}_2) + \sigma^2 \boldsymbol{I})$

Trick 1: Kernel can be written as a function of the Gram matrix

$\boldsymbol{G}_2 \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_1)/N_2, N_2)$

Trick 2: Gram matrices are Wishart distributed

$\boldsymbol{G}_1 \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_0)/N_1, N_1)$

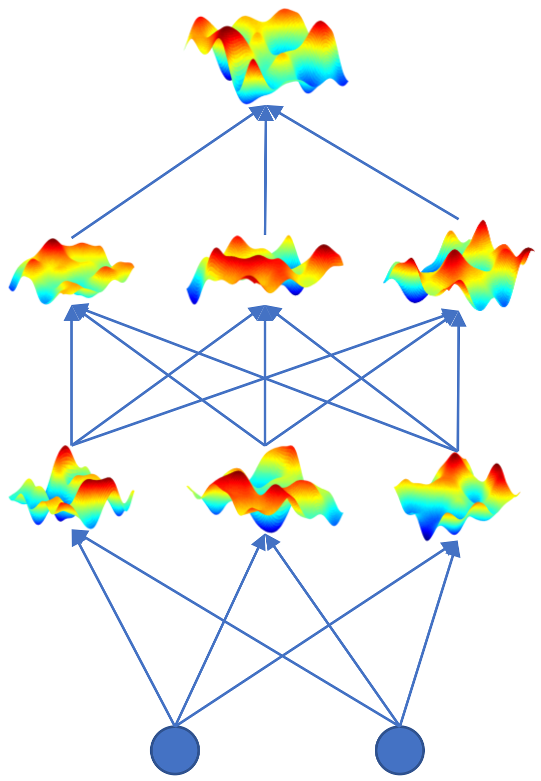
$\boldsymbol{G}_0 = \boldsymbol{X}\boldsymbol{X}^T/N_X$

Gram matrices

$\underset{P \times P}{\boldsymbol{G}_2} = \boldsymbol{F}_2 \boldsymbol{F}_2^T / N_2$

$\underset{P \times P}{\boldsymbol{G}_1} = \boldsymbol{F}_1 \boldsymbol{F}_1^T / N_1$

$\underset{P \times P}{\boldsymbol{G}_0} = \boldsymbol{X}\boldsymbol{X}^T/N_X$



DGP

outputs,
$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \underset{P \times P}{\boldsymbol{K}_\text{f}(\boldsymbol{F}_2)} + \sigma^2 \boldsymbol{I})$

hiddens,
$\underset{P \times N_2}{\boldsymbol{F}_2} \sim \mathcal{N}(\mathbf{0}, \underset{P \times P}{\boldsymbol{K}_\text{f}(\boldsymbol{F}_1)})$

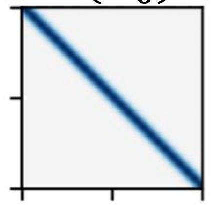
hiddens,
$\underset{P \times N_1}{\boldsymbol{F}_1} \sim \mathcal{N}(\mathbf{0}, \underset{P \times P}{\boldsymbol{K}_\text{f}(\boldsymbol{X})})$

batch of
input vectors, $\underset{P \times N_X}{\boldsymbol{X}}$

GPs from Duvenaud et al. (2014)

Sampling the prior in the kernelized DGP

$\boldsymbol{K}(\boldsymbol{G}_0)$



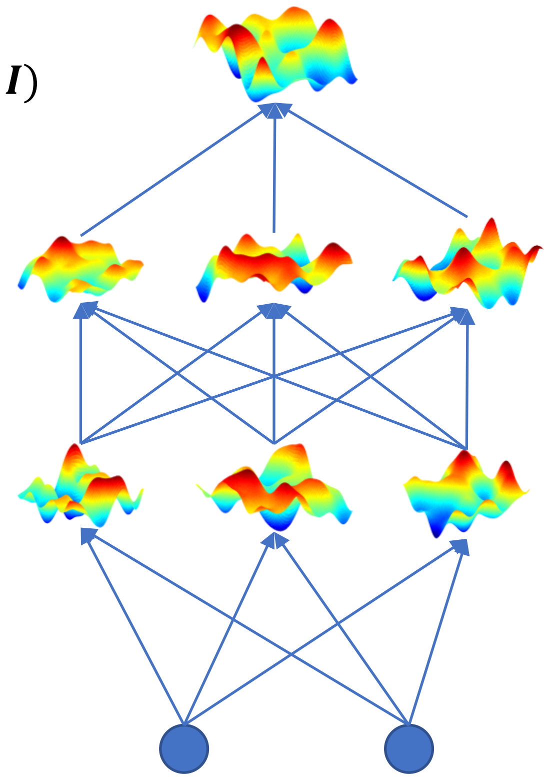
DKP

outputs,
$\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{K}\,(\boldsymbol{G}_2) + \sigma^2 \boldsymbol{I})$

$\boldsymbol{G}_2 \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_1)/N_2, N_2)$

$\boldsymbol{G}_1 \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_0)/N_1, N_1)$

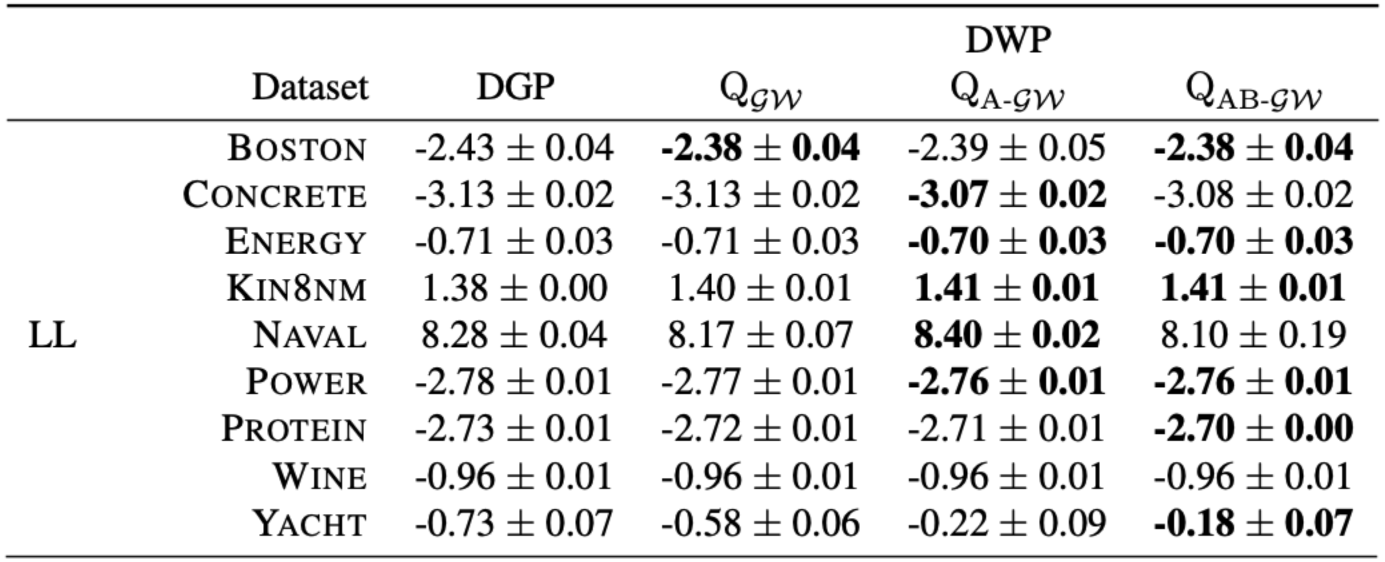
$\boldsymbol{G}_0 = \boldsymbol{X}\boldsymbol{X}^T/N_X$



Developing practical methods + our results

We developed:

- Two processes: “deep Wishart process” and “deep *inverse* Wishart process”
- VI with priors + approximate posteriors over Gram matrices, *not* features.
- a bunch of approximate posteriors (e.g. $\mathrm{Q}_{\mathcal{GW}}$)



	Dataset	DGP	DWP $\mathrm{Q}_{\mathcal{GW}}$	$\mathrm{Q}_{\text{A-}\mathcal{GW}}$	$\mathrm{Q}_{\text{AB-}\mathcal{GW}}$
	BOSTON	-2.43 ± 0.04	$\mathbf{-2.38 \pm 0.04}$	-2.39 ± 0.05	$\mathbf{-2.38 \pm 0.04}$
	CONCRETE	-3.13 ± 0.02	-3.13 ± 0.02	$\mathbf{-3.07 \pm 0.02}$	-3.08 ± 0.02
	ENERGY	-0.71 ± 0.03	-0.71 ± 0.03	$\mathbf{-0.70 \pm 0.03}$	$\mathbf{-0.70 \pm 0.03}$
	KIN8NM	1.38 ± 0.00	1.40 ± 0.01	$\mathbf{1.41 \pm 0.01}$	$\mathbf{1.41 \pm 0.01}$
LL	NAVAL	8.28 ± 0.04	8.17 ± 0.07	$\mathbf{8.40 \pm 0.02}$	8.10 ± 0.19
	POWER	-2.78 ± 0.01	-2.77 ± 0.01	$\mathbf{-2.76 \pm 0.01}$	$\mathbf{-2.76 \pm 0.01}$
	PROTEIN	-2.73 ± 0.01	-2.72 ± 0.01	-2.71 ± 0.01	$\mathbf{-2.70 \pm 0.00}$
	WINE	-0.96 ± 0.01	-0.96 ± 0.01	-0.96 ± 0.01	-0.96 ± 0.01
	YACHT	-0.73 ± 0.07	-0.58 ± 0.06	-0.22 ± 0.09	$\mathbf{-0.18 \pm 0.07}$

[1] Aitchison, Yang and Ober. “*Deep kernel processes*” ICML (2021)
[2] Ober and Aitchison “*An approximate posterior for the deep Wishart process*” NeurIPS (2021)
[3] Ober, Anson, Milsom and Aitchison “*An improved approximate posterior for the deep Wishart process*” UAI (2023)

Part 1

deep Gaussian processes

rewrite prior in terms of Gram matrices

infinite-width limit

deep kernel **processes**

deep kernel **machines**

Part 2

deep Gaussian processes

rewrite prior in terms of Gram matrices

infinite-width limit

deep kernel **processes**

deep kernel **machines**

Why **less(?) Bayesian** deep kernel **machines**?

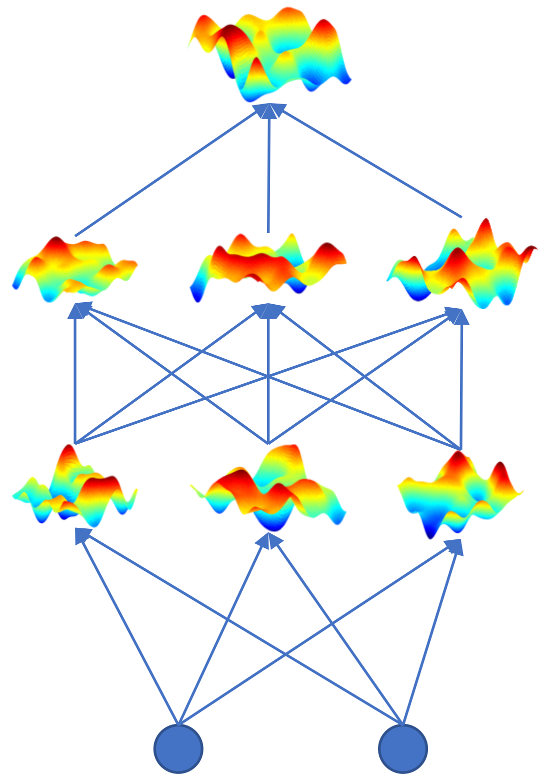
Less(?) Bayesian approach:

- simplifies implementation
- gives lower-variance updates that converge faster
- provides a cleaner link to NN / neuro-theory
- great preliminary results…

A "deep kernel machine" is an infinite-width DGP, trained using variational inference

DGP Approximate Posterior

DGP Prior



$$P(\underset{P\times 1}{\boldsymbol{y}}|\boldsymbol{F}_2) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_2) + \sigma^2\boldsymbol{I})$$

$$Q(\boldsymbol{f}_\lambda^2) = \mathcal{N}(\boldsymbol{f}_\lambda^2; \boldsymbol{0}, \boldsymbol{\Sigma}_2)$$

$$P(\underset{P\times 1}{\boldsymbol{f}_\lambda^2}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}_\lambda^2; \boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_1))$$

$$\boldsymbol{G}_2 = \boldsymbol{F}_2\boldsymbol{F}_2^T/N_2$$

$$Q(\boldsymbol{f}_\lambda^1) = \mathcal{N}(\boldsymbol{f}_\lambda^1; \boldsymbol{0}, \boldsymbol{\Sigma}_1)$$

$$P(\underset{P\times 1}{\boldsymbol{f}_\lambda^1}|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}_\lambda^1; \boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_0))$$

$$\boldsymbol{G}_1 = \boldsymbol{F}_1\boldsymbol{F}_1^T/N_1$$

true posterior is in this family!

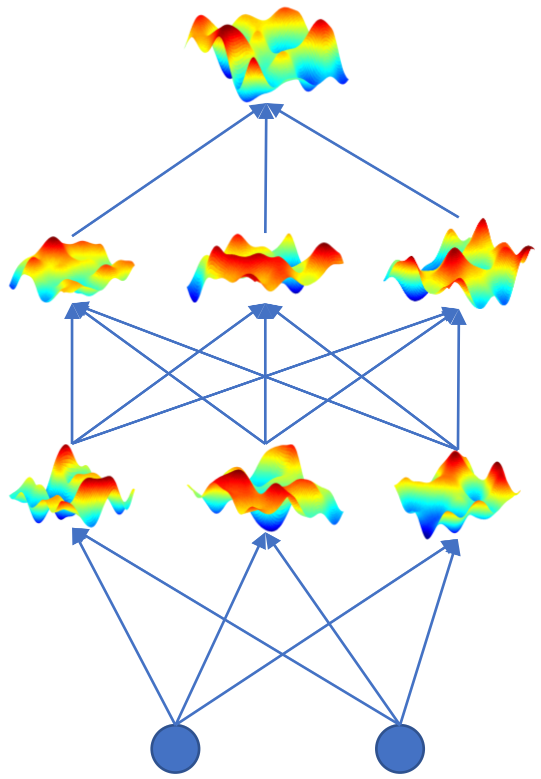
batch of input vectors, $\boldsymbol{X}$

$$\boldsymbol{G}_0 = \boldsymbol{X}\boldsymbol{X}^T/N_X$$

A "deep kernel machine" is an infinite-width DGP, trained using variational inference

DGP Approximate Posterior

DGP Prior



$P(\boldsymbol{y}|\boldsymbol{F}_2) = \mathcal{N}(\boldsymbol{y}; \boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_2) + \sigma^2 \boldsymbol{I})$

$P \times 1$

$Q(\boldsymbol{f}_\lambda^2) = \mathcal{N}(\boldsymbol{f}_\lambda^2; \boldsymbol{0}, \mathbf{G}_2)$

$P\left(\boldsymbol{f}_\lambda^2 \middle| \boldsymbol{X}\right) = \mathcal{N}(\boldsymbol{f}_\lambda^2; \boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_1))$

$P \times 1$

$\boldsymbol{G}_2 = \boldsymbol{F}_2 \boldsymbol{F}_2^T / N_2 = \boldsymbol{\Sigma}_2$

$Q(\boldsymbol{f}_\lambda^1) = \mathcal{N}(\boldsymbol{f}_\lambda^1; \boldsymbol{0}, \mathbf{G}_1)$

$P\left(\boldsymbol{f}_\lambda^1 \middle| \boldsymbol{X}\right) = \mathcal{N}(\boldsymbol{f}_\lambda^1; \boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_0))$

$P \times 1$

$\boldsymbol{G}_1 = \boldsymbol{F}_1 \boldsymbol{F}_1^T / N_1 = \boldsymbol{\Sigma}_1$

true posterior is in this family!

batch of input vectors, $\boldsymbol{X}$

$\boldsymbol{G}_0 = \boldsymbol{X}\boldsymbol{X}^T / N_X$

We're doing VI, so the objective is the ELBO

$$\mathrm{ELBO}\underbrace{(\boldsymbol{G}_1, \dots, \boldsymbol{G}_L)}_{\text{approx post covs}} = \underbrace{\log \mathrm{P}(\mathbf{Y} | \boldsymbol{G}_L)}_{\text{likelihood}} - \underbrace{\beta}_{\text{temp}} \sum_{\ell=1}^{L} N_\ell D_{\mathrm{KL}}\big(\underbrace{\mathcal{N}(0, \boldsymbol{G}_\ell)}_{Q(\boldsymbol{f}_\lambda^\ell)} \,\|\, \underbrace{\mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1}))}_{P(\boldsymbol{f}_\lambda^\ell | \boldsymbol{F}_{\ell-1})}\big)$$

- Optimizes Gram matrices (equivalently, approximate posterior covariances)
- Likelihood encourages Gram matrices/representations that give good performance on the task
- KL keeps Gram matrices, $\boldsymbol{G}_\ell$, similar to value we'd expect from previous layer, i.e. $\boldsymbol{K}(\boldsymbol{G}_{\ell-1})$

All the features have disappeared! The objective can be computed analytically, as a function of just the Gram matrices!

Taking the infinite-width limit of the ELBO gives the "deep kernel machine"

$$\mathrm{ELBO}(\boldsymbol{G}_1, \dots, \boldsymbol{G}_L) = \log \mathrm{P}(\mathbf{Y}|\, \boldsymbol{G}_L) - \beta \sum_{\ell=1}^{L} N_\ell D_{\mathrm{KL}}(\mathcal{N}(0, \boldsymbol{G}_\ell) \,\|\mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1})))$$

- We take the limit for all layer widths jointly, by sending $N \to \infty$, with
$$N_\ell = \nu_\ell N$$
- Problem: the ELBO blows up in the infinite-width limit.
- Solution: set $\beta = 1/\mathrm{N}$
- That's … it! So the DKM objective is:

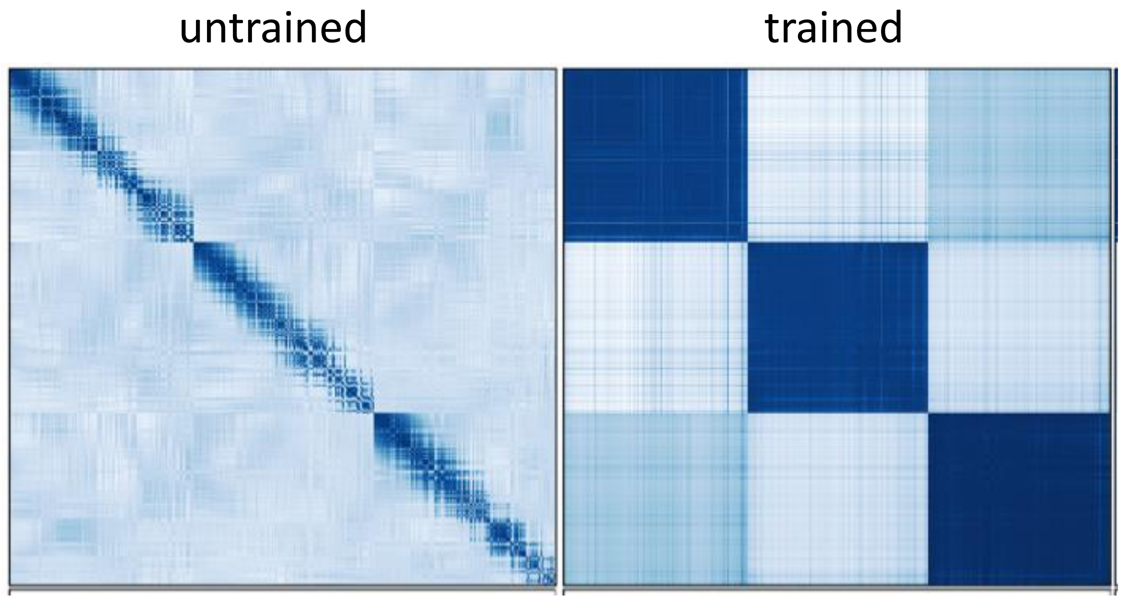
$$\mathrm{DKM}(\boldsymbol{G}_1, \dots, \boldsymbol{G}_L) = \log \mathrm{P}(\mathbf{Y}|\, \boldsymbol{G}_L) - \sum_{\ell=1}^{L} \nu_\ell D_{\mathrm{KL}}(\mathcal{N}(0, \boldsymbol{G}_\ell) \,\|\mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1})))$$

What is a deep kernel machine?

- A nonlinear function approximator
- With multiple layers
- Parameterised by *Gram matrices*, not features or weights
- Trained using the DKM objective:

$$\mathrm{DKM}(\boldsymbol{G}_1, \dots, \boldsymbol{G}_L) = \log \mathrm{P}(\mathbf{Y} | \boldsymbol{G}_L) - \sum_{\ell=1}^{L} \nu_\ell D_{\mathrm{KL}}(\mathcal{N}(0, \boldsymbol{G}_\ell) \,\|\, \mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1})))$$

The deep kernel machine viewpoint helps us understand theory!

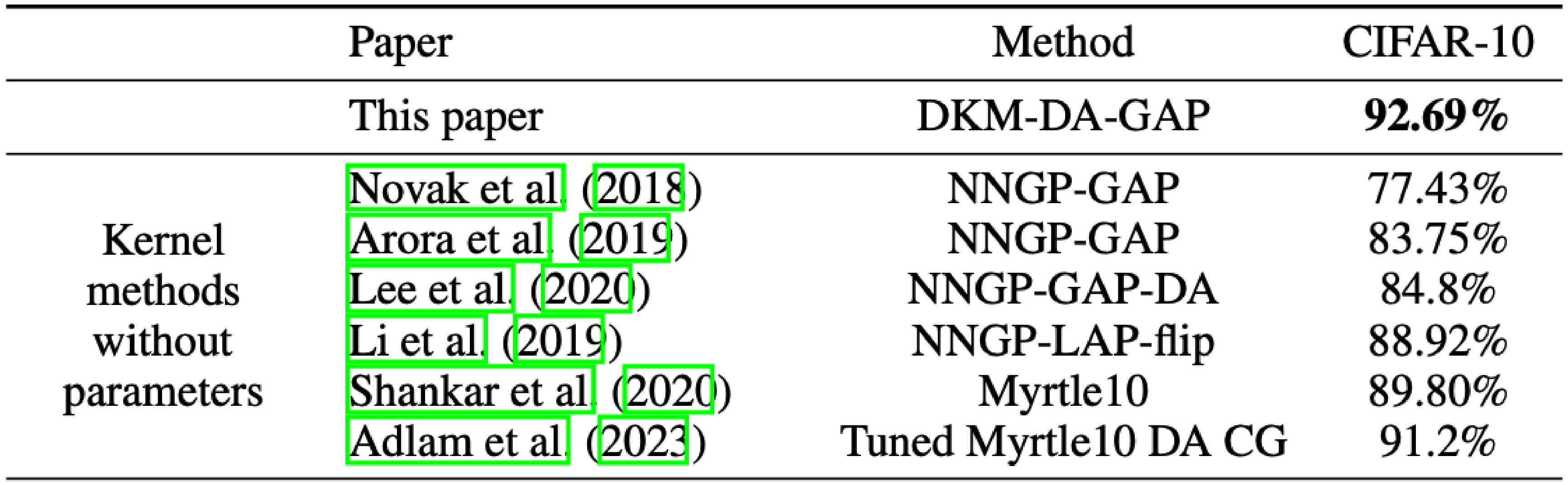


For regression, the DKM objective can be written:

$$\mathrm{DKM}(\boldsymbol{G}_1, \ldots, \boldsymbol{G}_L) = D_{\mathrm{KL}}(\mathcal{N}(0, \boldsymbol{Y}\boldsymbol{Y}^T/N_Y) \,\|\, \mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_L) + \sigma^2 \boldsymbol{I}))$$
$$- \textstyle\sum_{\ell=1}^{L} \nu_\ell D_{\mathrm{KL}}(\mathcal{N}(0, \boldsymbol{G}_\ell) \,\|\, \mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1})))$$

picture from Wu et al. (2022)

Deep kernel machines work well in practice!



	Paper	Method	CIFAR-10
	This paper	DKM-DA-GAP	**92.69%**
Kernel methods without parameters	Novak et al. (2018)	NNGP-GAP	77.43%
	Arora et al. (2019)	NNGP-GAP	83.75%
	Lee et al. (2020)	NNGP-GAP-DA	84.8%
	Li et al. (2019)	NNGP-LAP-flip	88.92%
	Shankar et al. (2020)	Myrtle10	89.80%
	Adlam et al. (2023)	Tuned Myrtle10 DA CG	91.2%

Edward Milsom

Ed has pushed performance further, to better than 94%!

But how slow are DKMs? Surprisingly fast!

- We develop a novel inducing-point scheme
- Same FLOPs as CNN (computations ultimately look v. similar)
- Slower than a CNN, but orders of magnitude faster than “full” kernel methods in table.

Summary

$$\begin{tikzcd}
 & \text{deep Gaussian processes} \arrow[dl, "\text{rewrite prior in terms of Gram matrices}"'] \arrow[dr, "\text{infinite-width limit}"] & \\
\text{deep kernel } \textbf{processes} & & \text{deep kernel } \textbf{machines}
\end{tikzcd}$$

Deep kernel landscape + our priorities

Our priorities

- More architectures for DKMs (GNNs + transformers).
- speed/scale-up:
  - memory efficiency
  - lower-precision
- user-friendly library (we can share preliminary work)

Huge future opportunities:

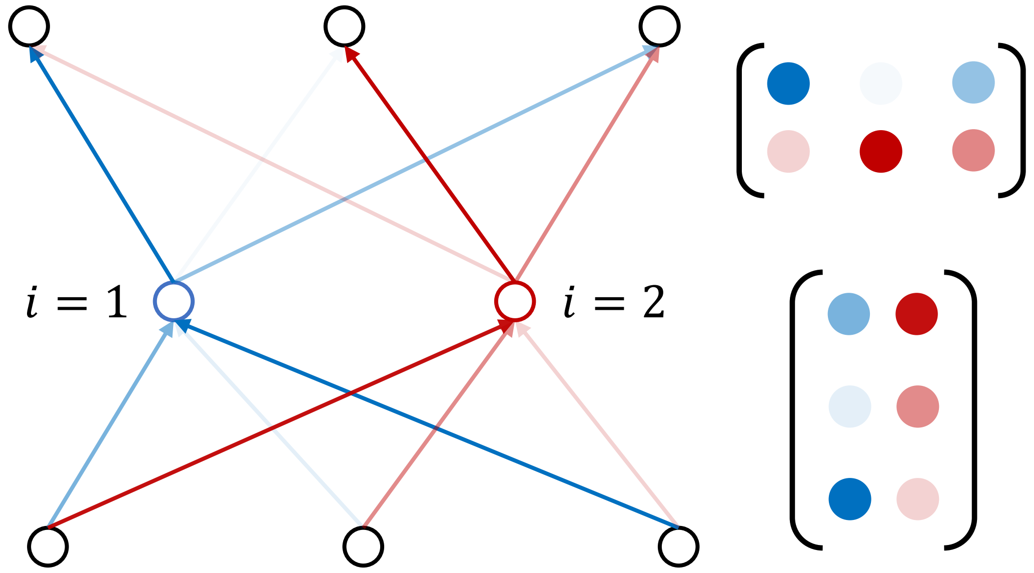
	shallow	deep
feature	linear regression	neural net
kernel	kernel ridge regression	**deep kernel methods**

If you're interested, get in touch:
laurence.aitchison@bristol.ac.uk

[1] Aitchison, Yang and Ober. "*Deep kernel processes*" ICML (2021)
[2] Ober and Aitchison "*An approximate posterior for the deep Wishart process*" NeurIPS (2021)
[3] Ober, Anson, Milsom and Aitchison "*An improved approximate posterior for the deep Wishart process*" UAI (2023)
[4] Yang, Robeyns, Milsom, Anson, Schoots, Aitchison "A theory of representation learning gives a deep generalisation of kernel methods" ICML (2023)
[5] Milsom, Anson, Aitchison "Convolutional deep kernel machines" ICLR (2024)

Appendix slides

Deep kernel processes should work better because they have fewer local optima



Deep kernel processes should work better because they have fewer local optima

- Implies loads of symmetric local optima…
- …and local optima are bad if you have unimodal approximate posteriors.
- DKPs don't have these symmetries, so *far* fewer local optima!