(Convolutional) deep kernel machines

Laurence Aitchison

Assistant Prof.

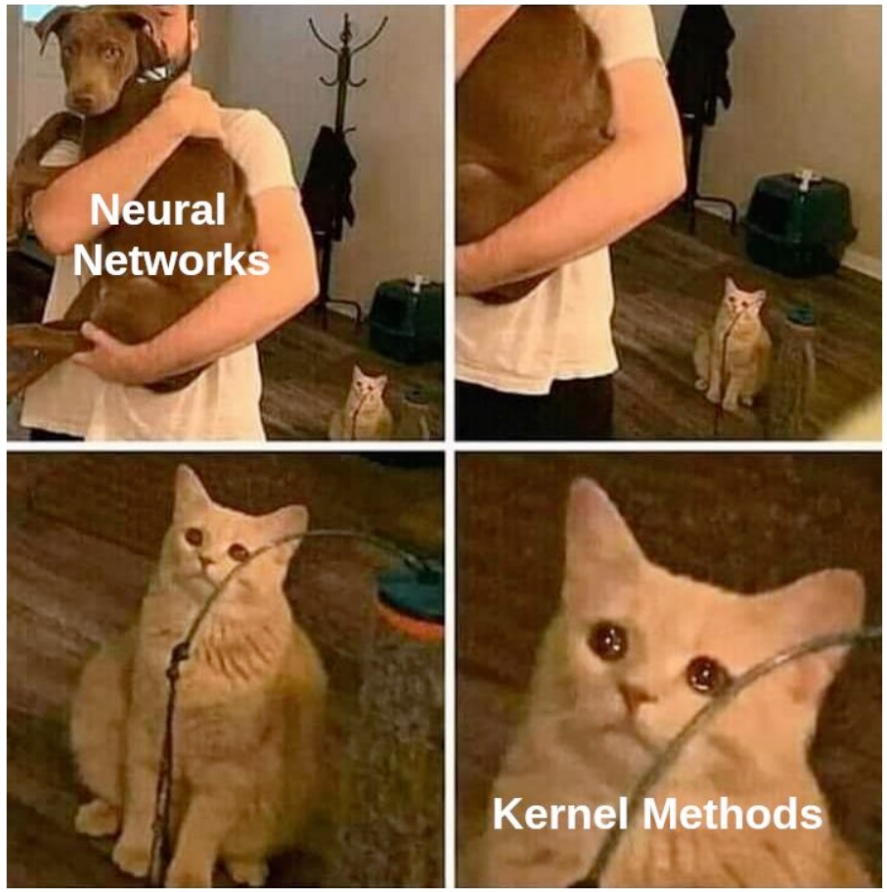
University of Bristol

Sebastian Ober

Adam Yang

Edward Milsom

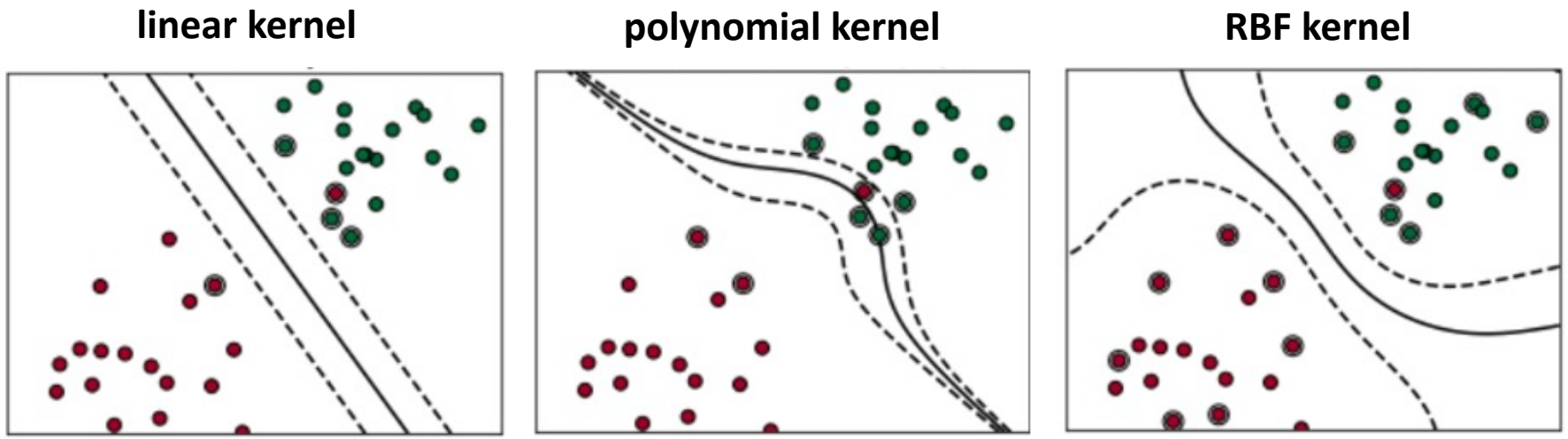
Ben Anson



Edward
Milsom

	shallow	deep
feature	linear regression	
kernel	kernel ridge regression	

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

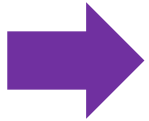


Problem: can’t choose a good feature extractor/kernel for complex data like images...

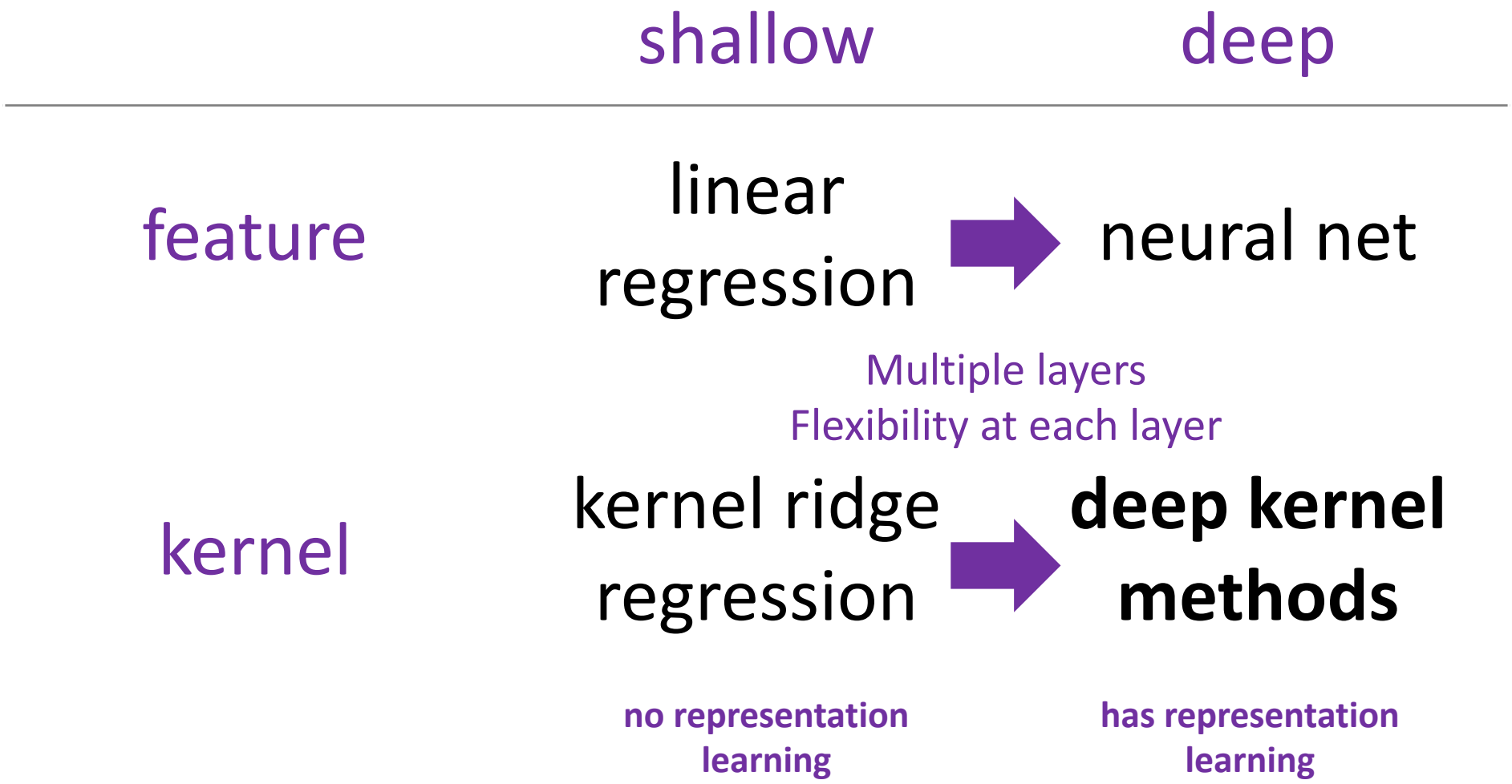
need to learn a representation
(i.e. feature extractor / kernel!)

	shallow
feature	linear regression
kernel	kernel ridge regression

	shallow	deep
feature	linear regression	neural net
kernel	kernel ridge regression	



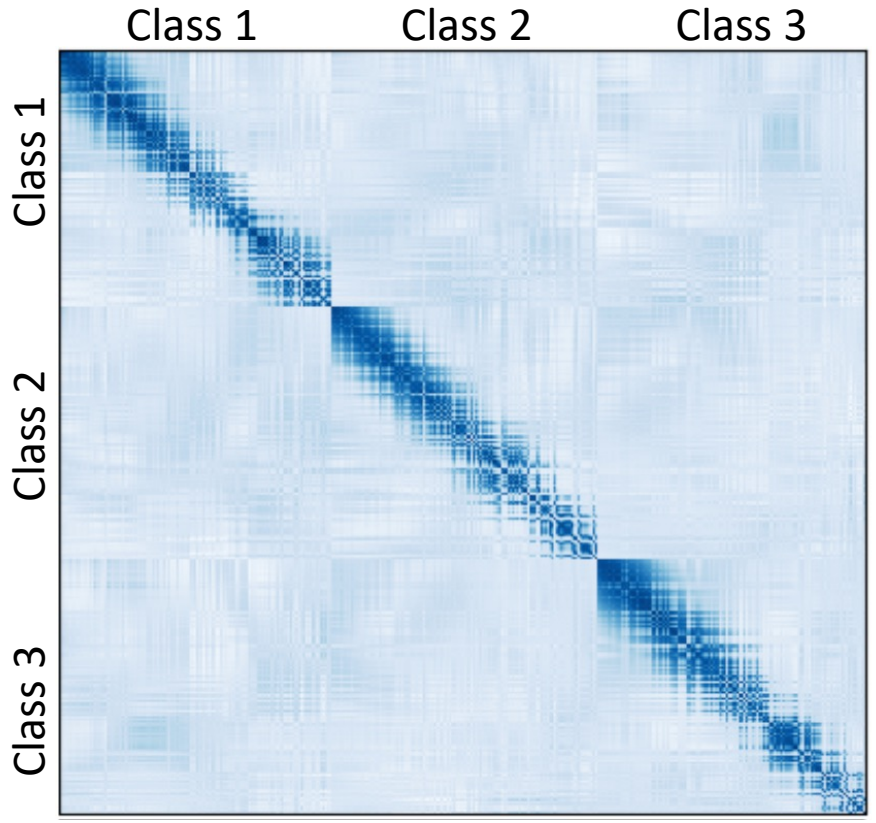
Multiple layers
Flexibility at each layer


shallow
deep
feature
linear regression
neural net
Multiple layers
Flexibility at each layer
kernel
kernel ridge regression
deep kernel methods
no representation learning
has representation learning

What representation learning looks like

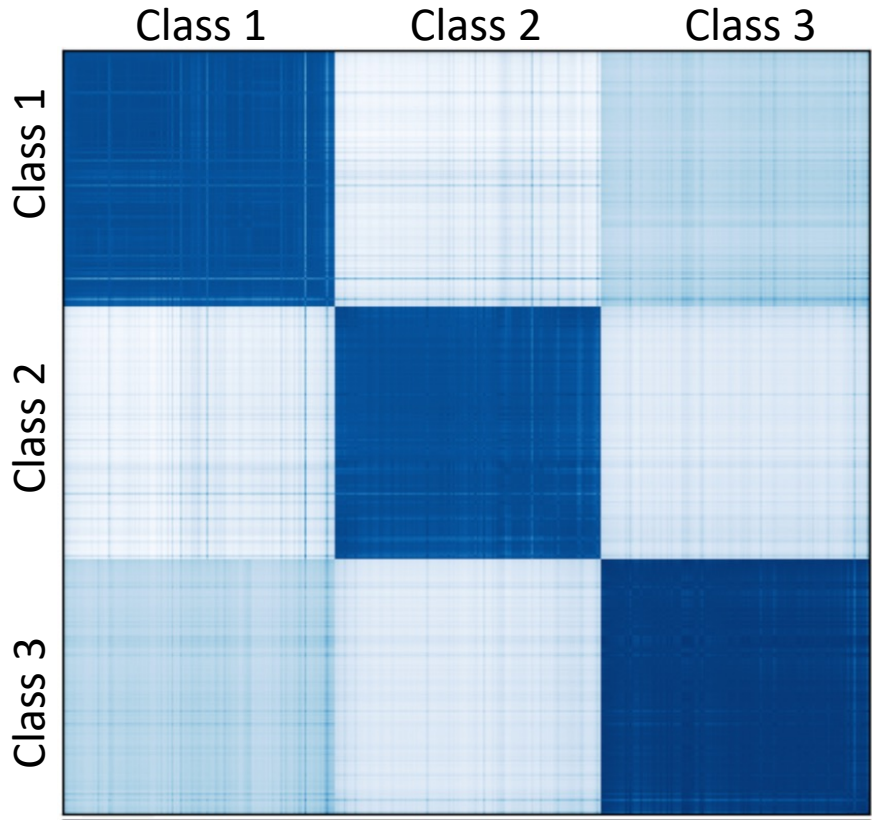
- dot product of neural activations, for all pairs of training examples.
- intuition: basically the cosine similarity

This matrix = the kernel!



before/no representation learning:

- Hand-picked kernel/features
- Finite randomly initialized NN
- Infinite randomly initialized network (NNGP/NTK)

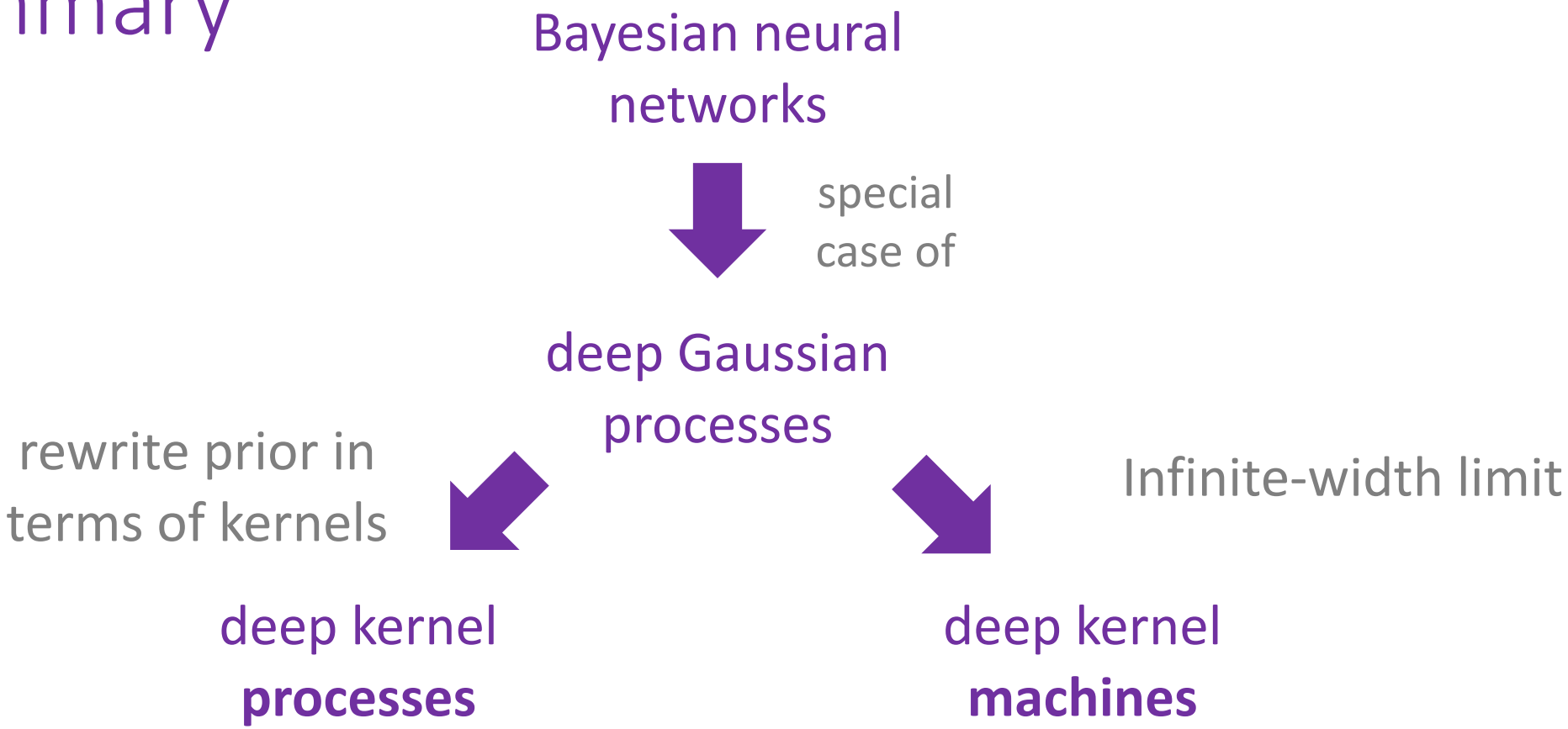


learned representation learning:

- Trained NN
- Trained deep kernel method.

Pictures from Wu et al. (2022)

Summary


Bayesian neural networks
special case of
deep Gaussian processes
rewrite prior in terms of kernels
Infinite-width limit
deep kernel **processes**
deep kernel **machines**

Part 0

Bayesian neural networks

special case of

deep Gaussian processes

rewrite prior in terms of kernels

Infinite-width limit

deep kernel **processes**

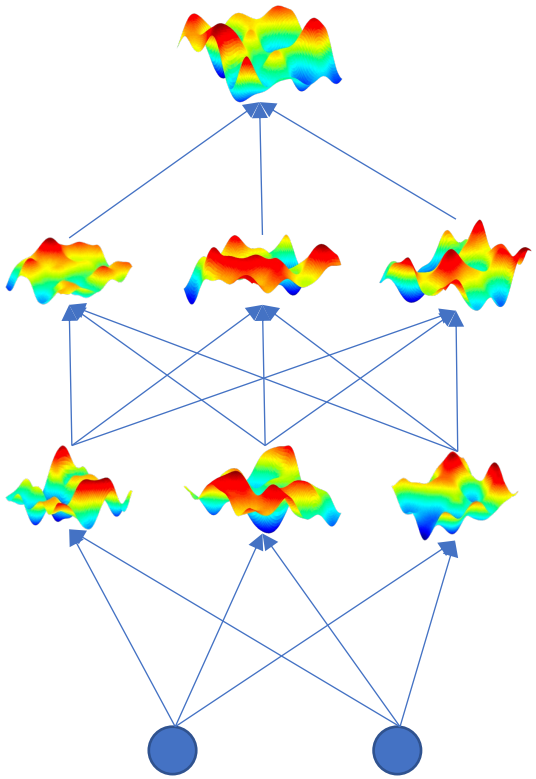
deep kernel **machines**

P = number of datapoints
N_ℓ = width of layer ℓ

Bayesian neural networks (BNNs) are a special case of deep Gaussian processes (DGPs)

BNNs are a special case of DGPs, with a particular choice of kernel:

$$\underset{P\times P}{\boldsymbol{K}_{\mathrm{f}}(\mathbf{F})} = \underset{P\times N}{\phi(\boldsymbol{F})}\big(\underset{N\times P}{\phi(\boldsymbol{F})}\big)^T$$



DGP

outputs,
$\underset{P\times 1}{\hat{\boldsymbol{y}}} \sim \mathcal{N}(\mathbf{0}, \underset{P\times P}{\boldsymbol{K}_{\mathrm{f}}(\boldsymbol{F}_2)})$

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

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

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

BNN

outputs,
$\boldsymbol{w}_3 \sim \mathcal{N}(0, 1/N_2))$
$\underset{P\times 1}{\hat{\boldsymbol{y}}} = \phi(\boldsymbol{F}_2)\boldsymbol{w}_2$

hiddens,
$\boldsymbol{W}_2 \sim \mathcal{N}(0, 1/N_1))$
$\underset{P\times N_2}{\boldsymbol{F}_2} = \phi(\boldsymbol{F}_1)\boldsymbol{W}_2$

hiddens,
$\boldsymbol{W}_0 \sim \mathcal{N}(0, 1/N_X))$
$\boldsymbol{W}_1 \sim \mathcal{N}(0, 1/N_0))$
$\underset{P\times N_1}{\boldsymbol{F}_1} = \phi(\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1$

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

GPs from Duvenaud et al. (2014)

Part 0

Bayesian neural networks

special case of

deep Gaussian processes

rewrite prior in terms of kernels

Infinite-width limit

deep kernel **processes**

deep kernel **machines**

Part 1

Bayesian neural networks

special case of

deep Gaussian processes

rewrite prior in terms of kernels

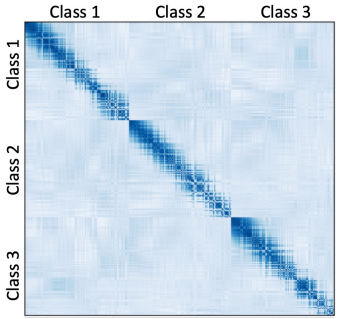
Infinite-width limit

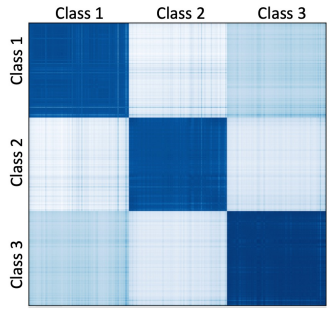
deep kernel **processes**

deep kernel **machines**

P = number of datapoints
N_ℓ = width of layer ℓ

Deep kernel processes are "reparametrized" deep Gaussian processes



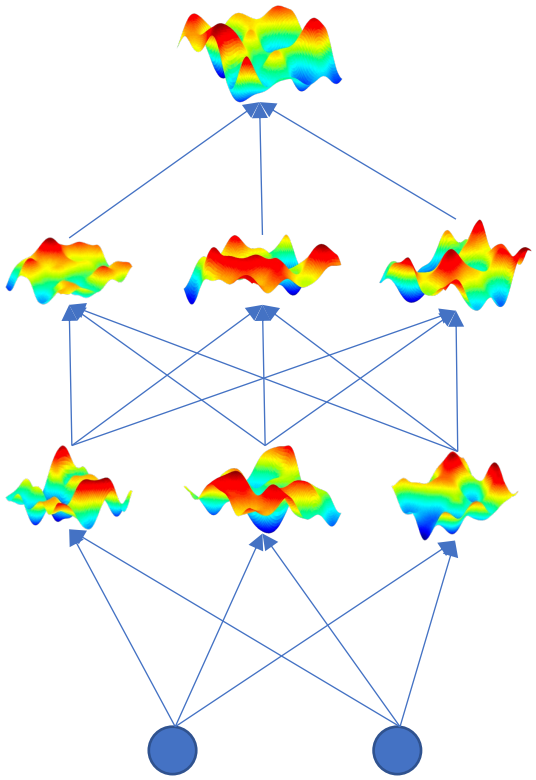


Gram matrices

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

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

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



DGP

outputs,
$$\underset{P\times 1}{\hat{\boldsymbol{y}}} \sim \mathcal{N}(\boldsymbol{0}, \underset{P\times P}{\boldsymbol{K}_f(\boldsymbol{F}_2)})$$

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

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

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

BNN

outputs,
$$\boldsymbol{w}_3 \sim \mathcal{N}(0, 1/N_2))$$
$$\underset{P\times 1}{\hat{\boldsymbol{y}}} = \phi(\boldsymbol{F}_2)\boldsymbol{w}_2$$

hiddens,
$$\boldsymbol{W}_2 \sim \mathcal{N}(0, 1/N_1))$$
$$\underset{P\times N_2}{\boldsymbol{F}_2} = \phi(\boldsymbol{F}_1)\boldsymbol{W}_2$$

hiddens,
$$\boldsymbol{W}_0 \sim \mathcal{N}(0, 1/N_X))$$
$$\boldsymbol{W}_1 \sim \mathcal{N}(0, 1/N_0))$$
$$\underset{P\times N_1}{\boldsymbol{F}_1} = \phi(\boldsymbol{X}\boldsymbol{W}_0)\boldsymbol{W}_1$$

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

GPs from Duvenaud et al. (2014)

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

$$\boldsymbol{K}_f(\boldsymbol{F}_\ell) = \boldsymbol{K}(\boldsymbol{G}_\ell)$$
$$\boldsymbol{G}_\ell = \boldsymbol{F}_\ell \boldsymbol{F}_\ell^T / N_\ell$$

- Holds for kernels corresponding to infinite-width BNNs!
- Doesn't quite hold for $\boldsymbol{K}_{\mathrm{f}}(\mathbf{F}) = \phi(\boldsymbol{F})\big(\phi(\boldsymbol{F})\big)^T$, which corresponds to finite BNNs.
- 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)

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

Trick 2: Gram matrices are Wishart distributed

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

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

And then compute the Gram matrix

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

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 | **Gram matrices** | **DGP**

outputs,
$\hat{\boldsymbol{y}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{K}(\boldsymbol{G}_2))$

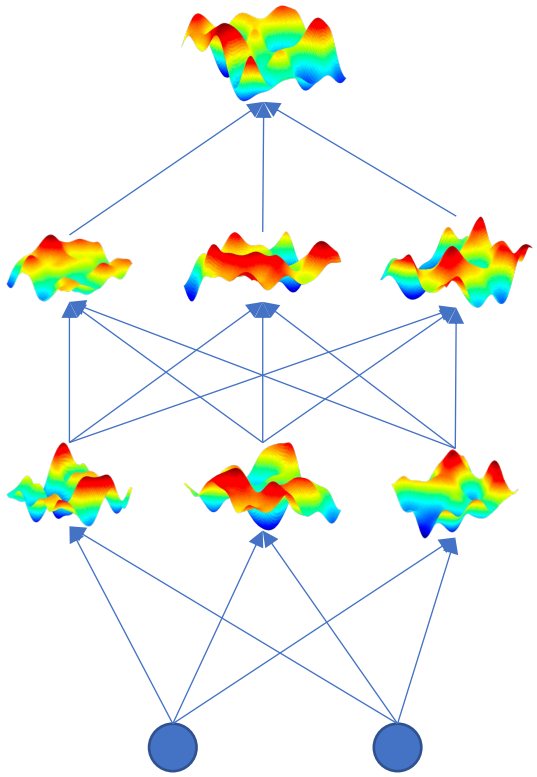
Trick 1: Kernel can be written as a function of the Gram matrix

$\boldsymbol{G}_2 \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_1)/N_2, N_2)$ | $\underset{P\times P}{\boldsymbol{G}_2} = \boldsymbol{F}_2\boldsymbol{F}_2^T/N_2$

Trick 2: Gram matrices are Wishart distributed

$\boldsymbol{G}_1 \sim \mathcal{W}(\boldsymbol{K}(\boldsymbol{G}_0)/N_1, N_1)$ | $\underset{P\times P}{\boldsymbol{G}_1} = \boldsymbol{F}_1\boldsymbol{F}_1^T/N_1$

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



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

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

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

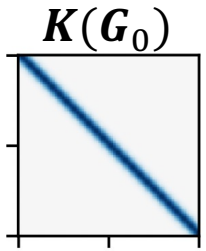
batch of input vectors, $\underset{P\times N_X}{\boldsymbol{X}}$

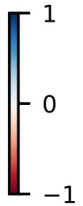
GPs from Duvenaud et al. (2014)

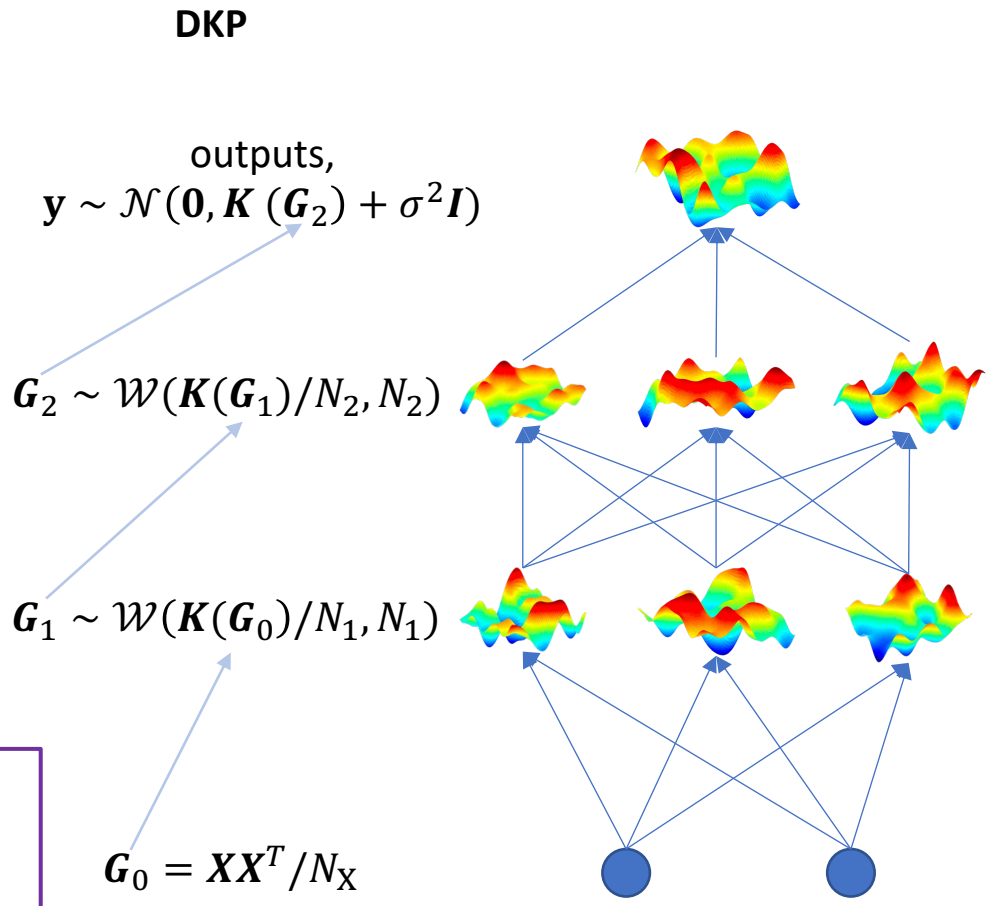
Sampling the prior in the kernelized DGP

- Next Gram matrix is "centered" on the kernel,

$$E[\boldsymbol{G}_1|\boldsymbol{G}_0] = \boldsymbol{K}(\boldsymbol{G}_0)$$







intuitive, understandable prior over functions in a deep, nonlinear function approximator!

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. $Q_{\mathcal{GW}}$)

	Dataset	DGP	DWP $Q_{\mathcal{GW}}$	DWP $Q_{\text{A-}\mathcal{GW}}$	DWP $Q_{\text{AB-}\mathcal{GW}}$
LL	BOSTON	-2.43 ± 0.04	**-2.38 ± 0.04**	-2.39 ± 0.05	**-2.38 ± 0.04**
	CONCRETE	-3.13 ± 0.02	-3.13 ± 0.02	**-3.07 ± 0.02**	-3.08 ± 0.02
	ENERGY	-0.71 ± 0.03	-0.71 ± 0.03	**-0.70 ± 0.03**	**-0.70 ± 0.03**
	KIN8NM	1.38 ± 0.00	1.40 ± 0.01	**1.41 ± 0.01**	**1.41 ± 0.01**
	NAVAL	8.28 ± 0.04	8.17 ± 0.07	**8.40 ± 0.02**	8.10 ± 0.19
	POWER	-2.78 ± 0.01	-2.77 ± 0.01	**-2.76 ± 0.01**	**-2.76 ± 0.01**
	PROTEIN	-2.73 ± 0.01	-2.72 ± 0.01	-2.71 ± 0.01	**-2.70 ± 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	**-0.18 ± 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

Bayesian neural networks

special case of

deep Gaussian processes

rewrite prior in terms of kernels

Infinite-width limit

deep kernel **processes**

deep kernel **machines**

Part 2

Bayesian neural networks

special case of

deep Gaussian processes

rewrite prior in terms of kernels

deep kernel **processes**

Infinite-width limit

deep kernel **machines**

Why take an infinite width limit?

- Connect to theory of NNs (which is also infinite-width).
- Get rid of stochasticity (which is a pain in practice).
- Develop effective, practical deep kernel methods.

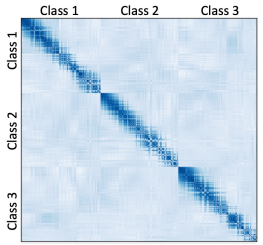
Taking the infinite-width limit of a DKP/DGP…

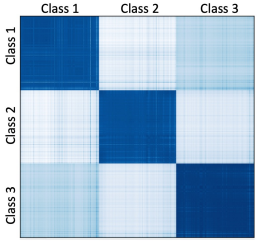
- VI in an infinite-width DKP/DGP.
- Need to be careful with the limit to make sure we keep representation learning

ELBO:

$$\mathcal{L}(\boldsymbol{G}_1, \ldots, \boldsymbol{G}_L) = \underbrace{\log \mathrm{P}(\mathbf{Y}|\, \boldsymbol{G}_L)}_{\text{likelihood}} - \sum_{\ell=1}^{L} \nu_\ell D_{\mathrm{KL}}(\underbrace{\mathcal{N}(0, \boldsymbol{G}_\ell)}_{\text{approx post}} \| \underbrace{\mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1}))}_{\text{prior}})$$

- Optimizes intermediate layer Gram matrices,
- Encourages good performance
- Keeps learned Gram matrix, $\boldsymbol{G}_\ell$, similar to NNGP Gram matrix, $\boldsymbol{K}(\boldsymbol{G}_{\ell-1})$



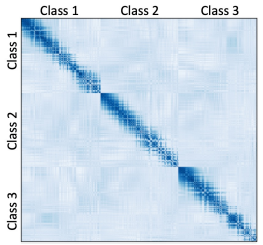


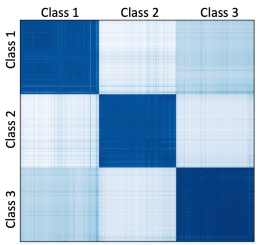
What is a deep kernel machine?

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

$$\mathcal{L}(\boldsymbol{G}_1, \ldots, \boldsymbol{G}_L) = \underbrace{\log \mathrm{P}(\mathbf{Y}|\, \boldsymbol{G}_L)}_{\text{likelihood}} - \sum_{\ell=1}^{L} \nu_\ell D_{\mathrm{KL}}(\underbrace{\mathcal{N}(0, \boldsymbol{G}_\ell)}_{\text{approx post}} \| \underbrace{\mathcal{N}(0, \boldsymbol{K}(\boldsymbol{G}_{\ell-1}))}_{\text{prior}})$$

- Optimizes intermediate layer Gram matrices,
- Encourages good performance
- Keeps learned Gram matrix, $\boldsymbol{G}_\ell$, similar to NNGP Gram matrix, $\boldsymbol{K}(\boldsymbol{G}_{\ell-1})$





Convolutional deep kernel machines are "kernel SOTA"

Table 4: Test accuracy of various kernel methods on CIFAR-10.

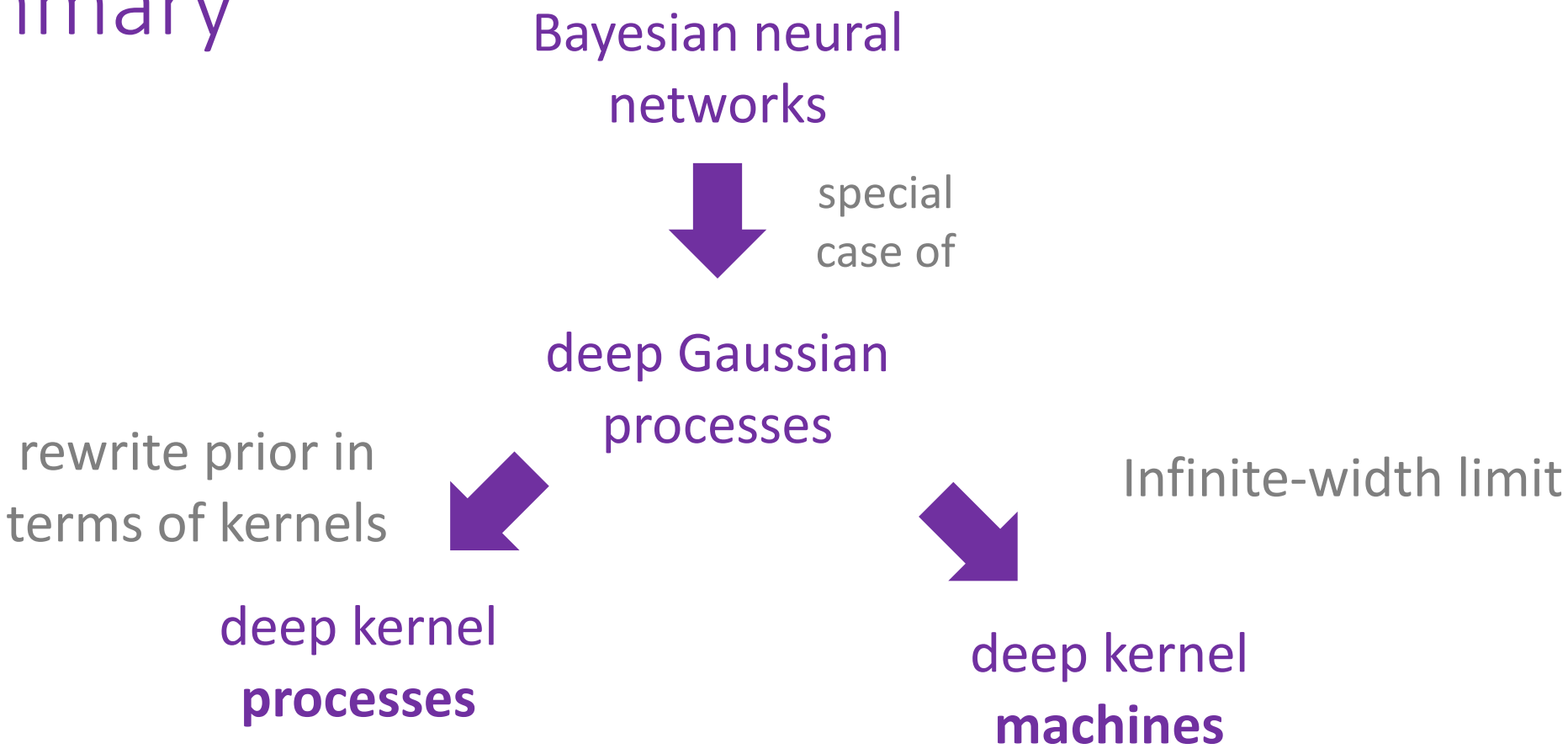
Paper	Method	CIFAR-10
This paper	**DKM**	**93.22%**
Novak et al. (2018)	NNGP-GAP	77.43%
Arora et al. (2019)	NNGP-GAP	83.75%
Lee et al. (2020)	NNGP-GAP-DA	84.8%
Li et al. (2019)	NNGP-LAP-flip	88.92%
Shankar et al. (2020)	Myrtle10	89.80%
Adlam et al. (2023)	Tuned Myrtle10 DA CG	91.2%

Edward Milsom

But how slow are DKMs? Surprisingly quick!

- We develop a novel inducing-point scheme…
- *…which was roughly same cost as training a standard CNN.*
- So the DKM was orders of magnitude faster than the kernel methods in the table…
- …but still slow compared to CNNs, (3 days/1200 epochs on 1 GPU for this result)
- But *lots* of structure in the parameter space: we're working on a natural gradient method that should converge *much* faster.

Summary


Bayesian neural networks
special case of
deep Gaussian processes
rewrite prior in terms of kernels
Infinite-width limit
deep kernel **processes**
deep kernel **machines**

Deep kernel landscape + our priorities

Our priorities

- Library: user friendly, ”drop in” replacement for NNs.
- Deep kernel transformers
- Speed up (natural gradients)

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*” arXiv

Appendix slides

We get a deep kernel machine by taking an infinite-width limit of a DGP

- True posterior over features becomes multivariate Gaussian [1]

$$P(\boldsymbol{F}_1, \ldots, \boldsymbol{F}_L | \boldsymbol{X}, \boldsymbol{Y}) = \prod_{\ell=1}^{L} \sum_{\lambda=1}^{N_\ell} \mathcal{N}(\boldsymbol{f}_\lambda^\ell; 0, \boldsymbol{G}_\ell^*)$$

- We choose a family of approximate posteriors capturing the true posterior:

$$Q(\boldsymbol{F}_1, \ldots, \boldsymbol{F}_L) = \prod_{\ell=1}^{L} \sum_{\lambda=1}^{N_\ell} \mathcal{N}\left(\boldsymbol{f}_\lambda^\ell; 0, \boldsymbol{G}_\ell\right)$$

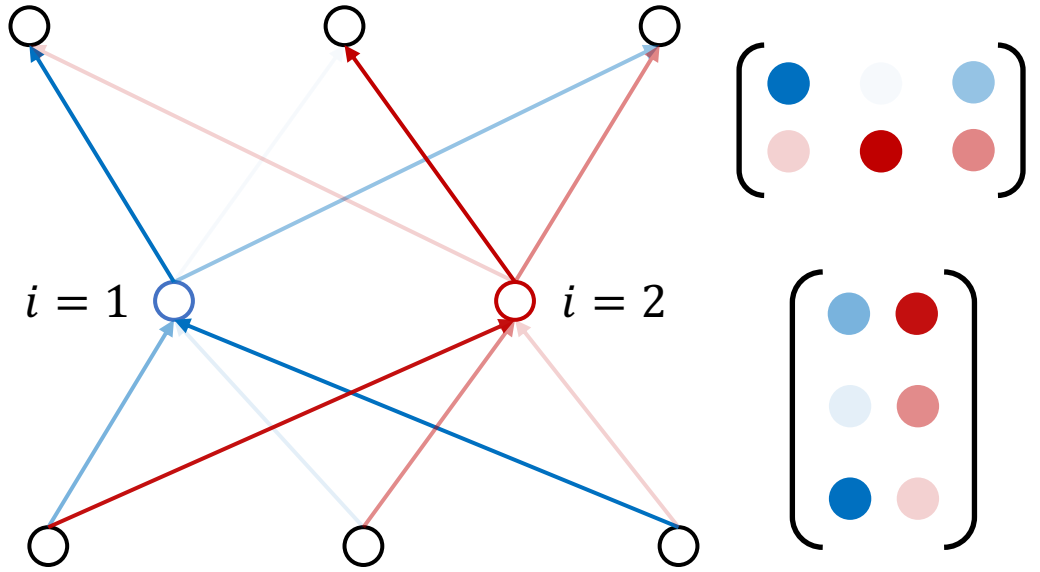
- Gram matrices, $\boldsymbol{G}_1, \ldots \boldsymbol{G}_L$, are the same kind-of-thing as in deep kernel process!

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

- But here, Gram matrices appear as parameters of approximate posterior
- So to find the Gram matrices, we optimize the ELBO!

[1] Yang, Robeyns, Milsom, Anson, Schoots, Aitchison “A theory of representation learning gives a deep generalisation of kernel methods” ICML (2023)

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!