BSc Informatics & Telecommunications

BSc Informatics & Telecommunications

MSc Informatics & AI

BSc Informatics & Telecommunications

MSc Informatics & AI

PhD Machine Learning

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- **Researcher ML & Robotics**

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- Researcher ML & Robotics
- **Start-up**

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- Researcher ML & Robotics
- Start-up

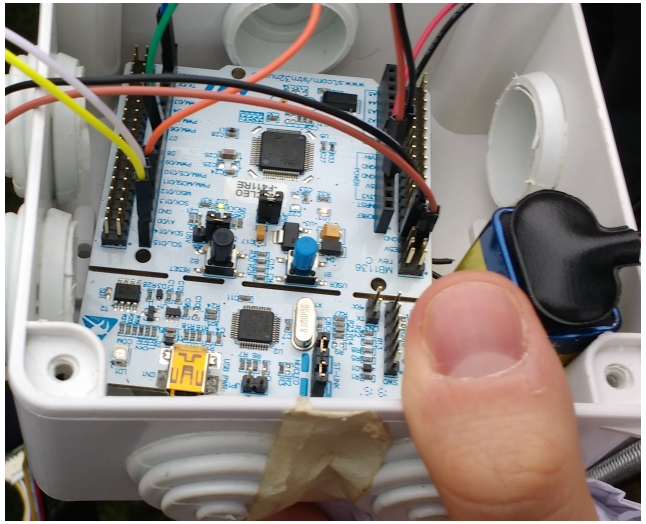
ATI Fellowship

- BSc Informatics & Telecommunications
- MSc Informatics & AI
- PhD Machine Learning
- Researcher ML & Robotics
- Start-up

  ATI Fellowship
- **Amazon**

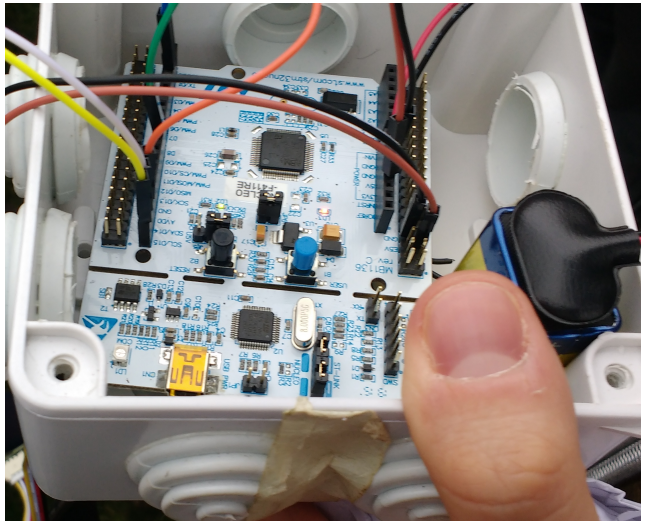
Data Science Africa

Data Science Africa

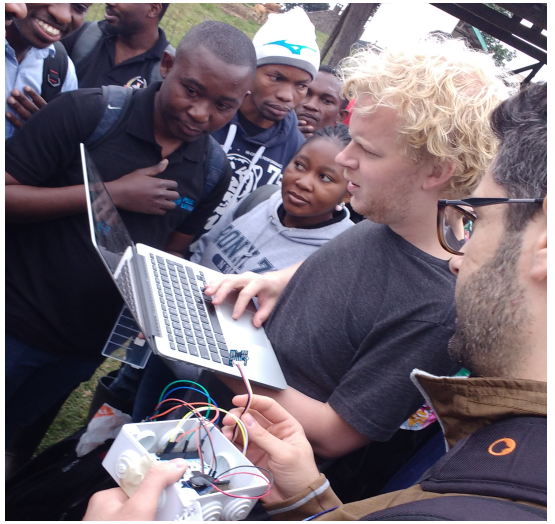


ARM Mbed

Data Science Africa



ARM Mbed



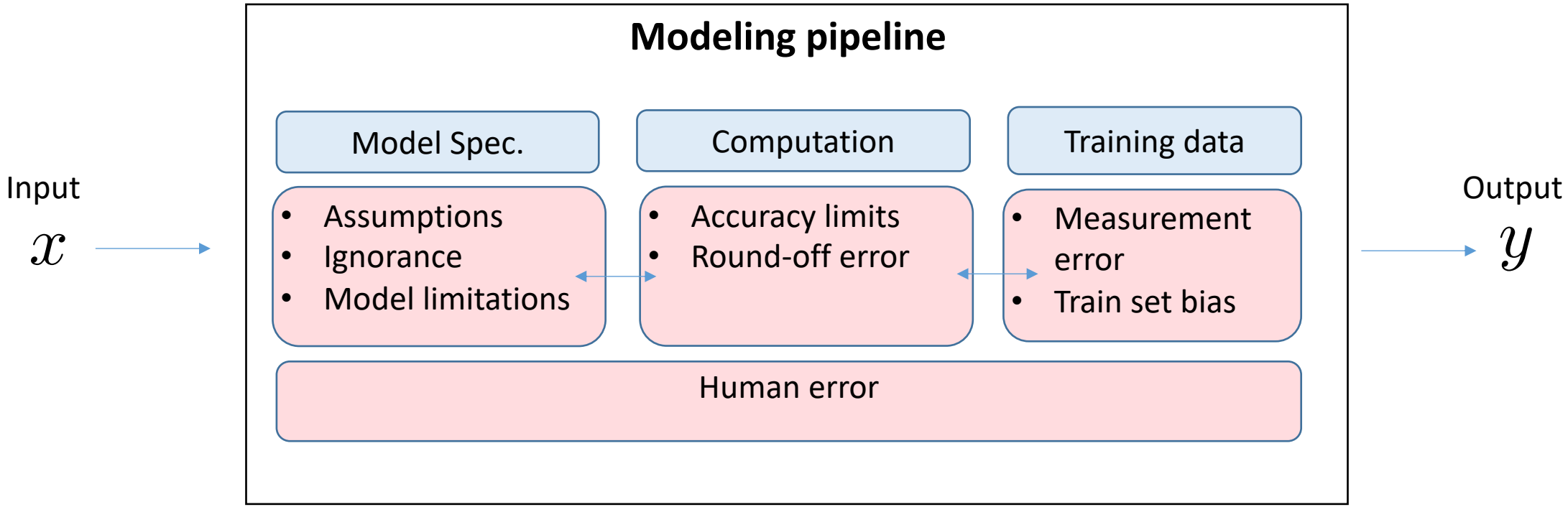
Gaussian process

Topics covered

- Why we want to model uncertainty, and how.
  - Bayesian neural networks
  - (Deep) Gaussian processes
- Challenges / computational considerations
- Efficient implementation & the (potential) role of hardware
- The best (?) of neural network and Gaussian process world

Uncertainty in every part of the system

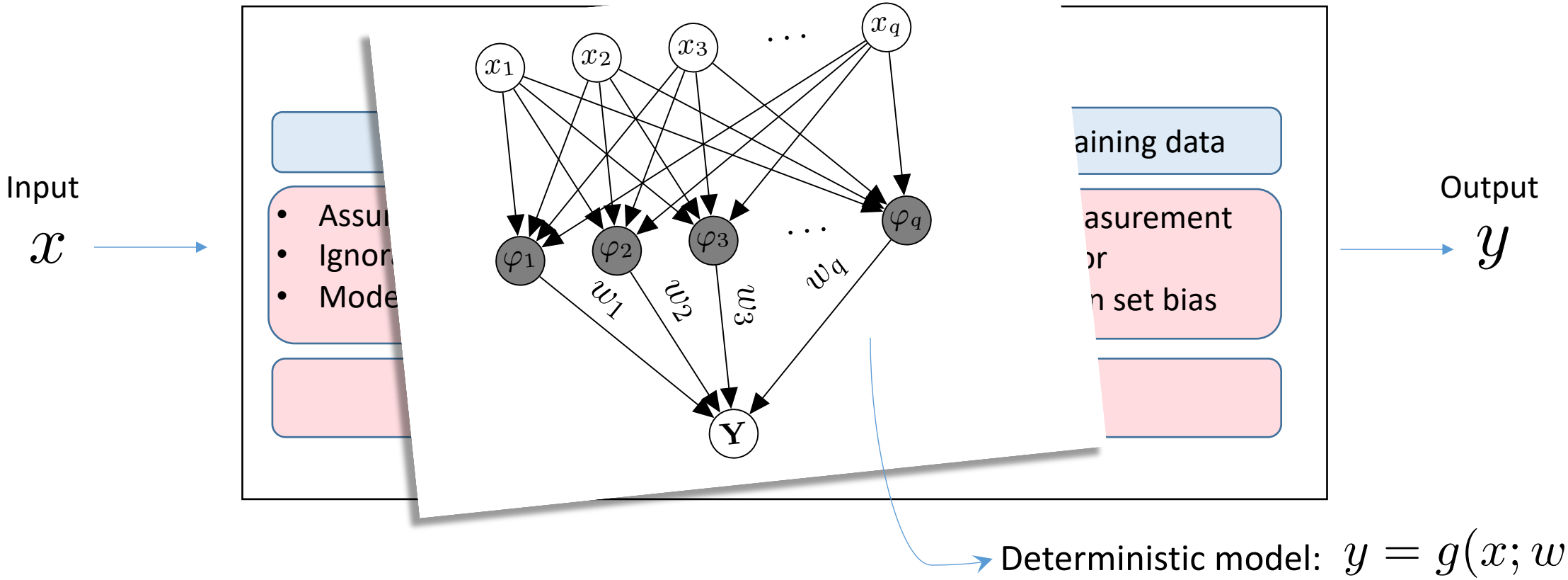
- Can we account for uncertainty in training and predictions?



Uncertainty = errors + lack of knowledge.

Uncertainty in every part of the system

- Can we account for uncertainty in training and predictions?



Uncertainty = errors + lack of knowledge.

Need for uncertainty

Quantifying what “you don’t know”:

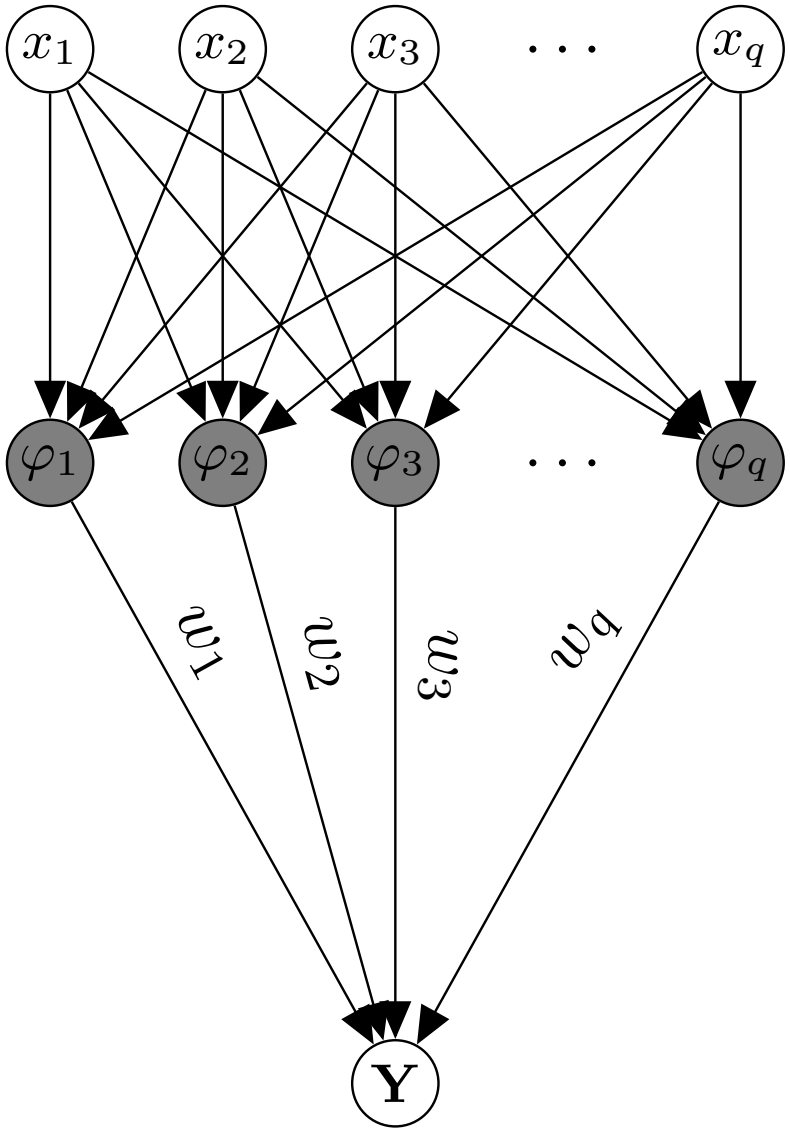
- Acts as a regularizer during training
- Helps to avoid overfitting
- Allows us to inspect the system & collect better data
- Provides uncertainty in predictions,
  useful for downstream tasks (health; self-driving cars)

→ *Better human-computer collaboration*

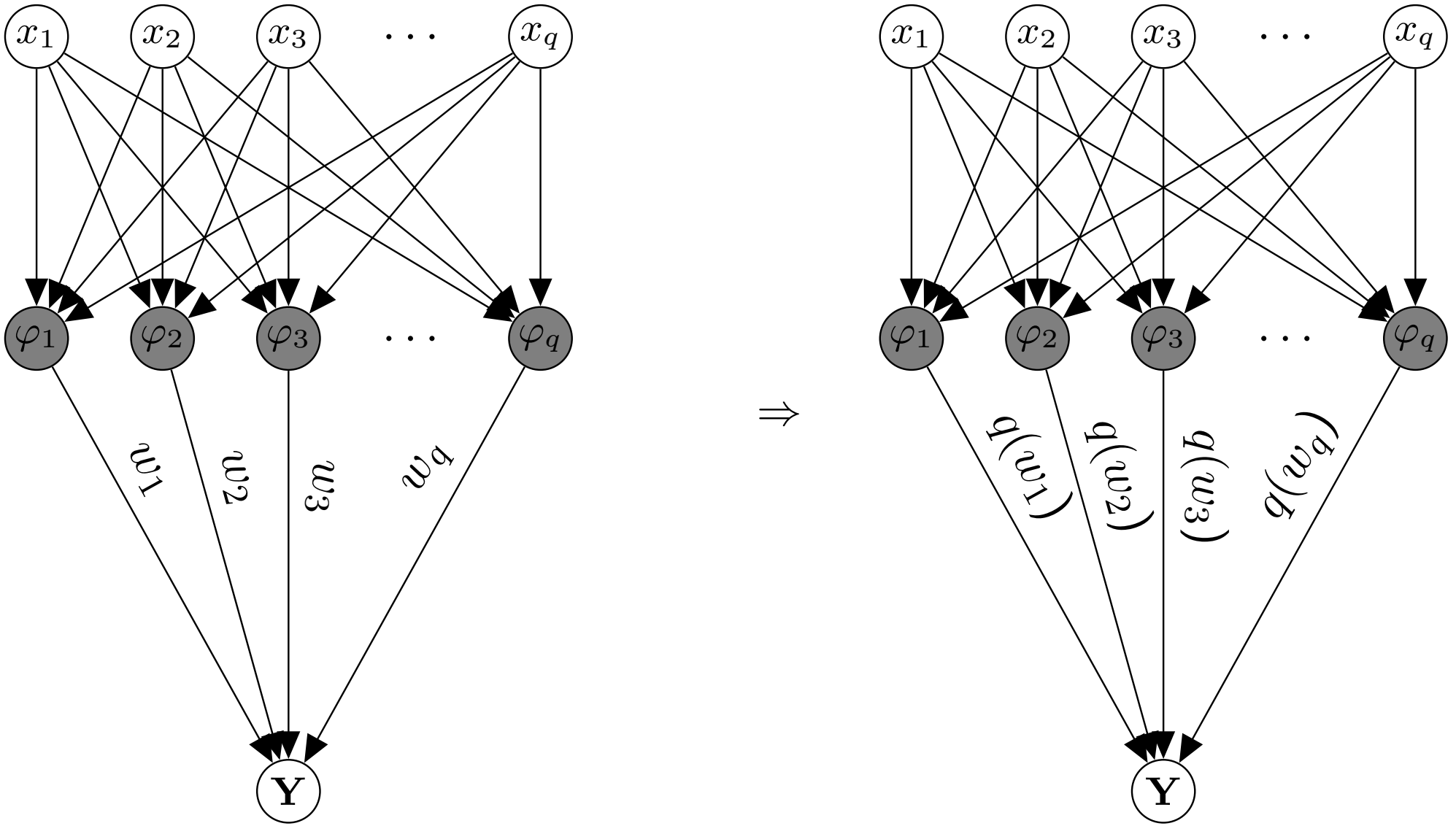
Probabilistic deep learning

See also: http://adamian.github.io/talks/Damianou_deep_learning_rss_2018.pdf

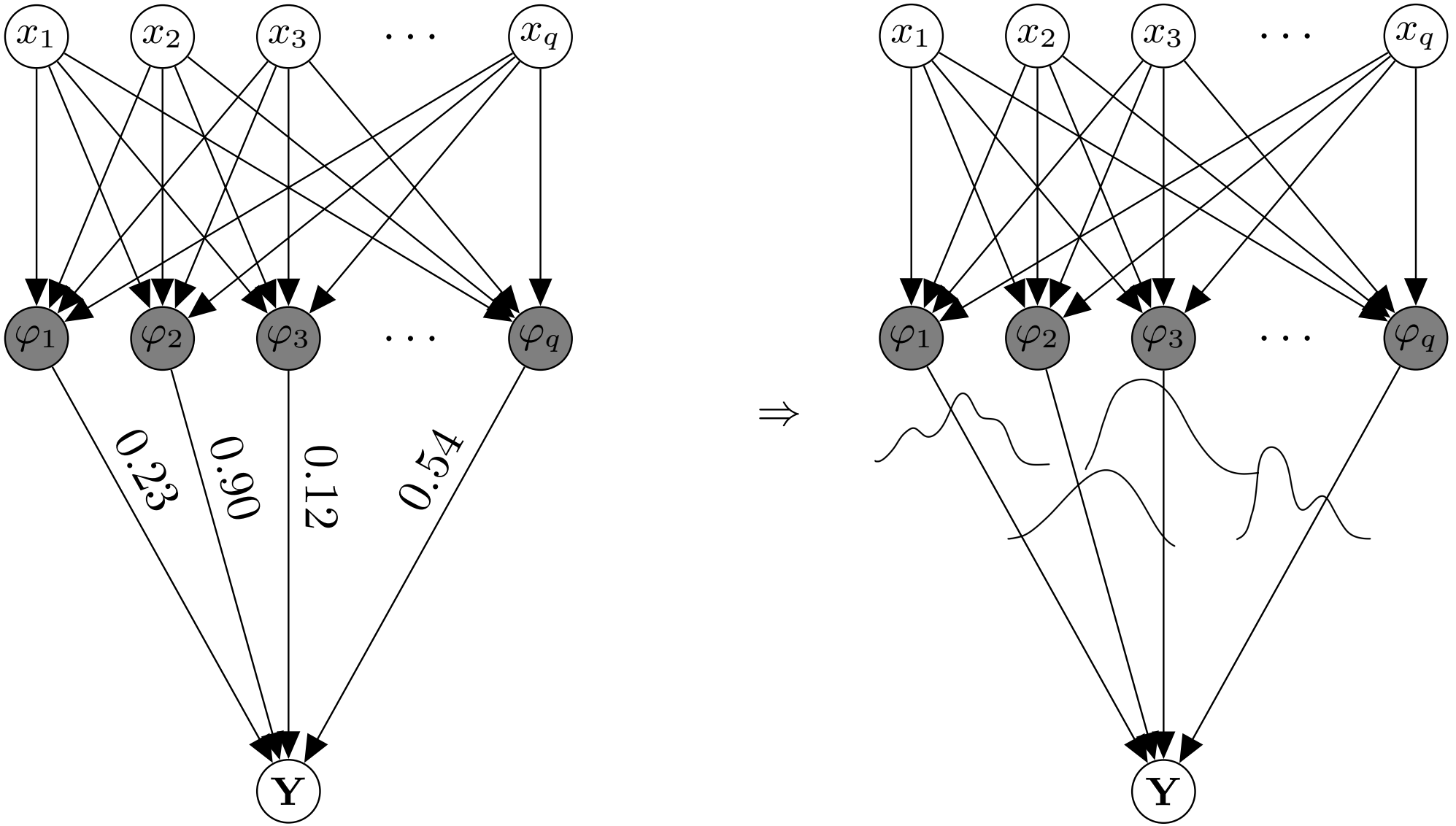
A standard neural network



BNN with priors on its weights



BNN with priors on its weights



Bayesian uncertainty modeling

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\textit{parametric}: \quad \mathbf{y}_* = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D})$$

$$\textit{Bayesian parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*) = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D})$$

Bayesian uncertainty modeling

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\textit{parametric}: \quad \mathbf{y}_* = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D})$$

$$\textit{Bayesian parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*) = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D})$$

Bayesian uncertainty modeling

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

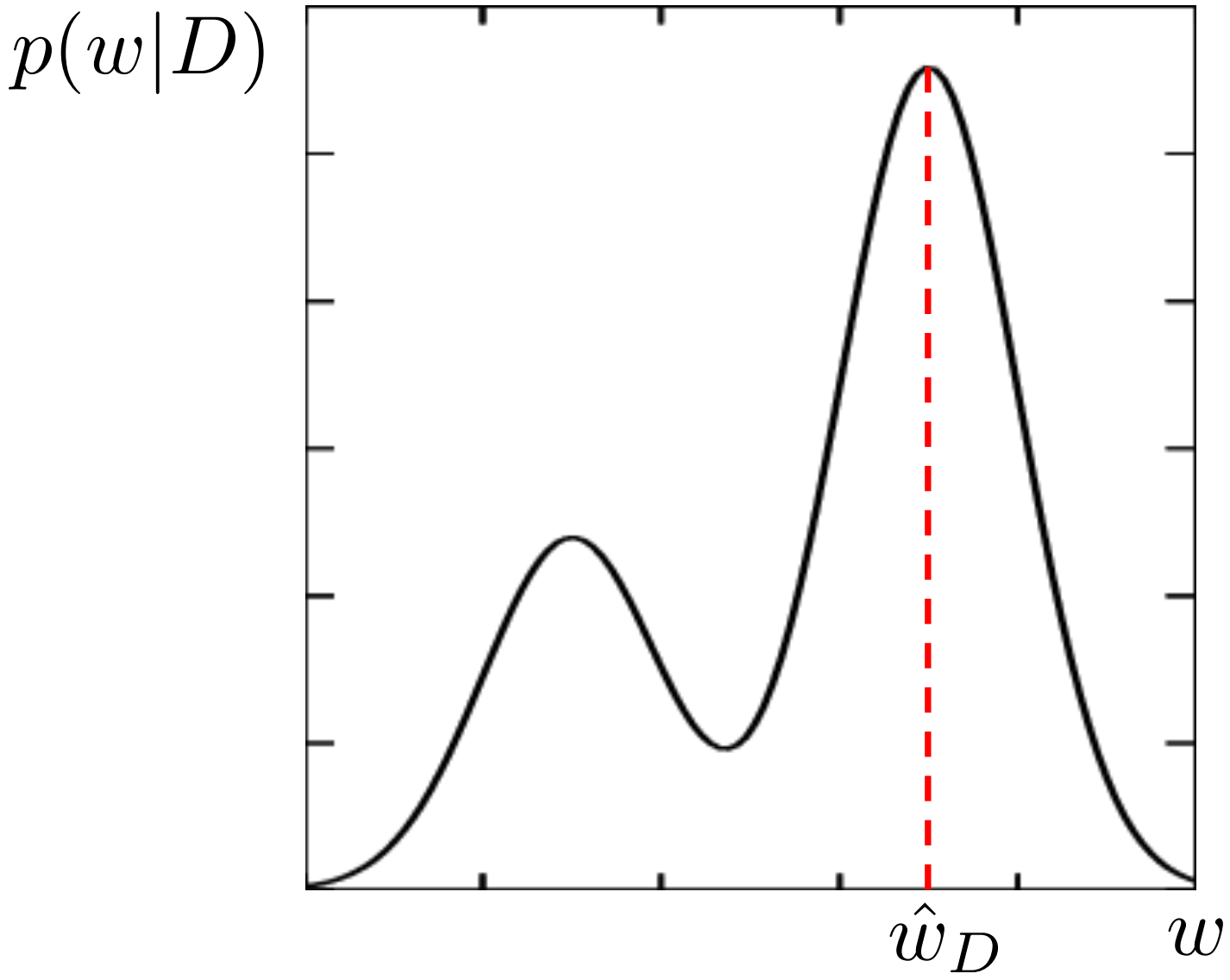
$$\textit{parametric}: \quad \mathbf{y}_* = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D})$$

$$\textit{Bayesian parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*) = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D})$$

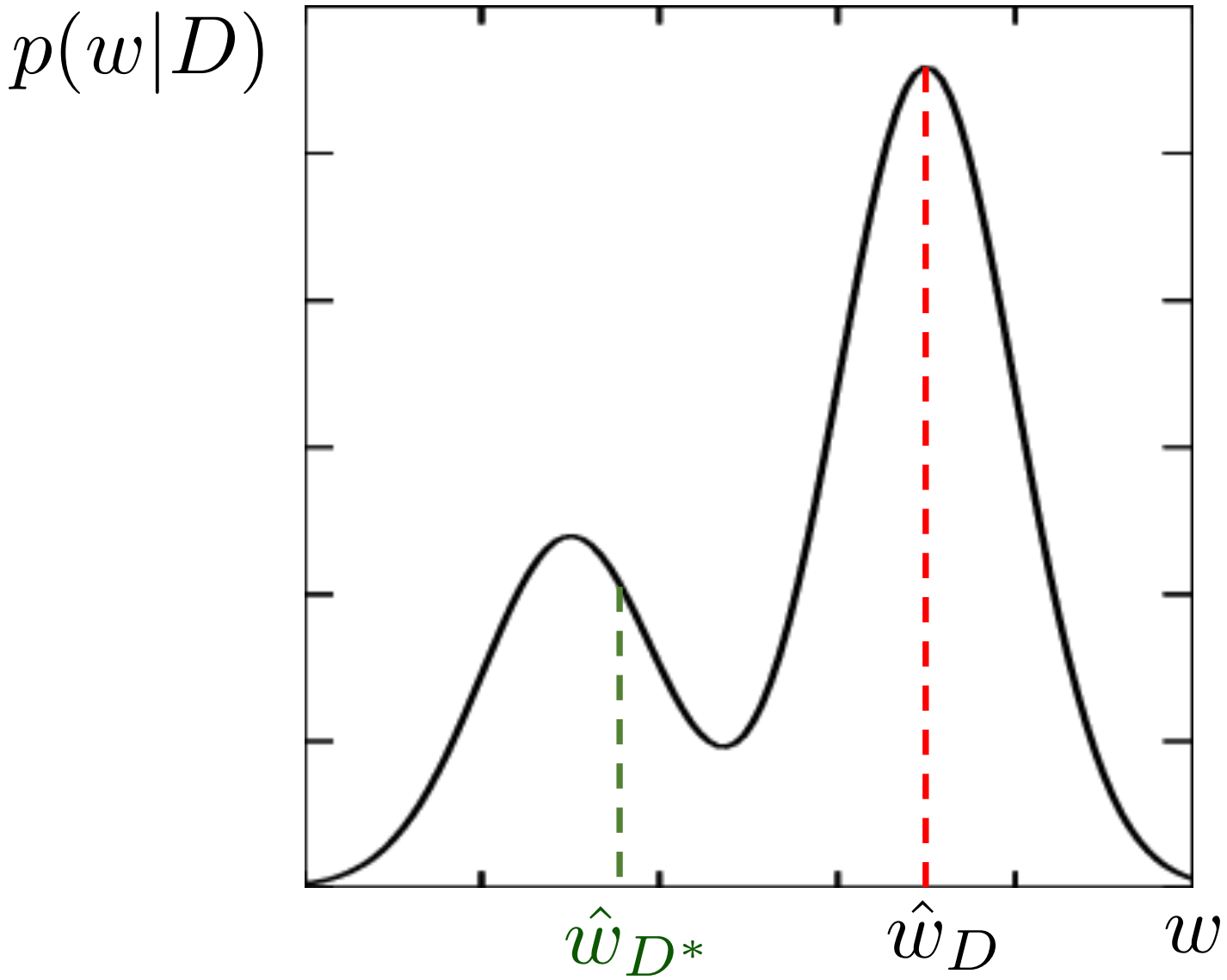
$p(\mathbf{w}|\mathbf{D})$ is the *posterior* obtained from **Bayes rule**:

$$p(\mathbf{w}|\mathbf{D}) = \frac{p(\mathbf{D}|\mathbf{w})p(\mathbf{w})}{\int_\mathbf{w} p(\mathbf{D}|\mathbf{w})p(\mathbf{w})}$$

Single point vs full posterior



Single point vs full posterior



$$p(\mathbf{w}|\mathbf{D}) = \frac{p(\mathbf{D}|\mathbf{w})p(\mathbf{w})}{\int_{\mathbf{w}} p(\mathbf{D}|\mathbf{w})p(\mathbf{w})} \rightarrow \textit{Intractable!}$$

Prediction with sampling

In practice we predict using sampling:

$$
\begin{aligned}
p(\mathbf{y}_*|\mathbf{x}_*) &= \int_{\mathbf{w}} p(\mathbf{y}_*|\phi(\mathbf{x}_*;\mathbf{w}))p(\mathbf{w}|\mathbf{D}) \\
&\approx \int_{\mathbf{w}} p(\mathbf{y}_*|\phi(\mathbf{x}_*;\mathbf{w}))q(\mathbf{w}) \\
&\approx \frac{1}{M}\sum_{m=1}^{M} p(\mathbf{y}_*|\phi(\mathbf{x}_*;\mathbf{w}_m))
\end{aligned}
$$

- Bayesian neural networks are yet to give state-of-the-art results
- Costly training to find $q(\mathbf{w})$
- Costly prediction to sample $\mathbf{w}_m$ and average

Prediction with sampling

In practice we predict using sampling:

$$
\begin{aligned}
p(\mathbf{y}_*|\mathbf{x}_*) &= \int_{\mathbf{w}} p(\mathbf{y}_*|\phi(\mathbf{x}_*;\mathbf{w}))p(\mathbf{w}|\mathbf{D}) \\
&\approx \int_{\mathbf{w}} p(\mathbf{y}_*|\phi(\mathbf{x}_*;\mathbf{w}))q(\mathbf{w}) \\
&\approx \frac{1}{M}\sum_{m=1}^{M} p(\mathbf{y}_*|\phi(\mathbf{x}_*;\mathbf{w}_m))
\end{aligned}
$$

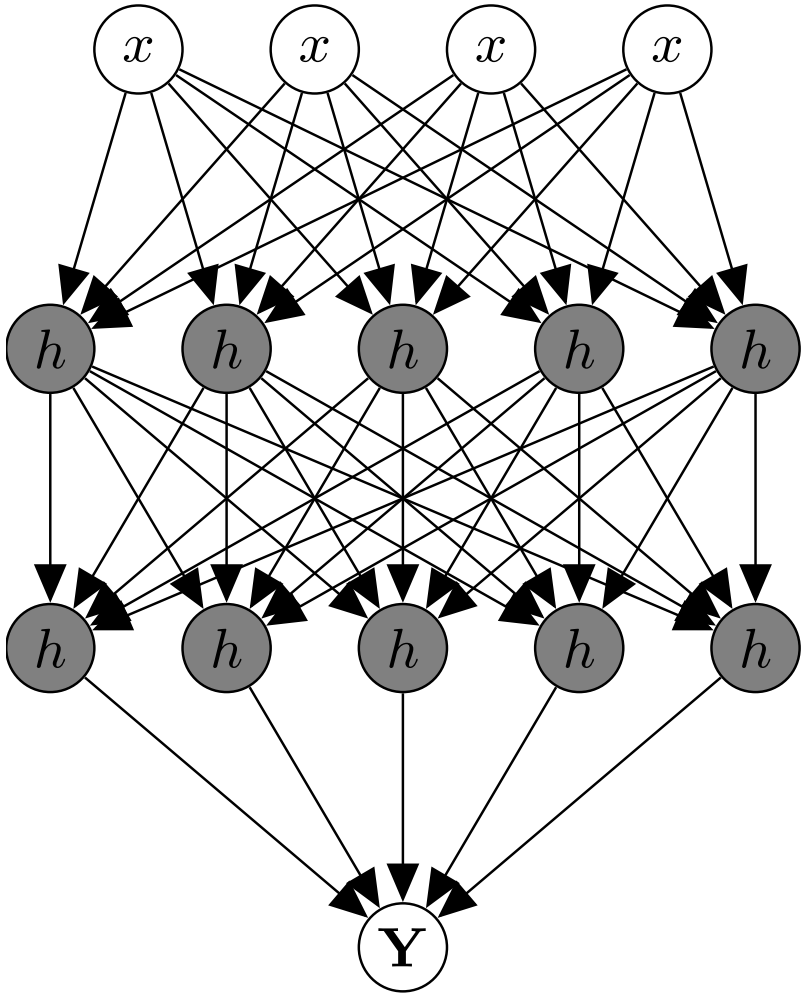
- Bayesian neural networks are yet to give state-of-the-art results
- Costly training to find $q(\mathbf{w})$
- Costly prediction to sample $\mathbf{w}_m$ and average

Energy-efficient predictions using uncertainty

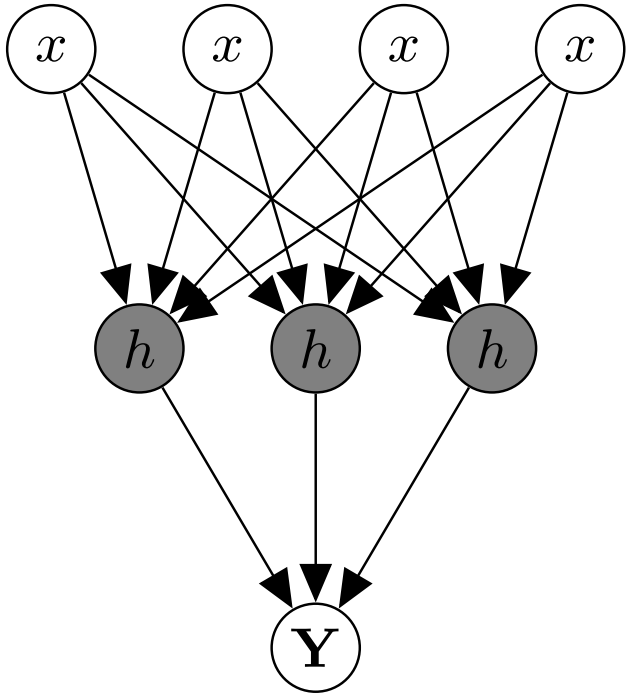
Some (I believe) relatively unexplored directions:

- **Compress** and naturally we then have fewer weights (for BNN).
- **Mixed-precision**: Compute the bits needed for each weight *[Louizos et al.]* within a posterior.
- **Mixture of experts ensemble:** "bad" experts get allocated fewer bit-precision (inspired by MC-dropout).
- **Bayesian student-teacher distillation** *[Ahn, Hu, Damianou, Lawrence, Dai]* for BNNs.

Bayesian student-teacher *[Ahn, Hu, Damianou, Lawrence, Dai 2019]*



Mutual Information



- Distribution on parameters induces uncertainty in function: $f(x; p(w))$

- Distribution on parameters induces uncertainty in function: $f(x; p(w))$
- Can we model uncertainty directly in the function space?

Gaussian processes

See also: adamian.github.io/talks/Damianou_GP_tutorial.html

Bayesian uncertainty modeling

Given training data $\mathbf{D}$, predict test outputs $\mathbf{y}_*$ from test inputs $\mathbf{x}_*$:

$$\textit{parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*, \hat{\mathbf{w}}_\mathbf{D}) = \phi(\mathbf{x}_*; \hat{\mathbf{w}}_\mathbf{D})$$

$$\textit{Bayesian parametric}: \quad p(\mathbf{y}_*|\mathbf{x}_*) = \int_\mathbf{w} p(\mathbf{y}_*|\phi(\mathbf{x}_*; \mathbf{w}))p(\mathbf{w}|\mathbf{D})$$

$$\textit{Gaussian process}: \quad p(\mathbf{y}_*|\mathbf{x}_*) = \int_{\mathbf{f}_*} p(\mathbf{y}_*|f_*(\mathbf{x})) \quad p(f_*|\mathbf{D})$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$p(\underbrace{f_1, f_2, \cdots, f_n}_{\mathbf{f}_n}, \underbrace{f_{n+1}, f_{n+2}, \cdots, f_s}_{\mathbf{f}_s}) = p(\mathbf{f}_n, \mathbf{f}_s) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}).$$

with:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_n \\ \boldsymbol{\mu}_s \end{bmatrix} \text{ and } \mathbf{K} = \begin{bmatrix} \mathbf{K}_{nn} & \mathbf{K}_{ns} \\ \mathbf{K}_{sn} & \mathbf{K}_{ss} \end{bmatrix}$$

Marginalisation:

$$p(\mathbf{f}_n) = \int_{\mathbf{f}_s} p(\mathbf{f}_n, \mathbf{f}_s) \mathrm{d}\mathbf{f}_s = \mathcal{N}(\boldsymbol{\mu}_n, \mathbf{K}_{nn})$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$p(\underbrace{f_1, f_2, \cdots, f_n}_{\mathbf{f}_n}, \underbrace{f_{n+1}, f_{n+2}, \cdots, f_s}_{\mathbf{f}_s}) = p(\mathbf{f}_n, \mathbf{f}_s) \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}).$$

with:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_n \\ \boldsymbol{\mu}_s \end{bmatrix} \text{ and } \mathbf{K} = \begin{bmatrix} \mathbf{K}_{nn} & \mathbf{K}_{ns} \\ \mathbf{K}_{sn} & \mathbf{K}_{ss} \end{bmatrix}$$

Marginalisation:

$$p(\mathbf{f}_n) = \int_{\mathbf{f}_s} p(\mathbf{f}_n, \mathbf{f}_s) \mathrm{d}\mathbf{f}_s = \mathcal{N}(\boldsymbol{\mu}_n, \mathbf{K}_{nn})$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$p(\underbrace{f_1, f_2, \cdots, f_n}_{\mathbf{f}_n}, \underbrace{f_{n+1}, f_{n+2}, \cdots, f_\infty}_{\mathbf{f}_\infty}) = p(\mathbf{f}_n, \mathbf{f}_\infty) \sim \mathcal{GP}(\boldsymbol{\mu}_\infty, \mathbf{K}_\infty).$$

with:

$$\boldsymbol{\mu}_\infty = \begin{bmatrix} \boldsymbol{\mu}_n \\ \cdots \end{bmatrix} \quad \text{and} \quad \mathbf{K}_\infty = \begin{bmatrix} \mathbf{K}_{nn} & \cdots \\ \cdots & \cdots \end{bmatrix}$$

Marginalisation:

$$p(\mathbf{f}_n) = \int_{\mathbf{f}_\infty} p(\mathbf{f}_n, \mathbf{f}_\infty) \mathrm{d}\mathbf{f}_\infty = \mathcal{N}(\boldsymbol{\mu}_n, \mathbf{K}_{nn})$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$\boldsymbol{\mu}_\infty = \begin{bmatrix} \boldsymbol{\mu}_n \\ \cdots \end{bmatrix} \quad \text{and} \quad \mathbf{K}_\infty = \begin{bmatrix} \mathbf{K}_{nn} & \cdots \\ \cdots & \cdots \end{bmatrix}$$

Infinite model... but we *always* work with finite sets!

Multivariate Gaussian for all $f_i = f(\mathbf{x}_i)$:

$$\boldsymbol{\mu}_\infty = \begin{bmatrix} \boldsymbol{\mu}_n \\ \dots \end{bmatrix} \quad \text{and} \quad \mathbf{K}_\infty = \begin{bmatrix} \mathbf{K}_{nn} & \dots \\ \dots & \dots \end{bmatrix}$$

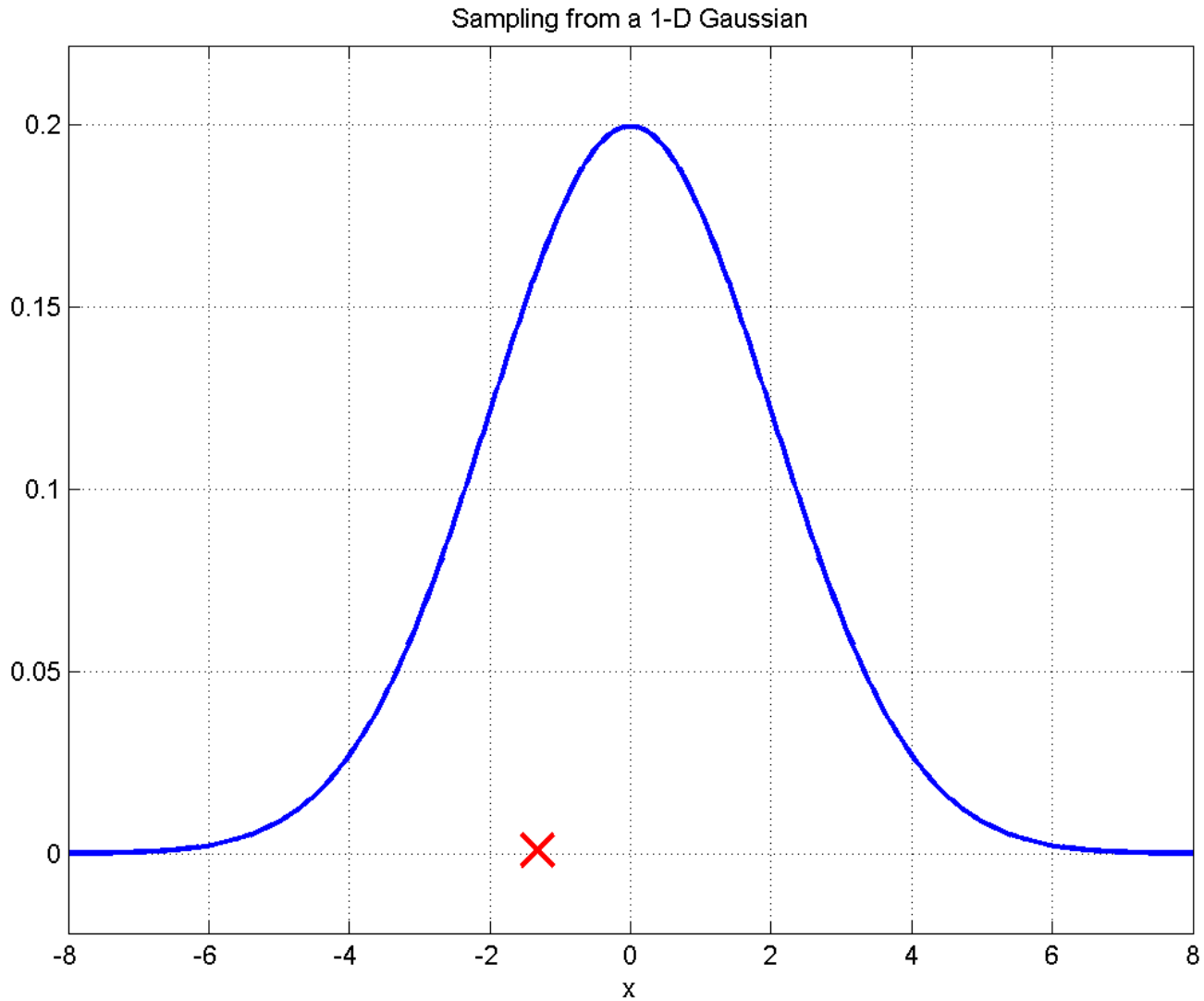
$$\boldsymbol{\mu}_i = \mu(\mathbf{x}_i)$$

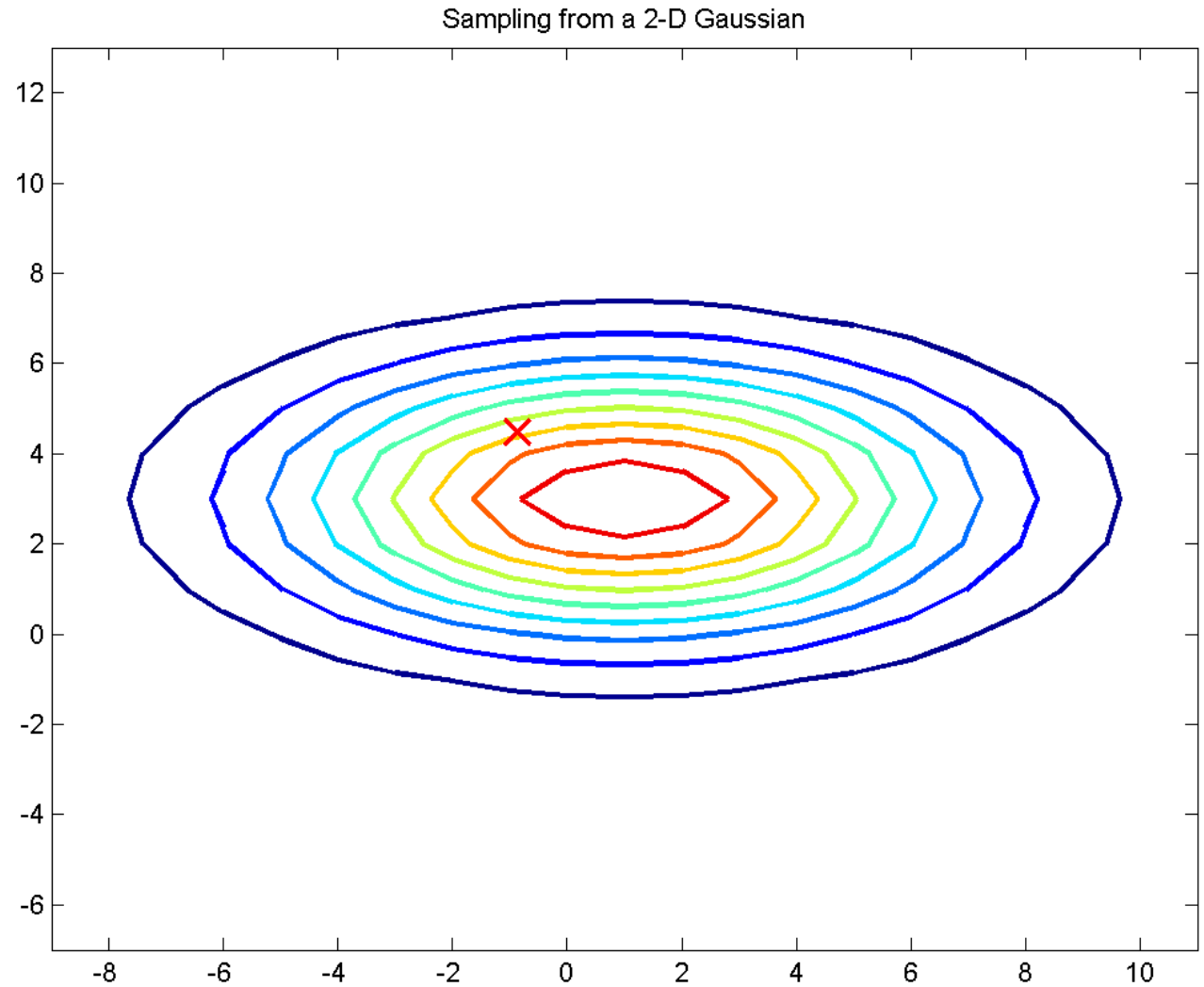
$$\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j; \theta)$$

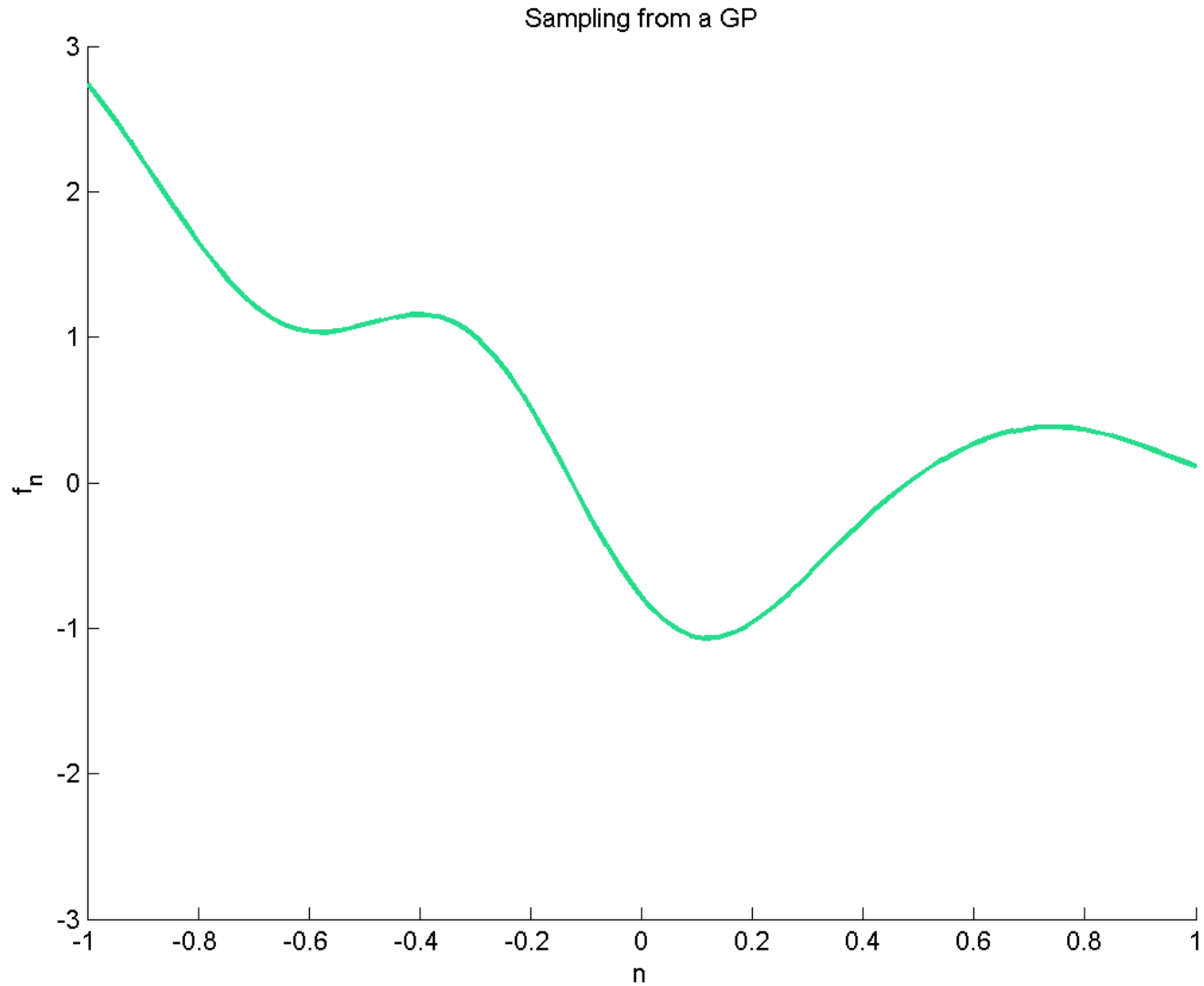
Train a GP means fitting θ

Introducing Gaussian Processes:

- A Gaussian distribution depends on a mean and a covariance matrix.
- A Gaussian process depends on a mean and a covariance function.


Sampling from a 1-D Gaussian
0.2
0.15
0.1
0.05
0
-8
-6
-4
-2
0
2
4
6
8
x

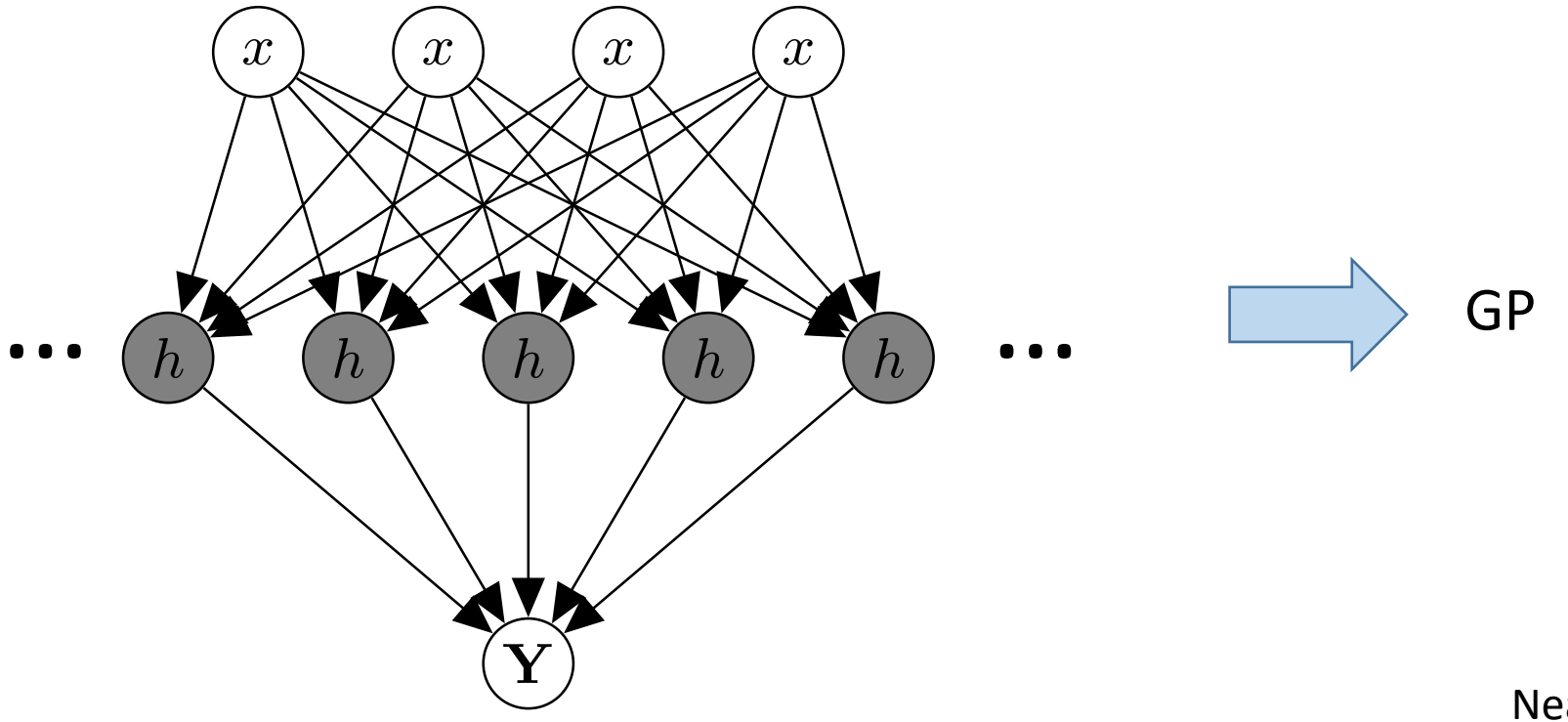

Sampling from a 2-D Gaussian
12
10
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
0
2
4
6
8
10


Sampling from a GP
3
2
1
0
-1
-2
-3
f_n
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
n

GP as the limit of a Bayesian NN

- Limit of infinite num. parameters:
  Optimization of finite num. parameters → Integration of infinite model

- *Closed form* solution for modelled function class!



Neal 1994; Williams 1997

Benefits of modeling in function space

- Closed form integration corresponds to “perfect” optimization.
- Easy to add prior knowledge/assumptions.
- Intuitive/interpretable modeling.
- Well-calibrated uncertainty.

Challenges #1: Computational

- **Training is slow**:

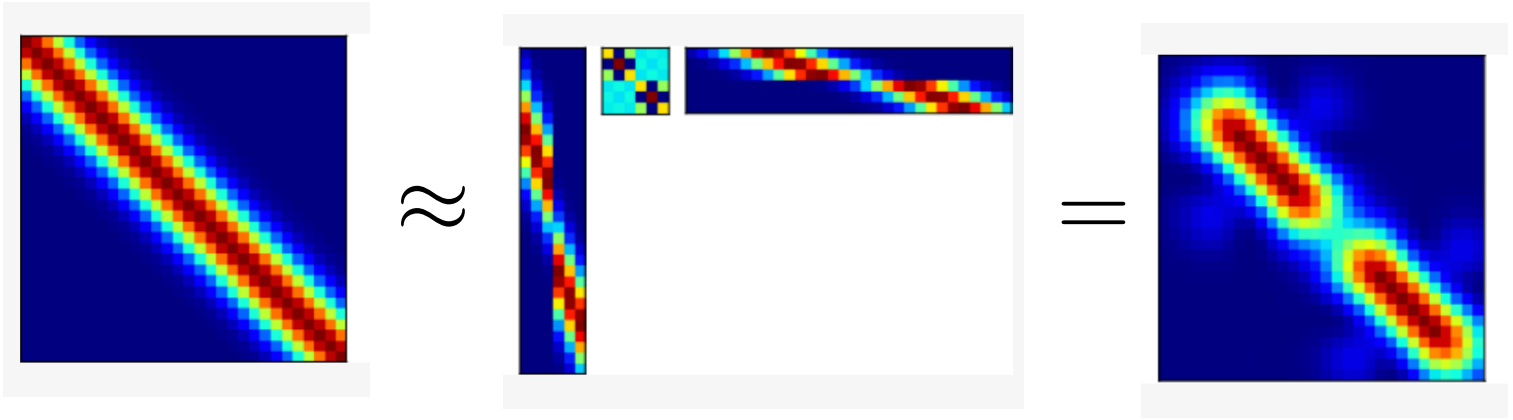
$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K}_{nn}) \propto e^{-\frac{1}{2}\mathbf{f}^\top \mathbf{K}_{nn}^{-1}\mathbf{f}}$$

- **Prediction is slow**:

$$\begin{aligned} p(f_*|\mathbf{x}_*) &= \mathcal{N}(f_*|\boldsymbol{\mu}_*, \mathbf{K}_*)) \\ \boldsymbol{\mu}_* &= \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{f} \\ \mathbf{K}_* &= k_{**} - \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{k}_{n*} \end{aligned}$$

- **Numerical instabilities** (requires high bit precision)

Distributed computations with GPU acceleration

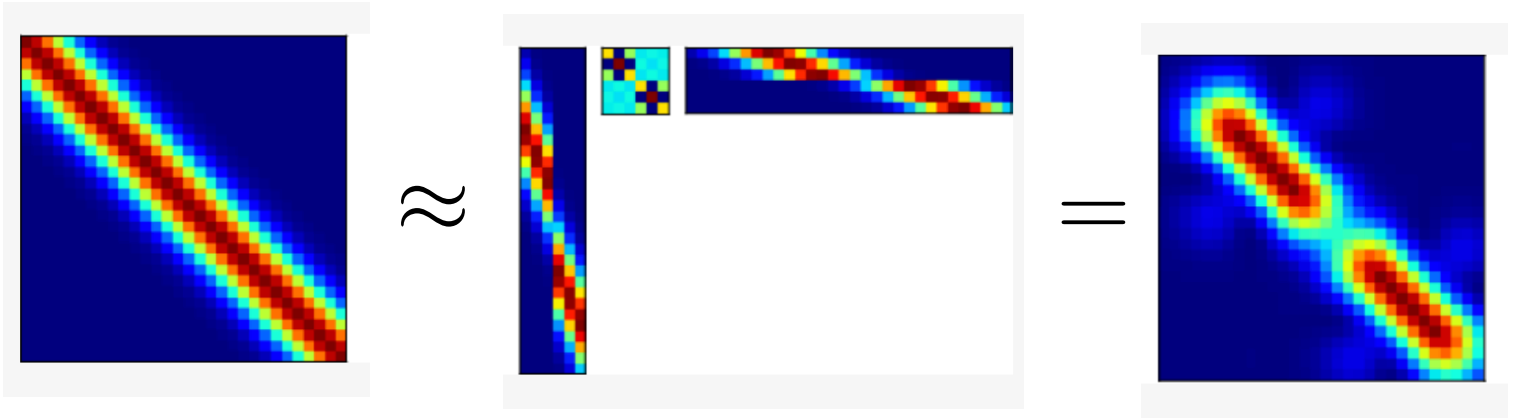


$$\mathbf{K}_{nn} \approx \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}$$

- Inversion of a much smaller matrix
- $\mathbf{K}_{nm}\mathbf{K}_{mn} = \sum_i \mathbf{K}_{mi}\mathbf{K}_{im}$ *Parallel computations!*

[Dai, Damianou, Hensman, Lawrence, "GP models with Parallelization and GPU acceleration", 2014]

Distributed computations with GPU acceleration

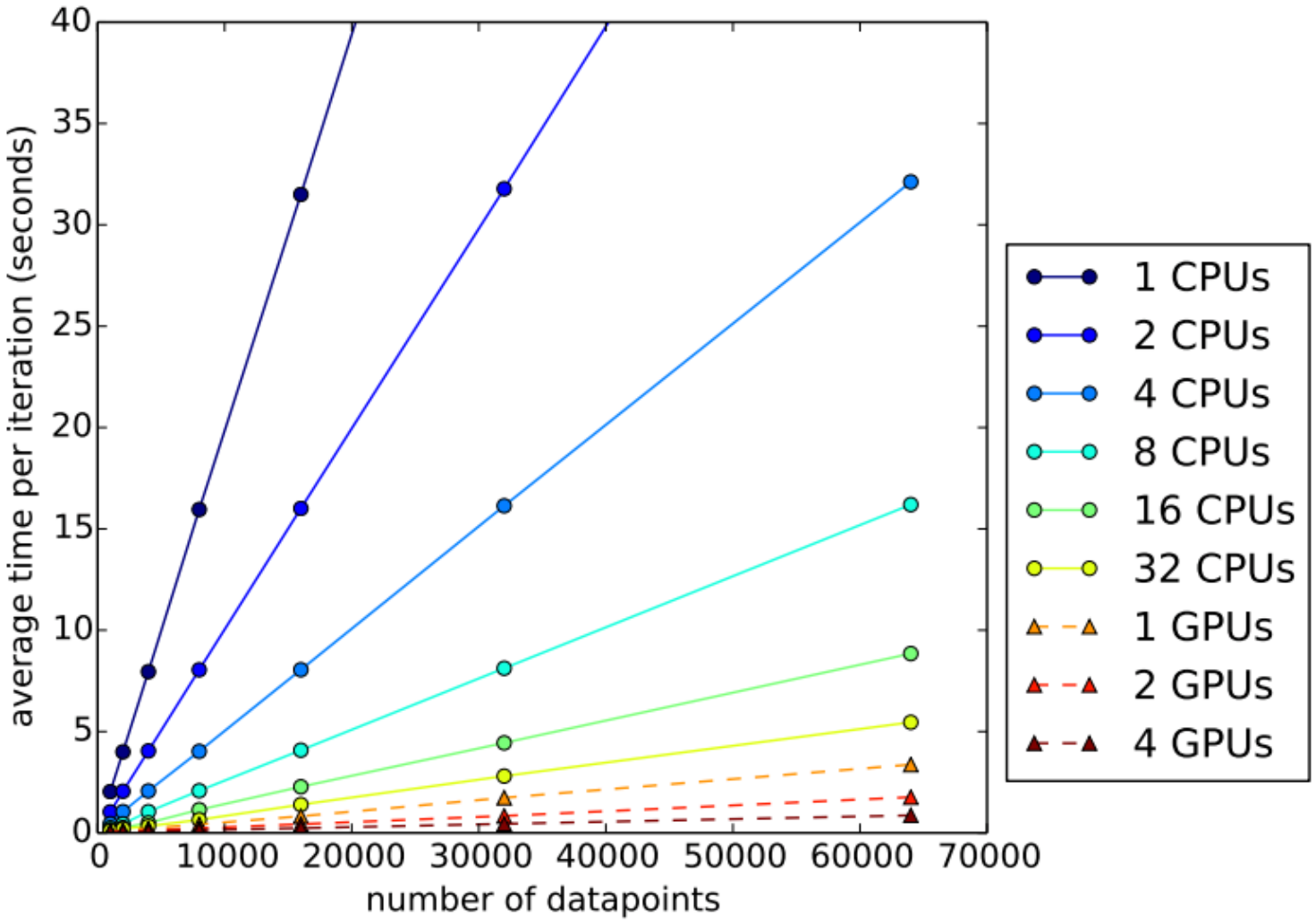


$$\mathbf{K}_{nn} \approx \mathbf{K}_{nm}\mathbf{K}_{mm}^{-1}\mathbf{K}_{mn}$$

- Inversion of a much smaller matrix
- $\mathbf{K}_{nm}\mathbf{K}_{mn} = \sum_i \mathbf{K}_{mi}\mathbf{K}_{im}$ *Parallel computations!*

[Dai, Damianou, Hensman, Lawrence, "GP models with Parallelization and GPU acceleration", 2014]

Linear scaling (wrt n) for training!



Fast & economical predictions

$$\boldsymbol{\mu}_* = \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{k}_{n*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has solution $\mathbf{a}^* = \mathbf{K}^{-1}\mathbf{f}$

- For t iterations the covariance expressed in terms of matrices with t columns.
- $O(tm)$ for storage and computation for variance (can reduce to $O(t)$ if dimension m forms a grid).

Fast & economical predictions

$$\boldsymbol{\mu}_* = \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{k}_{n*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

  $g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f}$ has solution $\mathbf{a}^* = \mathbf{K}^{-1}\mathbf{f}$

- For t iterations the covariance expressed in terms of matrices with t columns.
- $O(tm)$ for storage and computation for variance (can reduce to $O(t)$ if dimension m forms a grid).

Fast & economical predictions

$$\boldsymbol{\mu}_* = \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{k}_{n*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization: *[Pleiss et al. 2018]*

$$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f} \text{ has solution } \mathbf{a}^* = \mathbf{K}^{-1}\mathbf{f}$$

- For t iterations the covariance expressed in terms of matrices with t columns.
- $O(tm)$ for storage and computation for variance (can reduce to $O(t)$ if dimension m forms a grid).

Fast & economical predictions

$$\boldsymbol{\mu}_* = \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{f}$$
$$k_* = k_{**} - \mathbf{k}_{*n}\mathbf{K}_{nn}^{-1}\mathbf{k}_{n*}$$

- Observation 1: Difficult parts can be cached.
- Observation 2: We can turn inversion into optimization:

[Pleiss et al. 2018]

$$g(\mathbf{a}) = \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{f} \text{ has solution } \mathbf{a}^* = \mathbf{K}^{-1}\mathbf{f}$$

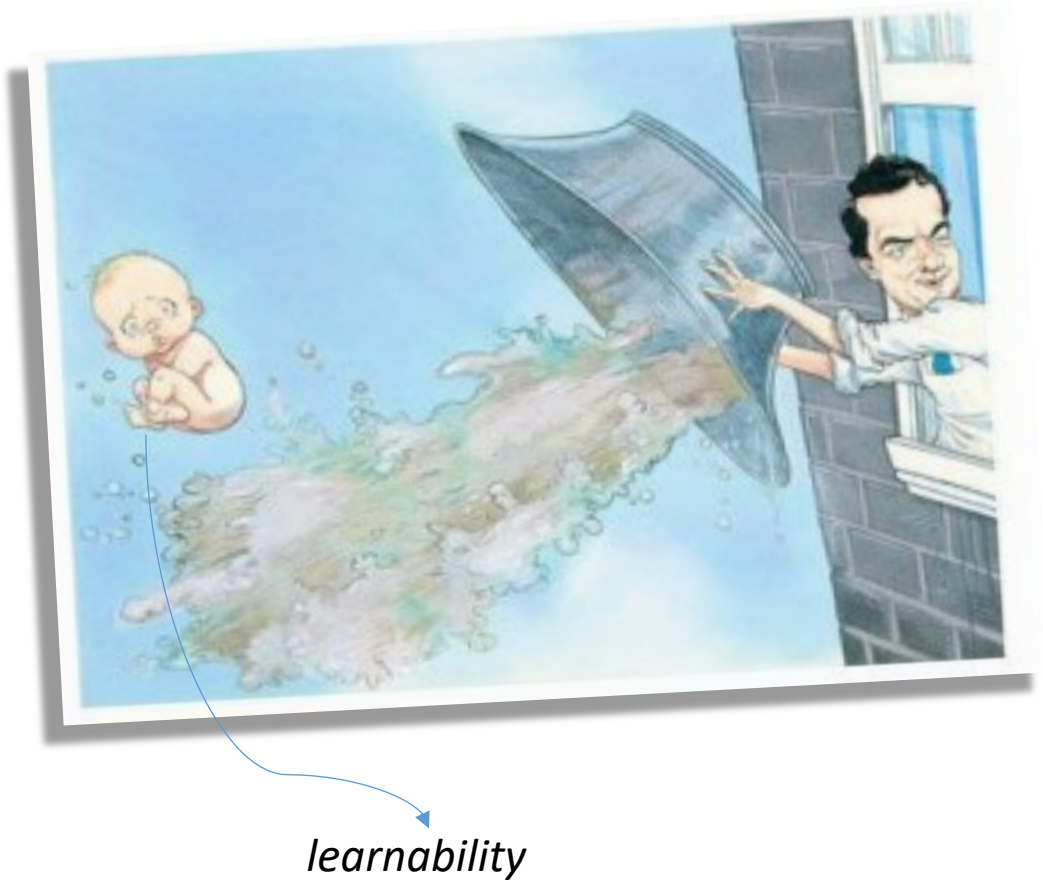
- For t iterations the covariance expressed in terms of matrices with t columns.
- $O(tm)$ for storage and computation for variance (can reduce to $O(t)$ if dimension m forms a grid).

Challenges #2: Learnability

In the presence of finite data the analytic solution might not be the best!

Neural networks, instead, perform ***hierarchical concept learning****.*

Did we throw the baby out with the bathwater? (D. MacKay)



- Maybe having finite parameters ***is*** the way to approximate GPs in a manageable way??

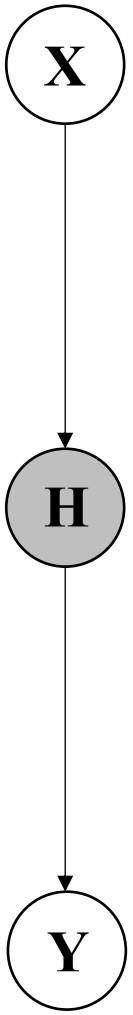
Deep Gaussian process

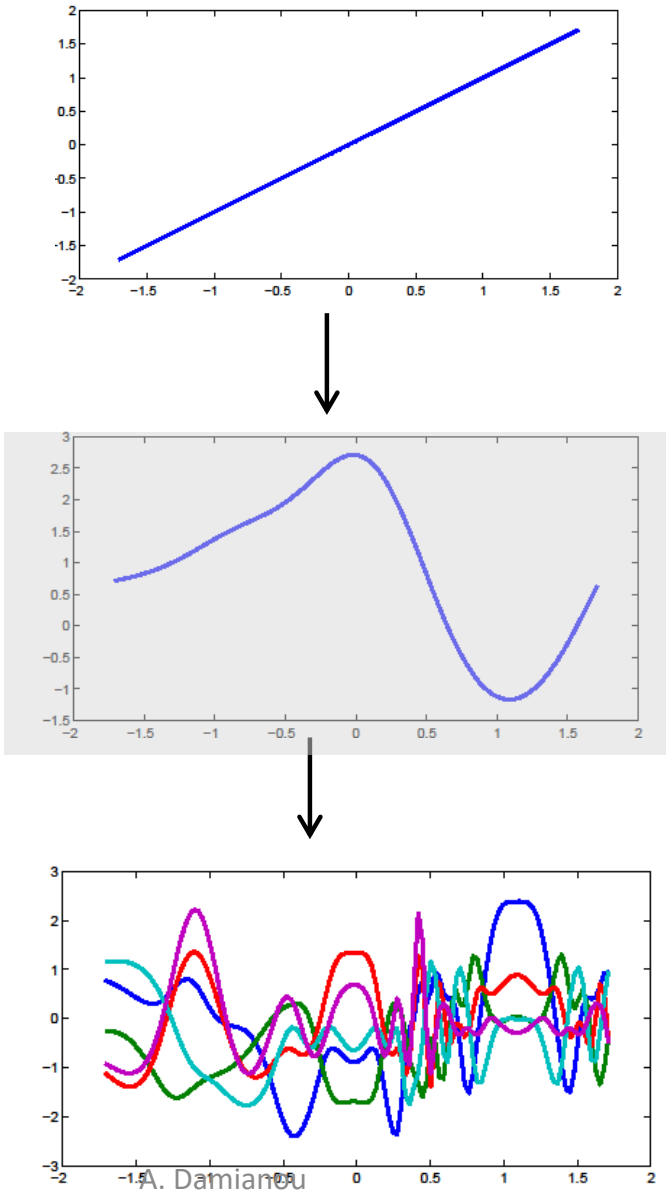
$$g(x) = f(f(f \cdots (x))) \qquad \text{where} \qquad f \sim \mathcal{GP}$$

[Damianou, Lawrence. "Deep Gaussian processes", 2013]
[Damianou, "Deep Gaussian processes and variational propagation of uncertainty", PhD Thesis]
[Kumar, Singh, Srijith, Damianou, "Deep Gaussian processes with convolutional kernels", 2018]

Deep Gaussian process

[Damianou,, PhD Thesis]





Input

Unobserved

Output

GP with NN inductive biases

[Maddox, Tang, Moreno, Wilson, Damianou (under submission)]

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
  No GP training required.
- Predictions: $\mathbf{K}_{*n} = k(x^*, \mathbf{x}) = J(x^*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$ etc.

GP with NN inductive biases

[Maddox, Tang, Moreno, Wilson, Damianou (under submission)]

- Fit DNN's parameters $\mathbf{w}$ on data $\mathbf{x}$.
- Consider a GP with kernel: $k(x, x') = J(x; \mathbf{w})^\top J(x'; \mathbf{w})$.
  No GP training required.
- Predictions: $\mathbf{K}_{*n} = k(x^*, \mathbf{x}) = J(x^*; \mathbf{w})^T J(\mathbf{x}; \mathbf{w})$ etc.
- Take *any** trained (non-Bayesian) neural network and obtain economical closed form uncertainties without further training!

Can we implement GPs in hardware?

Not currently. However, I think it's possible (but also hard!).

1. We can turn inversions into matrix-vector-products. HW is good at that (and can parallelize).
2. Prediction with clever caching can be fast.
3. Learning is feasible by incorporating inductive biases from NNs.
4. Biggest unexplored issue is the needed precision (1. can partially help).

Can we implement GPs in hardware?

Not currently. However, I think it's possible (but also hard!).

1. We can turn inversions into matrix-vector-products. HW is good at that (and can parallelize).
2. Prediction with clever caching can be fast.
3. Learning is feasible by incorporating inductive biases from NNs.
4. Biggest unexplored issue is the needed precision (1. can partially help).

Brainstorming:

- Energy efficient using mixed-precision informed by the models' self-doubt (entropy). Implicit or explicit mixture of experts. $\sigma_*^{-2} = \sum_m \beta_k \sigma_{k,*}^{-2}$
- Deep GP architecture search to compress a bigger GP.

Conclusion

- Uncertainty modeling turns optimization into integration.
- Challenges: expensive computations; required precision.
- Computations more economical with recent advances and GPUs.
- Deep GP and GP-NN hybrids allow for uncertainty modelling with good learnability.
- I shared some thoughts on uncertainty modelling in hardware.

Thanks!

Questions?

Relevant tutorials & publications: andreasdamianou.com

Appendix

GPs *for* hardware

"[round-off errors] are strictly very complicated but uniquely defined number theoretical functions [of the inputs], yet our ignorance of their true nature is such that we best treat them as random variables."

(von Neumann and Goldstine 1947, p. 1027).

We can actually use Gaussian processes to model uncertainty coming from the computation of hardware.
(Verification & Testing?)

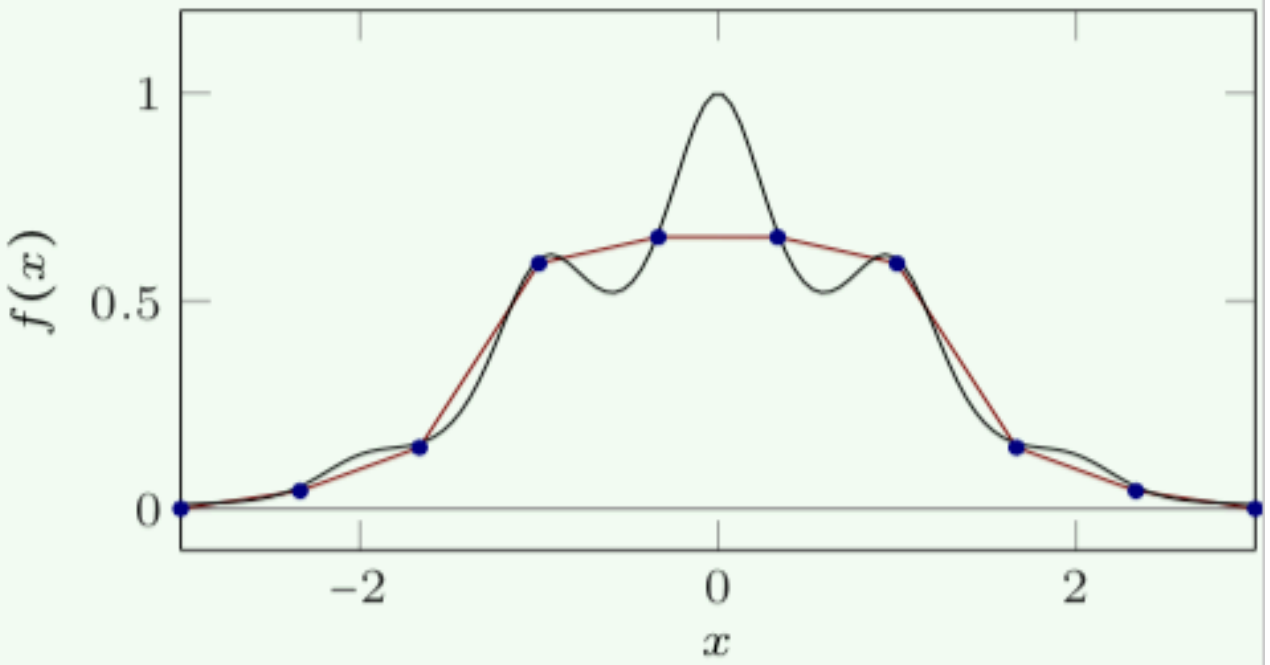
The answer to a numeric problem can only be approximated,

e.g.

$$F = \int_{-3}^{3} f(x)\mathrm{d}x$$

for

$$f(x) = \exp\left(-\left(\sin(3x)\right)^2 - x^2\right).$$



Slide from Michael Osborne

The trapezoidal rule is the posterior mean estimate for the integral

$$F = \int_a^b f(x)\,\mathrm{d}x$$

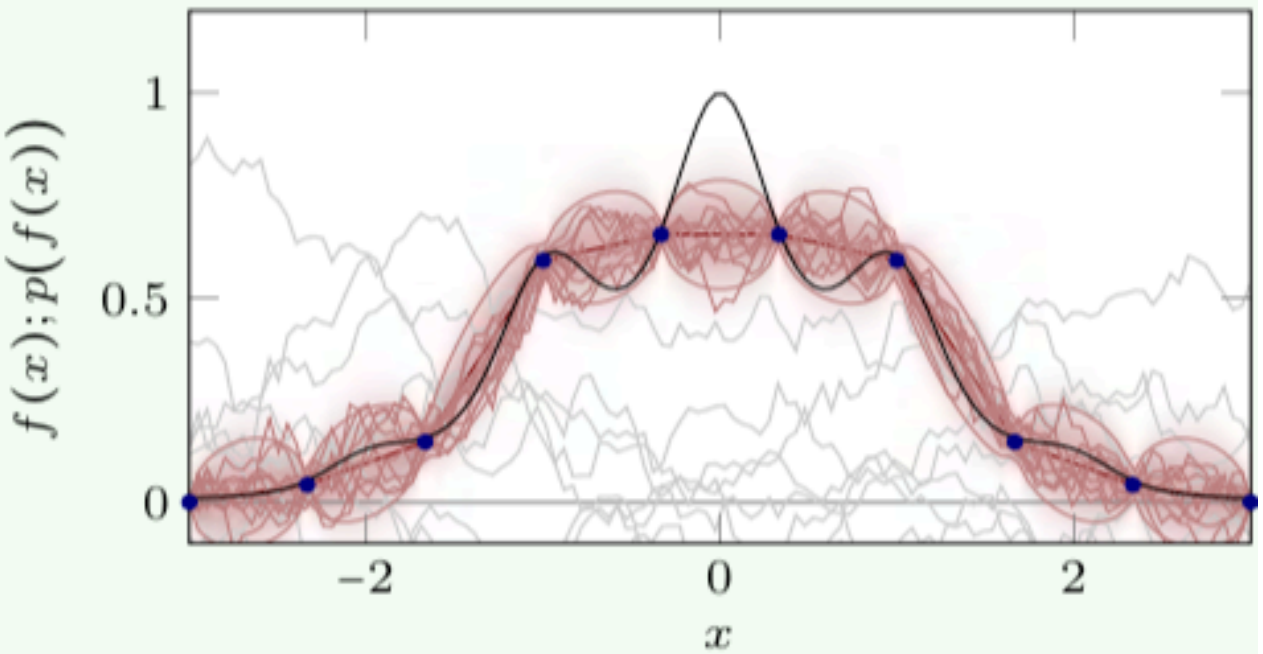
under any centered Wiener process prior

$$p(f) = \mathcal{GP}(f; 0, k)$$

with

$$k(x, x') = \theta^2 \left(\min(x, x') - \chi\right)$$

for arbitrary $\theta \in \Re_+$ and $\chi < a \in \Re$.



Slide from Michael Osborne

Computational complexities

- exact-GP : O(n^2)
- KISS-GP+Lanczos : O(t).
- exact-GP+Lanczos : O(tn) but k is very small
- sparseGP : O(m^2) but m is typically large
- sparseGP+Lanczos : O(km)

Storage costs

- exact-GP : O(n^2)
- KISS-GP+Lanczos : O(tm)
- exact-GP+Lanczos : O(tn) [but parallelizable]
- sparseGP : O(m^2)
- sparseGP+Lanczos : O(tm)

Complexities

- In training, $\mathbf{K}_{XX}$ is computed by $k(\mathbf{X}, \mathbf{X}; \theta)$. During optimization we change θ and thus have to compute the inverse in each optimization step. Cholesky is $O(n^3)$ and with ind. pts it can be done $O(nm^2)$. With KISS-GP it can be $O(n + m \log m)$ but requires grid.
- In predictions θ and hence $\mathbf{K}_{XX}$ is fixed. If we precompute whatever doesn't include $*$, then we have a standard vector-vector mult. for the mean, i.e. (O(n)). No linear solves required, so it can be done in a single GPU. For variance, if we precompute again, the remainder still costs $O(n^2)$ or $O(m^2)$ for Sparse GP or $O(k(n + mlogm))$ for KISS-GP.
- WIth KISS-GP + Lanczos (KISS-GP LOVE), instead of $\mathbf{k}_{*X}const$ we have $\mathbf{w}_{\mathbf{x}_*}$ which has only 4 nonzero elements, hence mean pred. is O(1). Pred. variance can then be O(k) with k very small, so they consider it O(1).
- With just Lanczos (and no KISS-GP), the variance goes back to O(nk) or O(mk) if used with sparse GPs. In the exact gp paper they say O(n), possibly because k is typically small. The mean remains O(n), I think, but can be parallelized.

Fast predictive variances

$$\tilde{\mathbf{k}}_{X\mathbf{x}_i^*} = W_X^\top K_{UU}\mathbf{w}_{\mathbf{x}_i^*}, \qquad \tilde{K}_{XX} = W_X^\top K_{UU} W_X$$

Approximate covariance using points in grid with weight W

$$k_{f|\mathcal{D}}(\mathbf{x}_i^*, \mathbf{x}_j^*) \approx k_{\mathbf{x}_i^*\mathbf{x}_j^*} - \tilde{\mathbf{k}}_{X\mathbf{x}_i^*}^\top (\tilde{K}_{XX} + \sigma^2 I)^{-1} \tilde{\mathbf{k}}_{X\mathbf{x}_j^*}. \quad (7)$$

By fully expanding the second term in (7), we obtain

$$\mathbf{w}_{\mathbf{x}_i^*}^\top \underbrace{K_{UU} W_X (\tilde{K}_{XX} + \sigma^2 I)^{-1} W_X^\top K_{UU}}_{C} \mathbf{w}_{\mathbf{x}_j^*} \quad (8)$$

Replace in GP predictive equation. Precomputing C with Lanczos.

$$C = K_{UU} W_X \underbrace{(\tilde{K}_{XX} + \sigma^2 I)^{-1}}_{\text{Apply Lanczos}} W_X^\top K_{UU}$$

$$\approx K_{UU} W_X (Q_k T_k^{-1} Q_k^\top) W_X^\top K_{UU}$$

$$= \underbrace{K_{UU} W_X Q_k}_{R^\top} \underbrace{T_k^{-1} Q_k^\top W_X^\top K_{UU}}_{R'}$$

C precomputed with t iterations, giving us structure R^T R

To compute R and R', we perform k iterations of Lanczos to achieve $(\tilde{K}_{XX} + \sigma^2 I) \approx Q_k T_k Q_k^\top$ using the av-

R is *m x t*