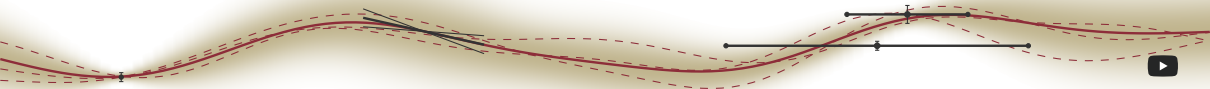
Probabilistic Machine Learning
Lecture 16
Deep Learning

Philipp Hennig

26 June 2023

Eberhard Karls
Universität Tübingen

Faculty of Science
Department of Computer Science
Chair for the Methods of Machine Learning



Recap

the course so far

- **Learning** is **Inference** – re-weighing a space of hypotheses through likelihood functions (Bayes' theorem). Probability theory is just the mathematical formulation of keeping correct track of volume/measure
- **Exponential Families** are the parametric probability distributions that allow tractable inference
- **Gaussian distributions** are the exponential family in which inference reduces to linear algebra
- They can be used to learn general linear *functions* $f : \mathbb{X} \to \mathbb{R}, f(x) = \phi(x)^\intercal w$ – **linear regression**
- Since this involses only inner products (kernels, covariance functions) $k(\bullet, \circ) = \phi(\bullet)^\intercal \Sigma \phi(\circ)$, it can be abstracted to a functional form that does not require explicit features, a *nonparametric* model called **Gaussian Process** regression
- For **Classification** (functions $f : \mathbb{X} \to [0, 1]^C$), we can adapt this framework slightly by considering a sigmoid likelihood (**logistic regression**). This requires *approximate* inference, which can be realised by **Laplace approximations**

Deep Learning

A vague definition

For our purposes, a **deep neural network** is a function

$$f(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{X} \times \mathbb{R}^D \to \mathbb{R}^F$$

parametrized by *parameters* $\boldsymbol{\theta} \in \mathbb{R}^D$ and mapping inputs $\boldsymbol{x} \in \mathbb{X}$ to outputs $f(\boldsymbol{x}, \boldsymbol{\theta}) \in \mathbb{R}^F$. Such functions are often realised in a hierarchical fashion

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{b}_L + \boldsymbol{w}_L \sigma\left(\boldsymbol{b}_{L-1} + \boldsymbol{w}_{L-1} \sigma\left(\cdots \sigma(\boldsymbol{b}_0 + \boldsymbol{w}_0 \boldsymbol{x})\right)\right)$$

parametrised by *weights* and *biases* $\boldsymbol{\theta} = [\boldsymbol{b}_i, \boldsymbol{w}_i]_{i=1,\ldots,L}$ and *nonlinearities* σ (e.g. ReLU, tanh, …), but we will ignore these details here (→ Deep Learning course)

Code

1. defining a model

How are Deep Architectures Trained?

Empirical Risk Minimization

Deep learning is usually realised as **empirical risk minimization** (ERM), to find parameters $\boldsymbol{\theta}_*$ on a training set $\mathcal{D} = [(x_i, y_i)]_{i=1,\dots,N}$

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{loss}} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right).$$

Typical choices of loss ℓ are **cross-entropy** (aka. **log loss**) for classification, **mean squared error** for regression (the sum then amounts to an independence assumption, $p(\boldsymbol{y} \mid f, \boldsymbol{x}) = \prod_{i=1}^{N} p(y_i \mid f(\boldsymbol{x}_i), \boldsymbol{x}_i)$).

$$\ell_{\text{logistic}}(y_i, \hat{y}_i) = -y_i \log \hat{y}_i - (1 - y_i) \log(1 - \hat{y}_i) = -\log \hat{p}(y_i) = -\log p(y_i \mid \hat{p}(y_i)) = \log(\sigma f(x_i))$$

$$\ell_{\text{CE}}(y_i, \hat{y}_i) = -\sum_{c=1}^{C} \mathbb{I}_{y_i=c} \left(\log(\hat{y}_{ic}) - \log \sum_{c'} \hat{y}_{ic'} \right) = -\log \hat{p}(y_i) = -\log p(y_i \mid \hat{p}(y_i)) = \log \operatorname{softmax} f(x_i)$$

$$\ell_{\text{MSE}}(y_i, \hat{y}_i) = \frac{1}{2} \|y_i - \hat{y}_i\|^2 = -\log \mathcal{N}(y_i; \hat{y}_i, I) = -\log p(y_i \mid \hat{y}_i)$$

How are Deep Architectures Trained?

Empirical Risk Minimization

Deep learning is usually realised as **empirical risk minimization** (ERM), to find parameters $\boldsymbol{\theta}_*$ on a training set $\mathcal{D} = [(x_i, y_i)]_{i=1,\dots,N}$

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{loss}} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right).$$

a typical choice of regularizer r is **weight decay** (L2-regularization)

$$\begin{aligned} r_{\text{L2}}(\boldsymbol{\theta}) &= \frac{\lambda}{2} \sum_{i=1}^{D} \boldsymbol{\theta}_i^2 = \frac{\lambda}{2} \|\boldsymbol{\theta}\|_2^2 \\ &= -\log \mathcal{N}(\boldsymbol{\theta}; 0, \lambda^{-1} I) = -\log p(\boldsymbol{\theta} \mid \lambda) \end{aligned}$$

How are Deep Architectures Trained?

Empirical Risk Minimization

Deep learning is usually realised as **empirical risk minimization** (ERM), to find parameters $\boldsymbol{\theta}_*$ on a training set $\mathcal{D} = [(x_i, y_i)]_{i=1,\dots,N}$

$$\boldsymbol{\theta}_* = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \underbrace{\mathcal{L}(\boldsymbol{\theta})}_{\text{loss}} = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \left(\frac{1}{N} \sum_{i=1}^{N} \underbrace{\ell(y_i; \overbrace{f(x_i, \boldsymbol{\theta})}^{\text{deep net}})}_{\text{empirical risk}} + \underbrace{r(\boldsymbol{\theta})}_{\text{regularizer}} \right).$$

We thus see that the ERM objective is equivalent to a **maximum a posteriori** (MAP) estimate

$$\begin{aligned} \boldsymbol{\theta}_* &= \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \mathcal{L}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}\in\mathbb{R}^D} -\log p(\mathcal{D} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) \\ &= \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^D} \underbrace{\log p(\mathcal{D} \mid \boldsymbol{\theta})}_{\text{likelihood}} + \underbrace{\log p(\boldsymbol{\theta})}_{\text{prior}} = \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^D} p(\boldsymbol{\theta} \mid \mathcal{D}) \end{aligned}$$

Code

1. defining a model
2. defining a loss
3. running an optimizer

Context: Parametric Regression

connecting previous lectures to deep learning

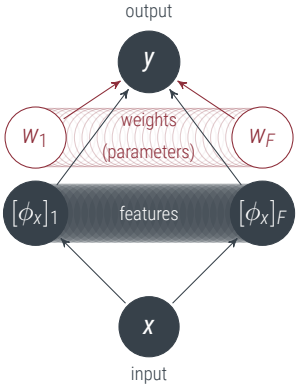
For Gaussian / parametric / least-squares regression, we posited functions of the *linear* form

$$f(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{X} \times \mathbb{R}^D \to \mathbb{R}^F, \qquad f(\boldsymbol{x}, \boldsymbol{\theta}) = \phi(\boldsymbol{x})^\intercal \boldsymbol{\theta}$$

Here $\phi : \mathbb{X} \to \mathbb{R}^D$ can have pretty much *any* form (including discontinuities, point-masses, etc., assuming we take care to handle things right in the code). And $\mathbb{X}$ can be pretty much *any* set. $\mathbb{R}^M$ is most common, but could also be

- strings – Natural Language Processing
- graphs – molecules, genes, proteins
- functions – operators, simulation
- Gödel numbers, etc.

because ϕ "masks" $\mathbb{X}$.



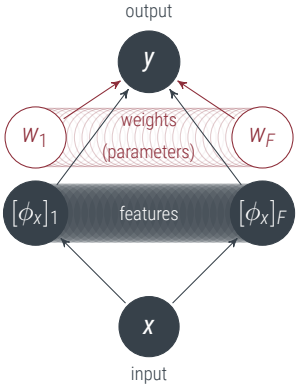
Context: Parametric Regression

connecting previous lectures to deep learning

$$f(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{X} \times \mathbb{R}^D \to \mathbb{R}^F, \qquad f(\boldsymbol{x}, \boldsymbol{\theta}) = \phi(\boldsymbol{x})^\intercal \boldsymbol{\theta}$$

Probabilistic inference on $\boldsymbol{\theta}$ from Gaussian generative model

$$\begin{aligned} p(\boldsymbol{\theta}) &= \mathcal{N}(\boldsymbol{\theta}; \mu, \Sigma) \qquad p(\boldsymbol{y} \mid \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y}; \phi_X^\intercal \boldsymbol{\theta}, \sigma^2 I) \\ p(\boldsymbol{\theta} \mid \boldsymbol{y}) &= \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\boldsymbol{y})} \\ &= \mathcal{N}(\boldsymbol{\theta}; \mu_{\text{post}}, \Sigma_{\text{post}}) \quad \text{with} \\ \mu_{\text{post}} &= (\Sigma^{-1} + \sigma^{-2} \phi_X \phi_X^\intercal)^{-1} (\Sigma^{-1} \mu + \sigma^{-2} \phi_X \boldsymbol{y}) \\ \Sigma_{\text{post}} &= (\Sigma^{-1} + \sigma^{-2} \phi_X \phi_X^\intercal)^{-1} \end{aligned}$$



Observation: μ_{post} and $-\Sigma_{\text{post}}^{-1}$ are *mode* and *Hessian* of $\log p(\boldsymbol{\theta} \mid \boldsymbol{y})$.

Context: Parametric Regression

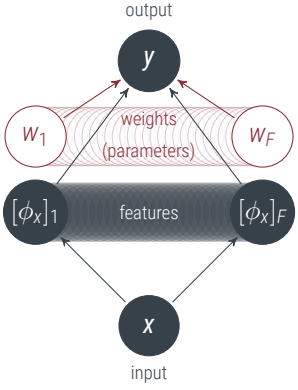
connecting previous lectures to deep learning

$$f(\boldsymbol{x}, \boldsymbol{\theta}) : \mathbb{X} \times \mathbb{R}^D \to \mathbb{R}^F, \qquad f(\boldsymbol{x}, \boldsymbol{\theta}) = \phi(\boldsymbol{x})^\intercal \boldsymbol{\theta}$$

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}; \mu, \Sigma) \qquad p(\boldsymbol{y} \mid f_X) = \mathcal{N}(\boldsymbol{y}; f_X, \sigma^2 I)$$

We could have described Gaussian inference as $L2$-regularized empirical risk minimization

$$\begin{aligned}
\mu_{\text{post}} &= \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} -\log p(\boldsymbol{\theta} \mid \boldsymbol{y}) = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} -\log p(\boldsymbol{y} \mid \boldsymbol{\theta}) - \log p(\boldsymbol{\theta}) \\
&= \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{2\sigma^2} \|\boldsymbol{y} - \phi_X^\intercal \boldsymbol{\theta}\|^2 + \frac{1}{2}(\boldsymbol{\theta} - \mu)^\intercal \Sigma^{-1} (\boldsymbol{\theta} - \mu) \\
&= \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{2} \sum_{i=1}^{N} (y_i - \phi_{x_i}^\intercal \boldsymbol{\theta})^2 + \frac{\sigma}{2} \|\boldsymbol{\theta} - \mu\|_\Sigma^2 \\
\Sigma_{\text{post}}^{-1} &= -\nabla \nabla^\intercal \log p(\boldsymbol{\theta} \mid \boldsymbol{y})
\end{aligned}$$



$$p(f) = \mathcal{GP}(f; m, k) \qquad p(\boldsymbol{y} \mid f_X) = \mathcal{N}(\boldsymbol{y}; f_X, \sigma^2 I) \qquad p(f \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid f_X) p(f)}{p(\boldsymbol{y})}$$

$$m_{\text{post}}(\bullet) = m_\bullet + K_{\bullet X}(k_{XX} + \sigma^2 I)^{-1}(f_X - m_X)$$

$$k_{\text{post}}(\bullet, \circ) = k_{\bullet\circ} - K_{\bullet X}(k_{XX} + \sigma^2 I)^{-1} K_{X\circ}$$

A Gaussian *process* model is *nonparametric* ("there are no weights"). But…

- $m_{\text{post}}(\bullet)$ is the minimizer of the *ridge loss* in the RKHS $\mathcal{H}_k$

$$m_{\text{post}}(\bullet) = \arg\min_{f \in \mathcal{H}_k} \left(\frac{1}{2} \sum_{i=1}^{N} (y_i - f(x_i))^2 + \frac{\sigma}{2} \|f\|^2_{\mathcal{H}_k} \right)$$

- $k_{\text{post}}(\bullet, \bullet)$ is a worst-case error estimate for RKHS functions of bounded norm (Lecture 11)
- Every RKHS function can be expanded in the (countably many) eigenfunctions ϕ of the kernel:

$$\mathcal{H}_k = \left\{ f(x) := \sum_{i \in I} \alpha_i \lambda_i^{1/2} \phi_i(x) \text{ such that } \|f\|^2_{\mathcal{H}_k} := \sum_{i \in I} \alpha_i^2 < \infty \right\} \quad \text{with} \quad \langle f, g \rangle_{\mathcal{H}_k} := \sum_{i \in I} \alpha_i \beta_i$$

Context: Logistic Regression / GP Classification

connecting previous lectures to deep learning

To model *classification* problems, we change the **likelihood**

$$p(y \mid f(x)) = \sigma(yf(x)) \quad \text{with} \quad \sigma(a) = \frac{1}{1+\exp(-a)}$$
$$-\log p(y \mid f(x) = \log(1+\exp(-yf(x))) := \log(1+\exp(-yf(x)))$$

or for multi-class:

$$p(y \mid f(x)) = \text{softmax}(f(x))_y \quad \text{with} \quad \text{softmax}(a)_i = \frac{\exp(a_i)}{\sum_{j=1}^{C}\exp(a_j)}$$

$$-\log p(y \mid f(x)) = -f(x)_y + \log \sum_{j=1}^{C} \exp(f(x)_j) \qquad \text{cross-entropy loss}$$

This requires *multi-output* GPs …

Multi-Output GPs

aside

- It is possible to define a GP over multiple functions $[f_c(x)]_{c=1,\dots,C}$ through a joint covariance function
$$k(f_c(a), f_d(b)) = k((a,c),(b,d))$$
- It is common to factorise covariance between inputs and outputs: $k((a,c),(b,d)) = g(a,b) \cdot h(c,d)$, i.e. the outputs are independent. This leads to numerical simplifications, because then the Gram matrix has *Kronecker* structure
$$[K_{XX}]_{a,c,b,d} = g(a,b) \cdot h(c,d) = [G \otimes H]_{a,c,b,d} \quad \text{and} \quad K_{XX}^{-1} = G^{-1} \otimes H^{-1}$$
- A special case is $h(c,d) = \delta_{cd}$, independent outputs. For regression with $p(\boldsymbol{y} \mid f(x)) = \prod_c \mathcal{N}(y_c; f_c(x), \sigma_c^2)$, this is like learning completely detached GP models. For classification, the likelihood *induces covariance*, which should be tracked (it's of rank 1, though).
- None of this is particularly difficult, but it does not fit well in the code structure we developed so far, so we will leave it out. (The price we pay is that we can't talk about multi-class classification in the exercises).

Are infinitely many weights enough?

[Micchelli, Xu, Zhang, JMLR 7 (2006) 2651–2667]

- From the above, we can think of a GP as "an infinitely wide single-layer neural network"
- (Note, however, that GP samples are not in the RKHS…)
- For some kernels, the RKHS "lies dense" in the space of all continuous functions (such kernels are known as "universal"). An example is the square-exponential / Gaussian / RBF kernel

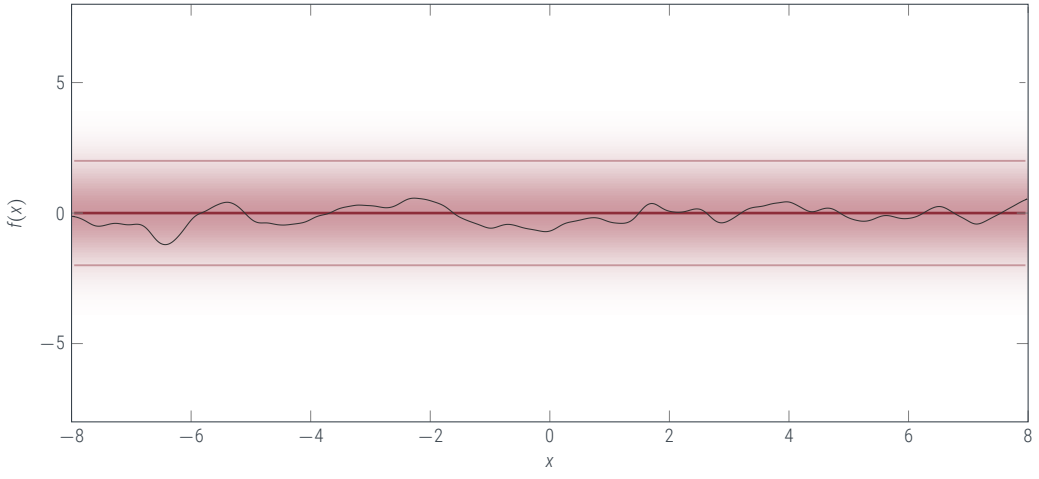
$$k(a,b) = \exp(-1/2(a-b)^2)$$

(in fact, there are many universal kernels. E.g. all stationary kernels with power spectrum of full support.)

- When using such kernels for GP / kernel-ridge regression, for any continuous functions f, for any $\epsilon > 0$ there is an RKHS element $\hat{f} \in \mathcal{H}_k$ such that $\|f - \hat{f}\| < \epsilon$ (where $\|\cdot\|$ is the maximum norm on a compact subset of $\mathbb{X}$).
- that is: Given enough data, the GP posterior mean can approximate *any function* arbitrarily well!

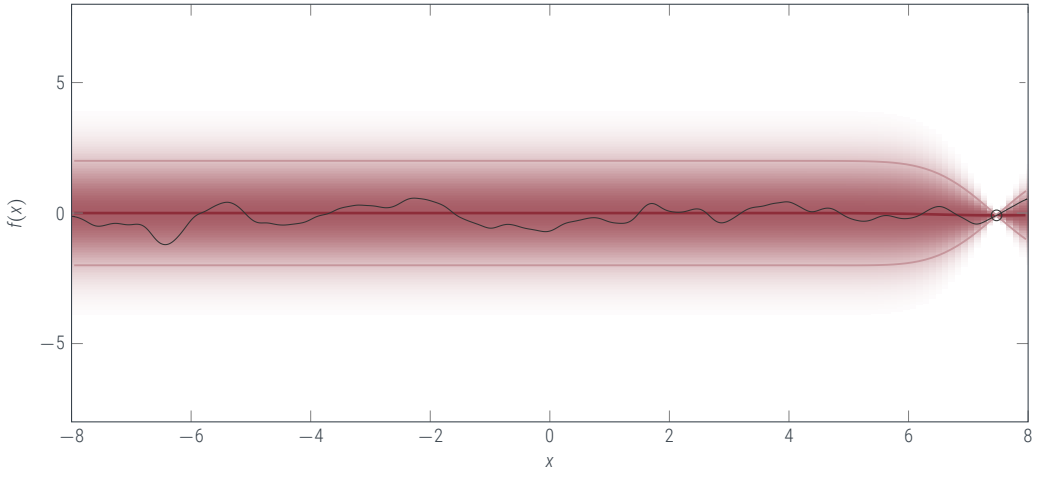
The bad news

if f is not in the RKHS – prior



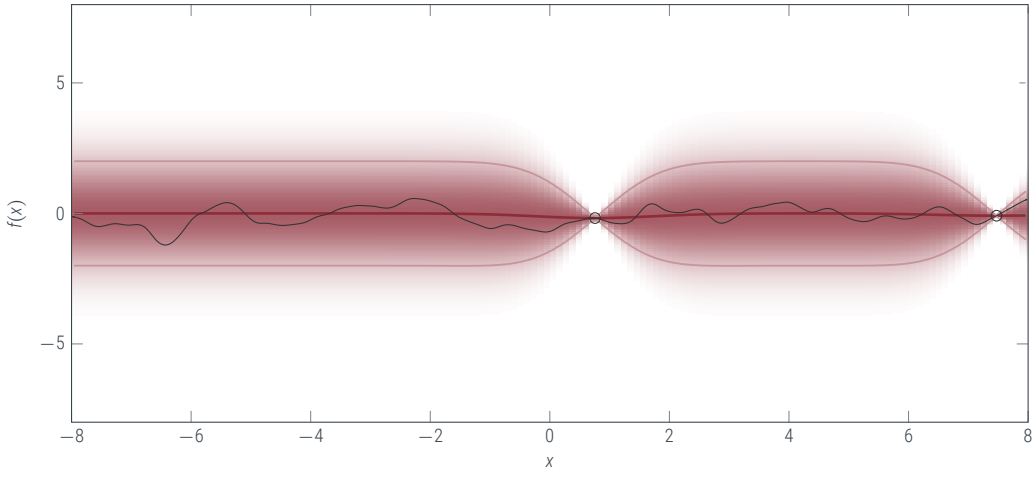
The bad news

if f is not in the RKHS – 1 evaluation



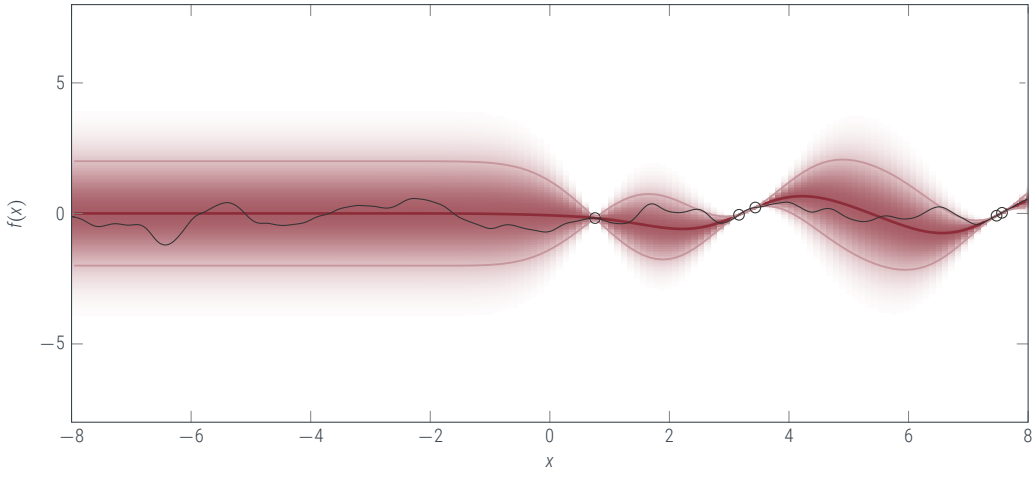
The bad news

if f is not in the RKHS – 2 evaluations



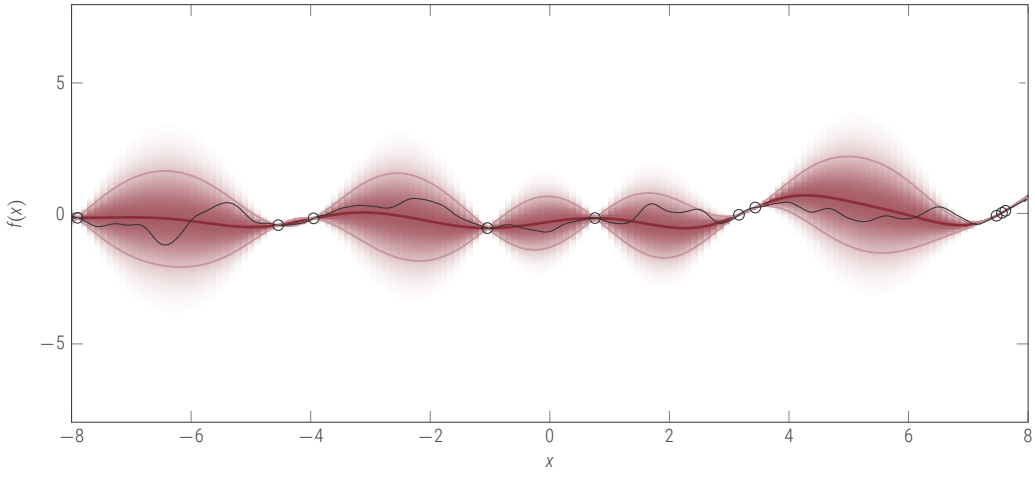
The bad news

if f is not in the RKHS – 5 evaluations



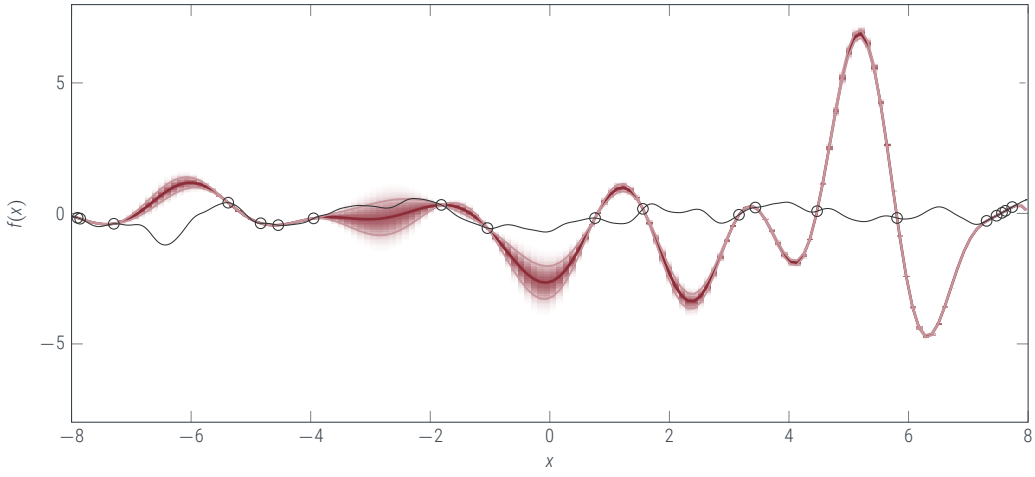
The bad news

if f is not in the RKHS – 10 evaluations



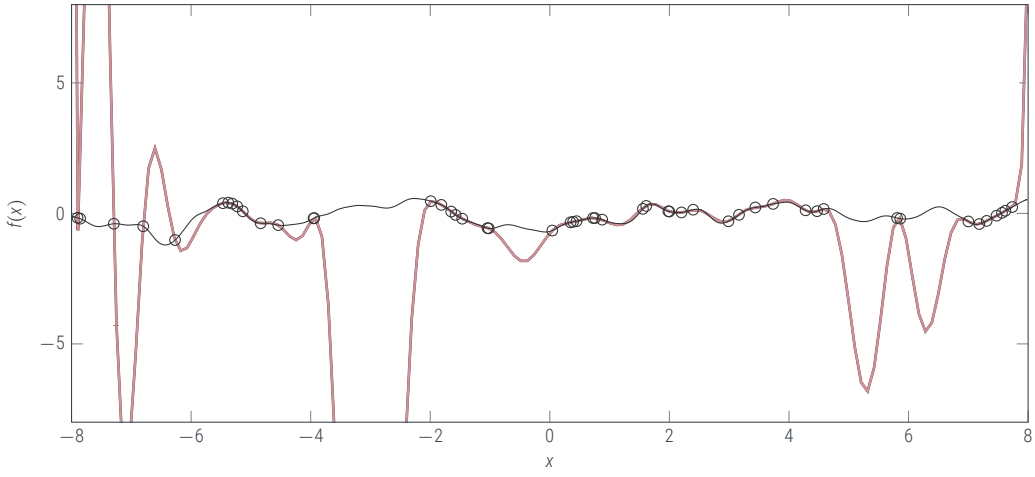
The bad news

if f is not in the RKHS – 20 evaluations



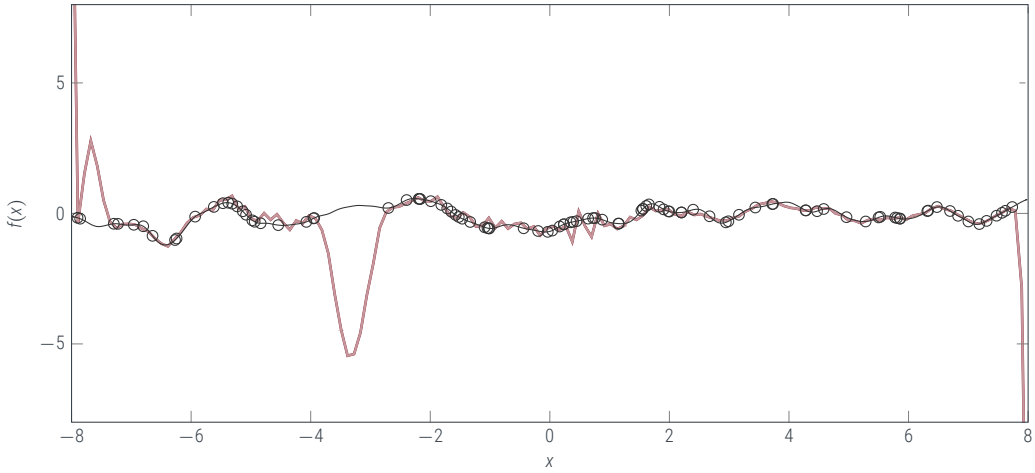
The bad news

if f is not in the RKHS – 50 evaluations



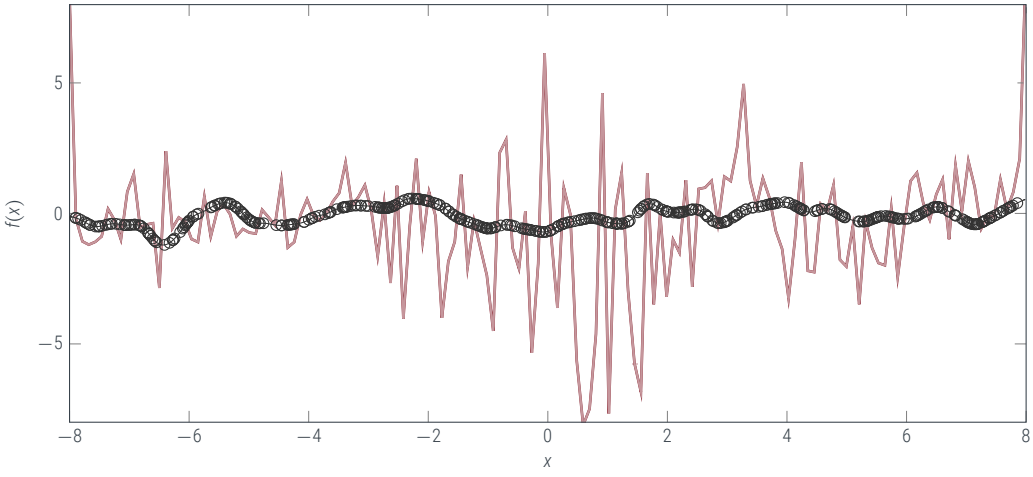
The bad news

if f is not in the RKHS – 100 evaluations



The bad news

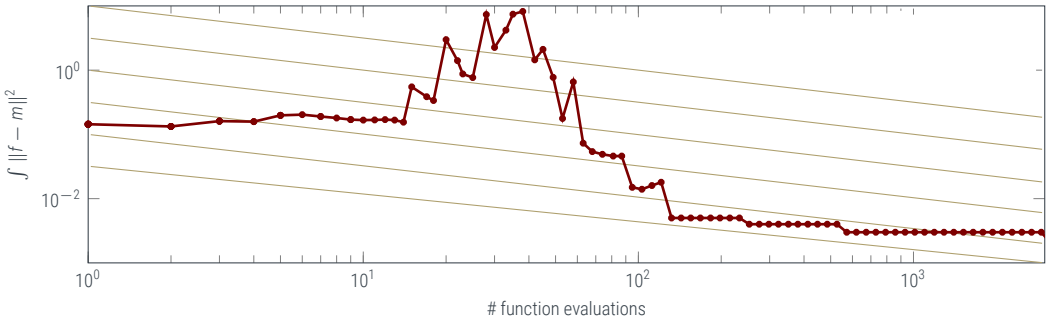
if f is not in the RKHS – 500 evaluations



Convergence Rates are Important

non-obvious aspects of f can ruin convergence

v.d.Vaart & v.Zanten. *Information Rates of Nonparametric GP models*. JMLR 12 (2011)



If f is "not well covered" by the RKHS, the number of datapoints required to achieve ϵ error can be exponential in ϵ. Outside of the observation range, there are no guarantees at all.

An Analogy

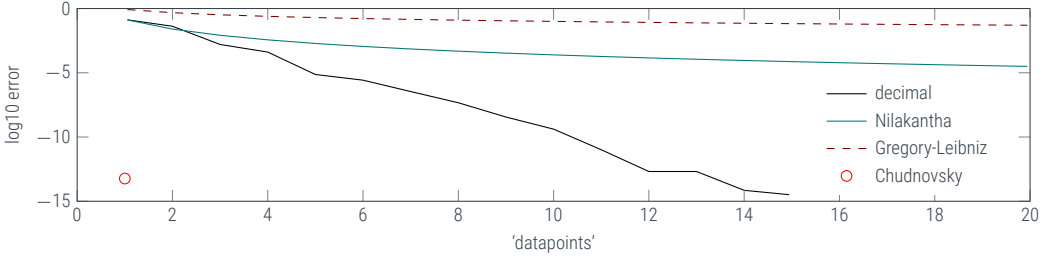
representing π in $\mathbb{Q}$

- $\mathbb{Q}$ is dense in $\mathbb{R}$

$$\pi = 3 \cdot \frac{1}{1} + 1 \cdot \frac{1}{10} + 4 \cdot \frac{1}{100} + 1 \cdot \frac{1}{1000} + \dots \qquad \text{decimal}$$

$$= 4 \cdot \frac{1}{1} - 4 \cdot \frac{1}{3} + 4 \cdot \frac{1}{5} - 4 \cdot \frac{1}{7} + \dots \qquad \text{Gregory-Leibniz}$$

$$= 3 \cdot \frac{1}{1} + 4 \cdot \frac{1}{2 \cdot 3 \cdot 4} - 4 \cdot \frac{1}{4 \cdot 5 \cdot 6} + 4 \cdot \frac{1}{6 \cdot 7 \cdot 8} \qquad \text{Nilakantha}$$



But if you're patient, you can learn anything!

The good news.

[wording from Kanagawa et al., 2018]

Theorem (v.d. Vaart & v. Zanten, 2011)

Let f_0 be an element of the Sobolev space $W_2^\beta[0,1]^d$ with $\beta > d/2$. Let k_s be a kernel on $[0,1]^d$ whose RKHS is norm-equivalent to the Sobolev space $W_2^s([0,1]^d)$ of order $s := \alpha + d/2$ with $\alpha > 0$. If $f_0 \in C^\beta([0,1]^d) \cap W_2^\beta([0,1]^d)$ and $\min(\alpha, \beta) > d/2$, then we have

$$\mathbb{E}_{\mathcal{D}_n | f_0}\left[\int \|f - f_0\|^2_{L_2(P_\mathbb{X})} d\Pi_n(f|\mathcal{D}_n)\right] = O(n^{-2\min(\alpha,\beta)/(2\alpha+d)}) \quad (n \to \infty), \tag{1}$$

where $\mathbb{E}_{X,Y|f_0}$ denotes expectation with respect to $\mathcal{D}_n = (x_i, y_i)_{i=1}^n$ with the model $x_i \sim P_\mathbb{X}$ and $p(\boldsymbol{y} \mid f_0) = \mathcal{N}(\boldsymbol{y}; f_0(X), \sigma^2 I)$, and $\Pi_n(f|\mathcal{D}_n)$ the posterior given by GP-regression with kernel k_s.

The Sobolev space $W_2^s(\mathbb{X})$ is the vector space of real-valued functions over $\mathbb{X}$ whose derivatives up to s-th order have bounded L_2 norm. $L_2(P_\mathbb{X})$ is the Hilbert space of square-integrable functions with respect to $P_\mathbb{X}$.

If f_0 is from a sufficiently smooth space, and H_k is "covering" that space well, then the entire GP posterior (including the mean!) can contract around the true function at a **linear** rate.
GPs are "infinitely flexible": They can learn infinite-dimensional functions arbitrarily well!

So what?

Deep or wide?

- Nonparametric models (kernel machines, GPs) provide the *theoretical foundation* for supervised learning
- They *can* learn fast, but just "having infinite flexibility" does not mean they learn *fast*
- There are applications in which designing the RKHS / sample space carefully matters. E.g. simulation methods (Pförtner et al. 2023). It's difficult to define a deep network with hypothesis class "all twice-differentiable functions".

The real reason why deep learning is popular are practical. But it's good to separate fact from misconception.

Deep Learning

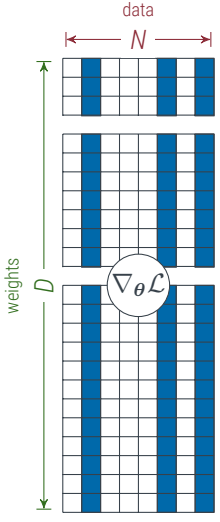
an assessment

What people like about deep learning:

- Training (with SGD, Adam, etc.) is "$\mathcal{O}(1)$", because one can compute *stochastic* gradients

$$\nabla \mathcal{L}(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \nabla \ell(y_i, f(\boldsymbol{w}, \boldsymbol{x}_i)) + \nabla r(\boldsymbol{w})$$
$$\approx \frac{1}{B} \sum_{j=1}^{B} \nabla \ell(y_{i(j)}, f(\boldsymbol{w}, \boldsymbol{x}_{i(j)})) + \nabla r(\boldsymbol{w}) =: g(\boldsymbol{w})$$

- The parametric loss has an *array* structure, allowing efficient sharding and parallelization
- People like metaphors. *Neural* network, skip connections, attention, pooling, compression, etc.
- Once the model is trained, you can throw away the data, and hide the model behind an API



Deep Learning

an assessment

What people **don't** like about deep learning:

- Training a deep net is fiddly, and requires *many* choices
  - Initialization strategy for the weights
  - Learning rate (learning-rate *schedule*!), other optimizer parameters
  - Regularization (dropout, batchnorm, weight decay, etc.)
  - Deciding when to stop, monitoring optimizer for stability
  - architecture overall
- Once the model is trained, it's unclear how to *update* it if new data arrives
- deep learning has certain conceptual pathologies (more in next lecture), leading to adversarial and out-of-distribution brittleness

What people **like** about GPs:

- Training is *linear algebra*. *No* parameters.
- Full probabilistic model, with interpretability (draw samples from the prior!), and uncertainty quantification
- Updating the model with new data is easy (Schur complement…)
- elegant mathematical theory (not fully covered in this lecture)

What people **don't** like about GPs:

- Training is "$\mathcal{O}(N^3)$"
- "The data is the model",
  - so releasing the model involves releasing the data
  - because all data interact directly (not via the weight-space), sharding and parallelization is not as straightforward

Summary:

- Deep Learning and GP regression/classification are closely related
- Central difference: Deep models are hierarchical, and thus *nonlinear*.
- This *may* be good for their performance, but also means training requires nonlinear optimization
- Shallow (GP) and deep models both have advantages and disadvantages.
- Next lecture: Combining GPs and deep learning

Please cite this course, as

```
@techreport{Tuebingen_ProbML23,
    title =
    {Probabilistic Machine Learning},
    author = {Hennig, Philipp},
    series = {Lecture Notes
        in Machine Learning},
    year = {2023},
    institution = {Tübingen AI Center}}
```