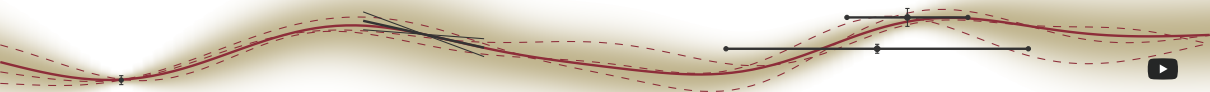
Probabilistic Inference and Learning
Lecture 08
Learning Representations

Philipp Hennig

12 May 2020

Eberhard Karls
Universität Tübingen

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



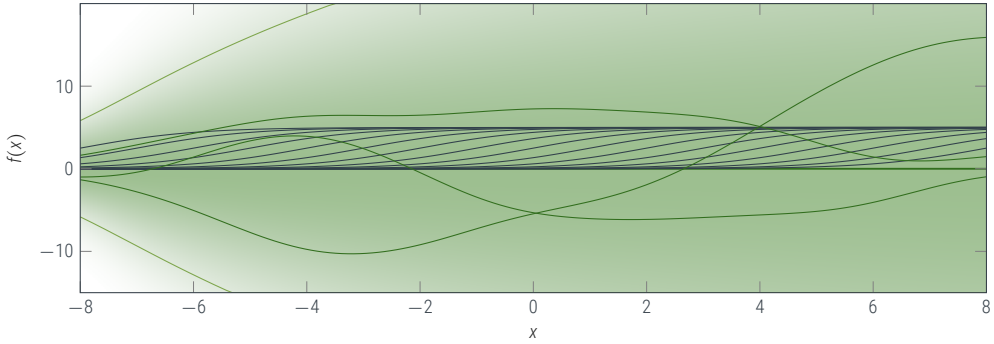
#	date	content	Ex
1	20.04.	Introduction	1
2	21.04.	Reasoning under Uncertainty	
3	27.04.	Continuous Variables	2
4	28.04.	Monte Carlo	
5	04.05.	Markov Chain Monte Carlo	3
6	05.05.	Gaussian Distributions	
7	11.05.	Parametric Regression	4
8	12.05.	**Learning Representations**	
9	18.05.	Gaussian Processes	5
10	19.05.	An Example for GP Regression	
11	25.05.	Understanding Kernels	6
12	26.05.	Gauss-Markov Models	
13	08.06.	GP Classification	7
14	09.06.	Logistic Regression	8
15	15.06.	Exponential Families	
16	16.06.	Graphical Models	9
17	22.06.	Factor Graphs	
18	23.06.	The Sum-Product Algorithm	10
19	29.06.	Example: Topic Models	
20	30.06.	Mixture Models	11
21	06.07.	EM	
22	07.07.	Variational Inference	12
23	13.07.	Example: Topic Models	
24	14.07.	Example: Inferring Topics	13
25	20.07.	Example: Kernel Topic Models	
26	21.07.	Revision	

Coming up: Ways to *learn representations*

- Can we *learn* the features?
- How do we do this in practice?
- hierarchical Bayesian inference
- Connections to deep learning

Reminder: General Linear Regression

An unbounded abundance of choices for features



$$p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi_x^\intercal \mu, \phi_x \Sigma \phi_x)$$

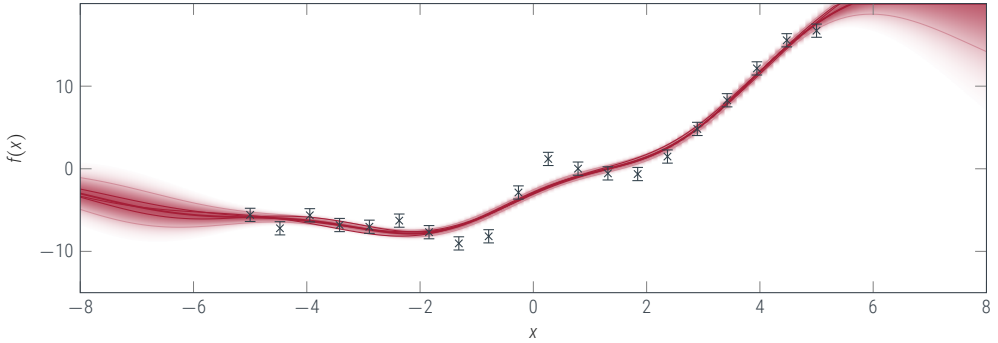
$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I) = \mathcal{N}(y; f_X, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x;\ \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),\ \phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi$$

(2)

Reminder: General Linear Regression

An unbounded abundance of choices for features



$$p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi_x^\intercal \mu, \phi_x \Sigma \phi_x)$$

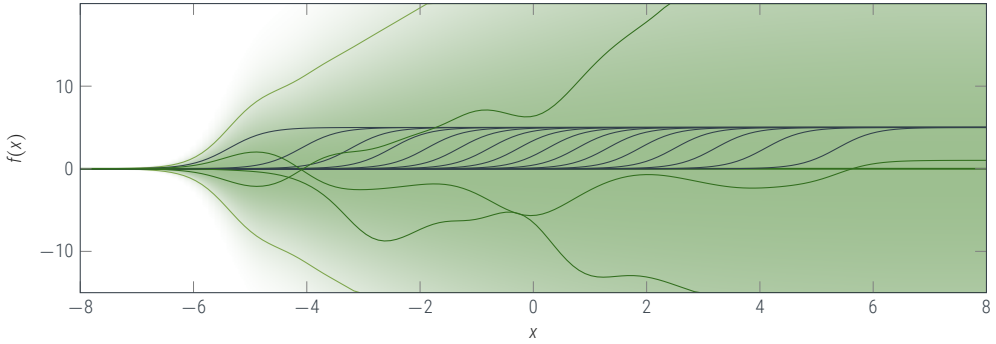
$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I) = \mathcal{N}(y; f_X, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x;\ \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),\ \phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi$$

(2)

Reminder: General Linear Regression

An unbounded abundance of choices for features



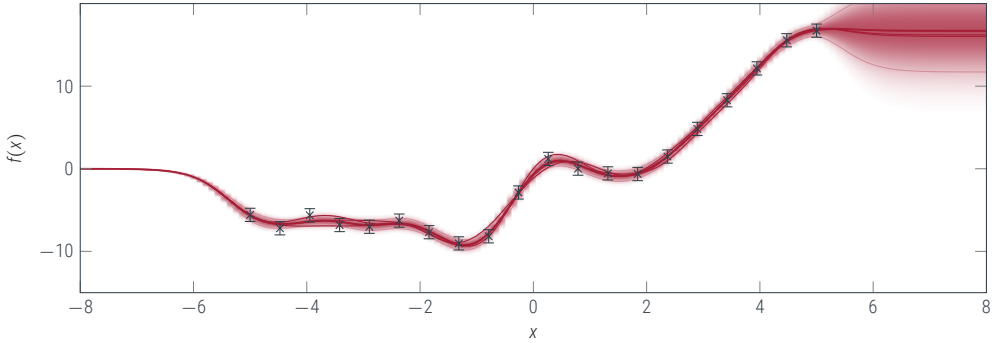
$$p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi_x^\intercal \mu, \phi_x \Sigma \phi_x)$$

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I) = \mathcal{N}(y; f_X, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x;\ \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),\ \phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi$$

Reminder: General Linear Regression

An unbounded abundance of choices for features



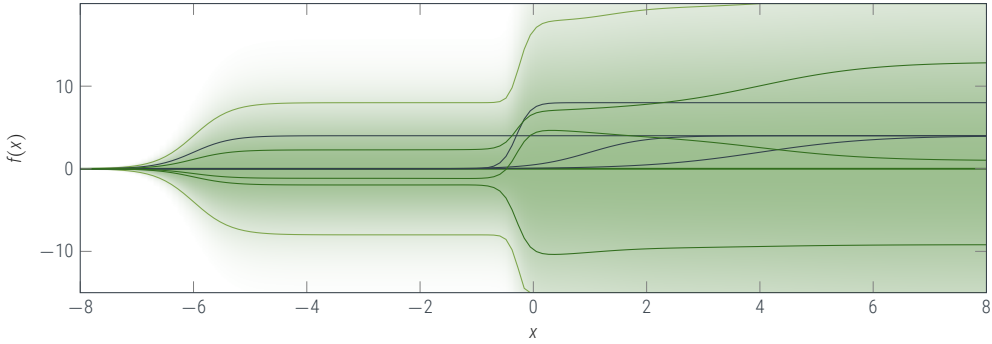
$$p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi_x^\intercal \mu, \phi_x \Sigma \phi_x)$$

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I) = \mathcal{N}(y; f_X, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x;\ \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),\ \phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi$$

Reminder: General Linear Regression

An unbounded abundance of choices for features



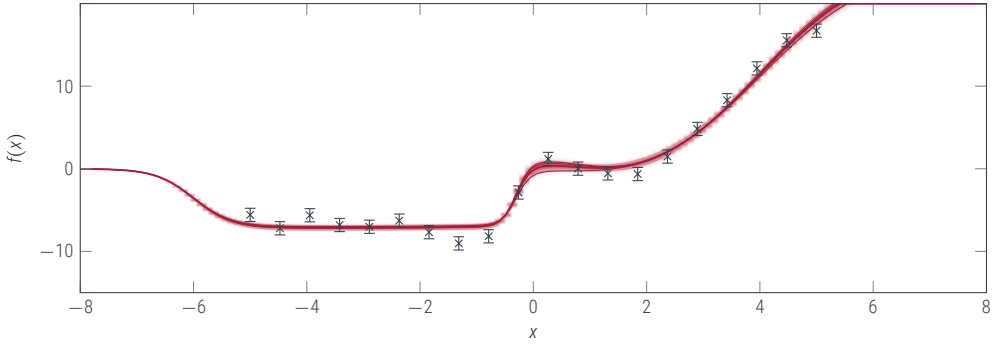
$$p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi_x^\intercal \mu, \phi_x \Sigma \phi_x)$$

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I) = \mathcal{N}(y; f_X, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x;\ \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),\ \phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi$$

Reminder: General Linear Regression

An unbounded abundance of choices for features

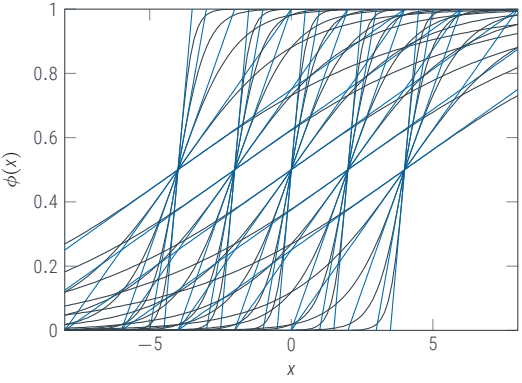


$$p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi_x^\intercal \mu, \phi_x \Sigma \phi_x)$$

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I) = \mathcal{N}(y; f_X, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x;\ \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),\ \phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi$$

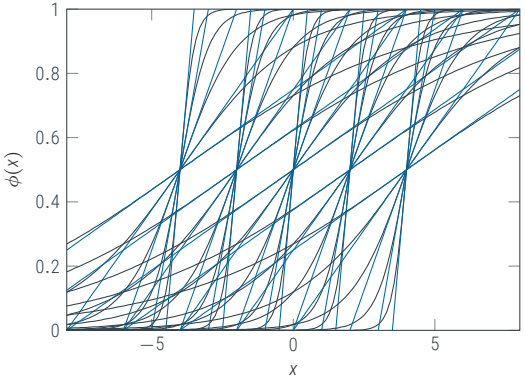
$$p(w \mid y, \phi) = \frac{p(y \mid w, \phi)p(w \mid \phi)}{p(y \mid \phi)}$$



- There is an infinite-dimensional space of feature functions to choose from
- Maybe we can restrict to a finite-dimensional sub-space and **search** in there? Say

$$\phi_i(x; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\frac{x-\theta_1}{\theta_2})}$$

$$p(w \mid y, \phi) = \frac{p(y \mid w, \phi) p(w \mid \phi)}{p(y \mid \phi)}$$

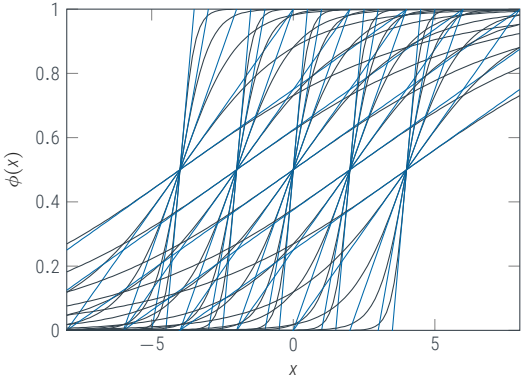


- There is an infinite-dimensional space of feature functions to choose from
- Maybe we can restrict to a finite-dimensional sub-space and **search** in there? Say

$$\phi_i(x; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\frac{x - \theta_1}{\theta_2})}$$

- θ_1, θ_2 are just unknown parameters!
- So can we infer them just like $\boldsymbol{w}$?

$$p(w \mid y, \phi) = \frac{p(y \mid w, \phi) p(w \mid \phi)}{p(y \mid \phi)}$$



- There is an infinite-dimensional space of feature functions to choose from
- Maybe we can restrict to a finite-dimensional sub-space and **search** in there? Say

$$\phi_i(x; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\frac{x - \theta_1}{\theta_2})}$$

- θ_1, θ_2 are just unknown parameters!
- So can we infer them just like $\boldsymbol{w}$?
- Yes, but not as easily: the likelihood

$$p(\boldsymbol{y} \mid w, \theta) = \mathcal{N}(\boldsymbol{y}; \phi(\boldsymbol{x}; \boldsymbol{\theta})^\intercal w, \sigma^2)$$

contains a **non-linear** map of $\boldsymbol{\theta}$.

$$p(f \mid \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta})}{\int p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta}) \, df} = \frac{p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta})}{p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})}$$

- **Model parameters** like θ are also known as **hyper-parameters**.
- This is largely a computational, practical distinction:

data	are observed	$\rightarrowtriangle$ **condition**
variables	are the things we care about	$\rightarrowtriangle$ **full probabilistic treatment**
parameters	are the things we have to deal with to get the model right	$\rightarrowtriangle$ **integrate out**
hyper-parameters	are the top-level, too expensive to properly infer	$\rightarrowtriangle$ **fit**

The **model evidence** in Bayes' Theorem is the (marginal) **likelihood** for the model. So we would like

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}) = \frac{p(\boldsymbol{y} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{\int p(\boldsymbol{y} \mid \boldsymbol{\theta}') p(\boldsymbol{\theta}') \, d\boldsymbol{\theta}'}$$

$$p(f \mid \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{\theta}) = \frac{p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta})}{\int p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta}) \, df} = \frac{p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta})}{p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})}$$

- For Gaussians, die evidence has **analytic form**:

$$\underbrace{\mathcal{N}(\boldsymbol{y}; \phi_X^{\boldsymbol{\theta}\mathsf{T}} w, \Lambda)}_{p(\boldsymbol{y}|f,\boldsymbol{x},\boldsymbol{\theta})} \cdot \underbrace{\mathcal{N}(w, \mu, \Sigma)}_{p(f)} = \underbrace{\mathcal{N}(w; m_{\text{post}}^{\boldsymbol{\theta}}, V_{\text{post}}^{\boldsymbol{\theta}})}_{p(f|\boldsymbol{y},\boldsymbol{x},\boldsymbol{\theta})} \cdot \underbrace{\mathcal{N}(\boldsymbol{y}; \phi_X^{\boldsymbol{\theta}\mathsf{T}} \mu, \phi_X^{\boldsymbol{\theta}\mathsf{T}} \Sigma \phi_X^{\boldsymbol{\theta}} + \Lambda)}_{p(\boldsymbol{y}|\boldsymbol{\theta},\boldsymbol{x})}$$

- **BUT:** It's not a linear function of $\boldsymbol{\theta}$, so analytic Gaussian inference is not available!

Computational complexity is *the* principal challenge of probabilistic reasoning.

The Toolbox

Framework:

$$\int p(x_1, x_2)\, dx_2 = p(x_1) \qquad p(x_1, x_2) = p(x_1 \mid x_2) p(x_2) \qquad p(x \mid y) = \frac{p(y \mid x) p(x)}{p(y)}$$

Modelling:

- Directed Graphical Models
- Gaussian Distributions
- hierarchical models
-
-
-
-

Computation:

- Monte Carlo
- Linear algebra / Gaussian inference
- Maximum likelihood / Maximum a-posteriori
-
-
-

ML / MAP in Practice

Finding the "best fit" θ in Gaussian models

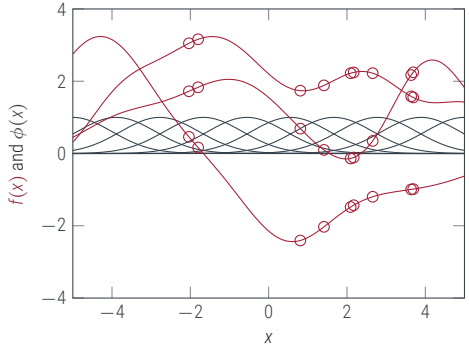
[e.g. DJC MacKay, *The evidence framework applied to classification networks*, 1992]

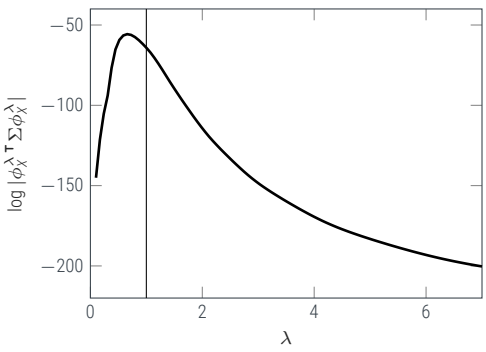
$$
\begin{aligned}
\hat{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \int p(\boldsymbol{y} \mid f, \boldsymbol{x}, \boldsymbol{\theta}) p(f \mid, \boldsymbol{\theta})\, df \\
&= \arg\max_{\boldsymbol{\theta}} \mathcal{N}(\boldsymbol{y}; \quad \phi_X^{\boldsymbol{\theta}\intercal}\mu, \quad \phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda) \\
&= \arg\max_{\boldsymbol{\theta}} \log \mathcal{N}(\boldsymbol{y}; \quad \phi_X^{\boldsymbol{\theta}\intercal}\mu, \quad \phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda) \\
&= \arg\min_{\boldsymbol{\theta}} -\log \mathcal{N}(\boldsymbol{y}; \quad \phi_X^{\boldsymbol{\theta}\intercal}\mu, \quad \phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda) \\
&= \arg\min_{\boldsymbol{\theta}} \frac{1}{2} \left(\underbrace{(\boldsymbol{y} - \phi_X^{\boldsymbol{\theta}\intercal}\mu)^\intercal \left(\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda\right)^{-1} (\boldsymbol{y} - \phi_X^{\boldsymbol{\theta}\intercal}\mu)}_{\text{square error}} + \underbrace{\log \left|\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda\right|}_{\text{model complexity / Occam factor}} \right) + \frac{N}{2}\log 2\pi
\end{aligned}
$$


$\log \left| \phi_X^{\boldsymbol{\theta}\mathsf{T}} \Sigma \phi_X^{\boldsymbol{\theta}} + \Lambda \right|$
Numquam ponenda est pluralitas sine necessitate.
Plurality must never be posited without necessity.
William of Occam
(1285 (Occam, Surrey)–1349 (Munich, Bavaria))
stained-glass window by Lawrence Lee

What is Model Complexity?

The Occam factor is not always straightforward



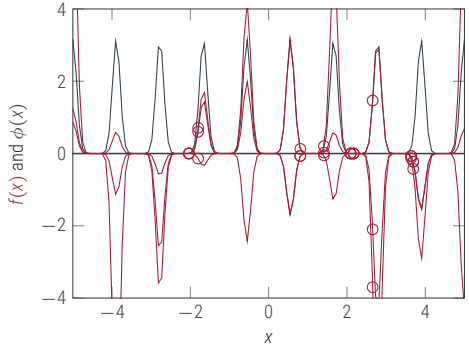


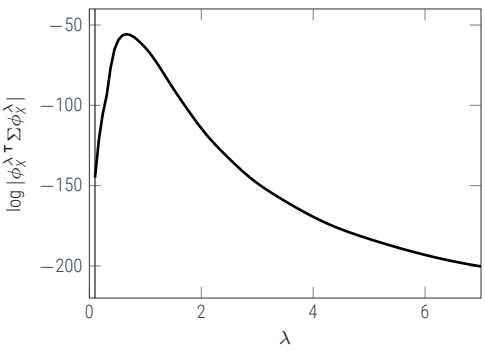
$$\log \left| \phi_X^{\boldsymbol{\theta}\mathsf{T}} \Sigma \phi_X^{\boldsymbol{\theta}} + \Lambda \right|$$

measures model complexity as the "volume" of hypotheses covered by the joint Gaussian distribution.

What is Model Complexity?

The Occam factor is not always straightforward



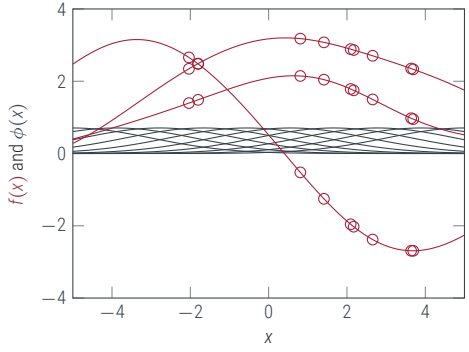


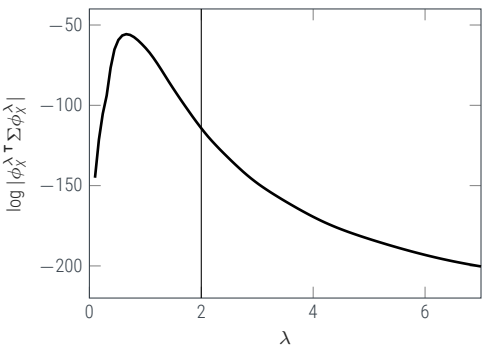
$$\log \left| \phi_X^{\boldsymbol{\theta}\mathsf{T}} \Sigma \phi_X^{\boldsymbol{\theta}} + \Lambda \right|$$

measures model complexity as the "volume" of hypotheses covered by the joint Gaussian distribution.

What is Model Complexity?

The Occam factor is not always straightforward



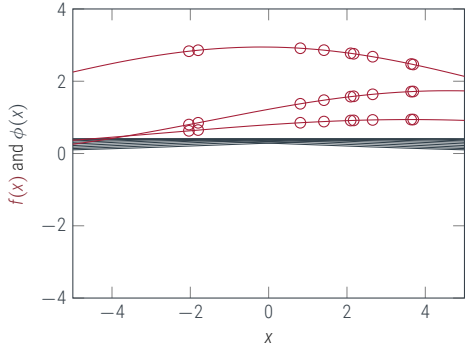


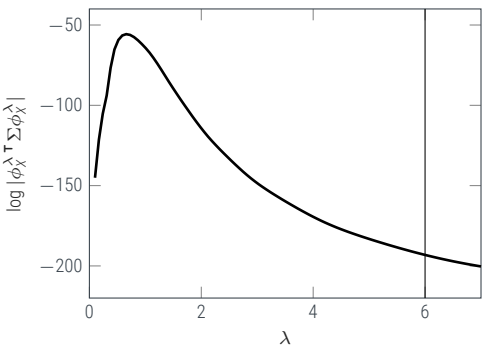
$$\log \left| \phi_X^{\boldsymbol{\theta}\mathsf{T}} \Sigma \phi_X^{\boldsymbol{\theta}} + \Lambda \right|$$

measures model complexity as the "volume" of hypotheses covered by the joint Gaussian distribution.

What is Model Complexity?

The Occam factor is not always straightforward



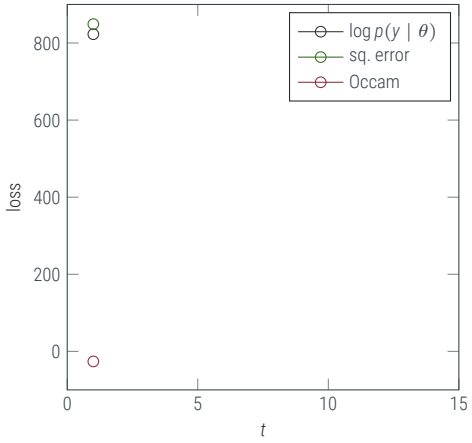


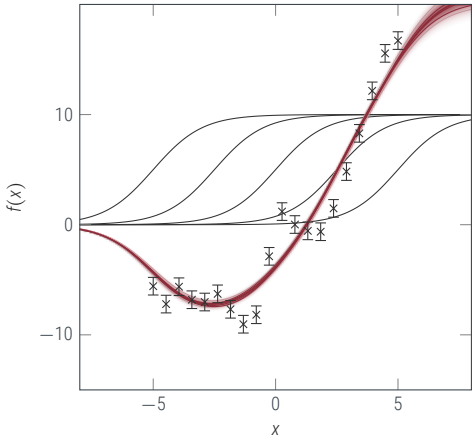
$$\log \left| \phi_X^{\boldsymbol{\theta}\,\mathsf{T}} \Sigma \phi_X^{\boldsymbol{\theta}} + \Lambda \right|$$

measures model complexity as the "volume" of hypotheses covered by the joint Gaussian distribution.

Type II Inference

Fitting a probabilistic model by maximum marginal likelihood





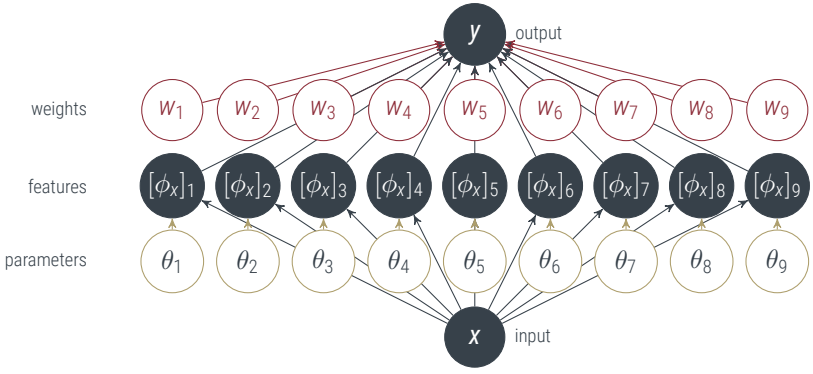
- *Parameters* $\boldsymbol{\theta}$ that affect the model should ideally be part of the inference process. The *evidence*

$$p(y \mid \boldsymbol{\theta}) = \int p(y \mid f, \boldsymbol{\theta}) p(f \mid \boldsymbol{\theta}) \, df$$

  (the denominator in Bayes' theorem) is the ("type-II" or "marginal") *likelihood* for $\boldsymbol{\theta}$
- If analytic inference on $\boldsymbol{\theta}$ is intractable (which it usually is), $\boldsymbol{\theta}$ can be *fitted* by "type-II" maximum likelihood (or maximum a-posteriori).
- Bayesian inference still has effects here because the marginal likelihood gives rise to complexity penalties / Occam factors.

A Structural Observation

Graphical Model



A linear Gaussian regressor is a **single hidden layer** neural network, with quadratic output loss, and fixed input layer. Hyperparameter-fitting corresponds to training the input layer. The usual way to train such network, however, does not include the Occam factor.

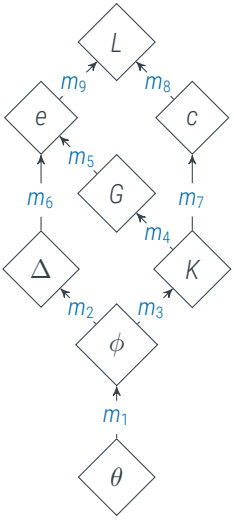
What does the Optimizer need from us?

A bit of algorithmic wizardry

$$L(\theta) = \frac{1}{2}\Bigg(\underbrace{(y - \phi_x^{\theta\intercal}\mu)^\intercal \underbrace{\Bigg(\underbrace{\phi_x^{\theta\intercal}\Sigma\phi_x^{\theta}}_{=:K} + \Lambda\Bigg)^{-1}}_{=:G} \overbrace{(y - \phi_x^{\theta\intercal}\mu)}^{=:\Delta}}_{=:e} + \underbrace{\log\left|\phi_x^{\theta\intercal}\Sigma\phi_x^{\theta} + \Lambda\right|}_{=:c} \Bigg)$$

What does the Optimizer need from us?

Automatic Differentiation



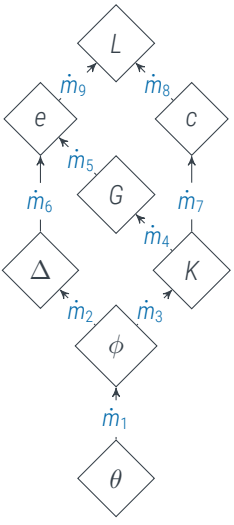
$$L(\theta) = \frac{1}{2}\Bigg(\underbrace{(y - \phi_X^{\boldsymbol{\theta}\intercal}\mu)^\intercal \underbrace{\Bigg(\underbrace{\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}}}_{=:K} + \Lambda\Bigg)^{-1}}_{=:G} \overbrace{(y - \phi_X^{\boldsymbol{\theta}\intercal}\mu)}^{=:\Delta}}_{=:e} + \underbrace{\log\left|\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda\right|}_{=:c} \Bigg)$$

$$= m_9 + m_8 = (m_6{}^\intercal m_5 m_6) + \log|m_7 + \Lambda|$$

$$= \dots$$

What does the Optimizer need from us?

Automatic Differentiation — Forward Mode

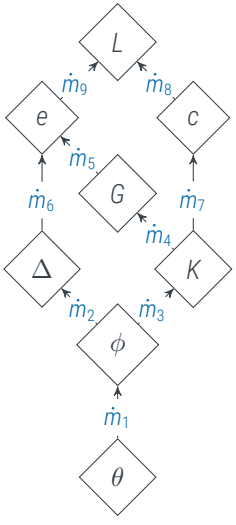


$$L(\theta) = \underbrace{\frac{1}{2}\Bigg(\underbrace{(\boldsymbol{y} - \phi_X^{\boldsymbol{\theta}\intercal}\mu)^\intercal \underbrace{\Bigg(\underbrace{\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}}}_{=:K} + \Lambda \Bigg)^{-1}}_{=:G} \overbrace{(\boldsymbol{y} - \phi_X^{\boldsymbol{\theta}\intercal}\mu)}^{=:\Delta}}_{=:e} + \underbrace{\log\left|\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda\right|}_{=:c} \Bigg)}$$

$$\begin{aligned}
\frac{\partial L}{\partial \theta} &= \frac{\partial L}{\partial e}\frac{\partial e}{\partial \theta} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial \theta} = \dot{m}_9\frac{\partial e}{\partial \theta} + \dot{m}_8\frac{\partial c}{\partial \theta} = \dot{m}_9\left(\frac{\partial e}{\partial \Delta}\frac{\partial \Delta}{\partial \theta} + \frac{\partial e}{\partial G}\frac{\partial G}{\partial \theta}\right) + \dot{m}_8\frac{\partial c}{\partial K}\frac{\partial K}{\partial \theta} \\
&= \dot{m}_9\left(\dot{m}_6\frac{\partial \Delta}{\partial \theta} + \dot{m}_5\frac{\partial G}{\partial \theta}\right) + \dot{m}_8\dot{m}_7\frac{\partial K}{\partial \theta} = \dot{m}_9\left(\dot{m}_6\frac{\partial \Delta}{\partial \phi}\frac{\partial \phi}{\partial \theta} + \dot{m}_5\frac{\partial G}{\partial K}\frac{\partial K}{\partial \theta}\right) + \dot{m}_8\dot{m}_7\frac{\partial K}{\partial \theta} \\
&= \dot{m}_9\dot{m}_6\dot{m}_2\frac{\partial \phi}{\partial \theta} + (\dot{m}_9\dot{m}_5\dot{m}_4 + \dot{m}_8\dot{m}_7)\frac{\partial K}{\partial \theta} = \left(\dot{m}_9\dot{m}_6\dot{m}_2 + (\dot{m}_9\dot{m}_5\dot{m}_4 + \dot{m}_8\dot{m}_7)\frac{\partial K}{\partial \phi}\right)\frac{\partial \phi}{\partial \theta} \\
&= (\dot{m}_9\dot{m}_6\dot{m}_2 + (\dot{m}_9\dot{m}_5\dot{m}_4 + \dot{m}_8\dot{m}_7)\,\dot{m}_3)\frac{\partial \phi}{\partial \theta}\frac{\partial \theta}{\partial \theta} \\
&= (\dot{m}_9\dot{m}_6\dot{m}_2 + (\dot{m}_9\dot{m}_5\dot{m}_4 + \dot{m}_8\dot{m}_7)\,\dot{m}_3)\,\dot{m}_1 1
\end{aligned}$$

What does the Optimizer need from us?

Automatic Differentiation — Forward Mode



$$L(\theta) = \frac{1}{2}\Bigg(\underbrace{(y - \phi_X^{\boldsymbol{\theta}\intercal}\mu)^\intercal \underbrace{\Big(\underbrace{\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}}}_{=:K} + \Lambda\Big)^{-1}}_{=:G} \overbrace{(y - \phi_X^{\boldsymbol{\theta}\intercal}\mu)}^{=:\Delta}}_{=:e} + \underbrace{\log\left|\phi_X^{\boldsymbol{\theta}\intercal}\Sigma\phi_X^{\boldsymbol{\theta}} + \Lambda\right|}_{=:c} \Bigg)$$

$$\dot{m}_9 = \frac{\partial L}{\partial e} = 1/2 \qquad \dot{m}_8 = \frac{\partial L}{\partial c} = 1/2 \qquad [\dot{m}_7]_{ij} = \frac{\partial c}{\partial K_{ij}} = K^{-1}_{ij}$$

$$[\dot{m}_6]_i = \frac{\partial e}{\partial \Delta_i} = 2[G\Delta]_i \qquad [\dot{m}_5]_{ij} = \frac{\partial e}{\partial G_{ij}} = \Delta_i\Delta_j \qquad [\dot{m}_4]_{ij,k\ell} = \frac{\partial G_{ij}}{\partial K_{k\ell}} = -G_{ik}G_{j\ell}$$

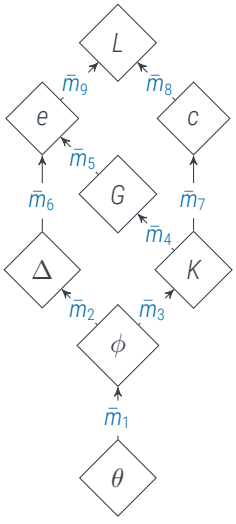
$$[\dot{m}_3]_{ij,ab} = \frac{\partial K_{ij}}{\partial \phi_{ab}} = \delta_{ia}[\Sigma\phi]_{bj} + \delta_{jb}[\Sigma\phi]_{kj}$$

$$[\dot{m}_2]_{i,ab} = \frac{\partial \Delta_i}{\partial \phi_{ab}} = -\delta_{ia}\mu_b \qquad [\dot{m}_1]_{ab,\ell} = \frac{\partial \phi_{ab}}{\partial \theta_\ell} = \text{your choice!}$$

What does the Optimizer need from us?

Automatic Differentiation – Backward Mode

[Seppo Linnainmaa, 1970]



$$L(\theta) = \frac{1}{2}\Bigg(\underbrace{(y - \phi_X^{\theta\intercal}\mu)^\intercal \underbrace{\Bigg(\underbrace{\phi_X^{\theta\intercal}\Sigma\phi_X^{\theta}}_{=:K} + \Lambda\Bigg)^{-1}}_{=:G} \overbrace{(y - \phi_X^{\theta\intercal}\mu)}^{=:\Delta}}_{=:e} + \underbrace{\log\left|\phi_X^{\theta\intercal}\Sigma\phi_X^{\theta} + \Lambda\right|}_{=:c} \Bigg)$$

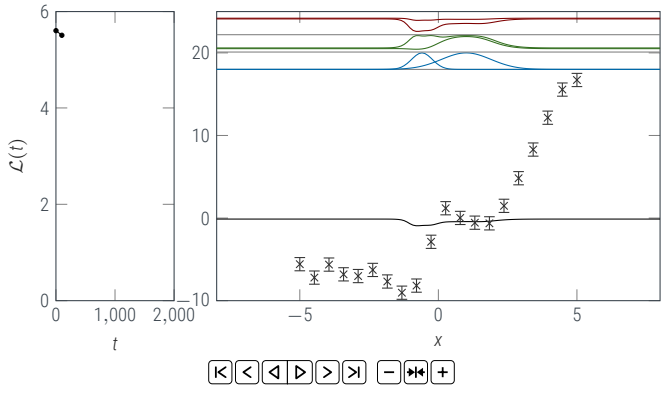
$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial \phi}\frac{\partial \phi}{\partial \theta} =: \bar{m}_1 = \left(\frac{\partial L}{\partial \Delta}\frac{\partial \Delta}{\partial \phi} + \frac{\partial L}{\partial K}\frac{\partial K}{\partial \phi}\right)\frac{\partial \phi}{\partial \theta} =: (\bar{m}_2 + \bar{m}_3)\frac{\partial \phi}{\partial \theta}$$

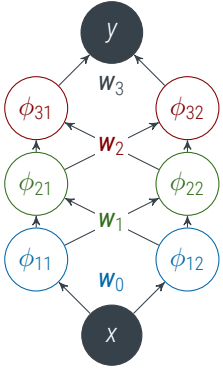
$$\bar{m}_2 = \frac{\partial L}{\partial e}\frac{\partial e}{\partial \Delta}\frac{\partial \Delta}{\partial \phi} =: \bar{m}_6\frac{\partial \Delta}{\partial \phi} \qquad \bar{m}_3 = \left(\frac{\partial L}{\partial G}\frac{\partial G}{\partial K} + \frac{\partial L}{\partial c}\frac{\partial c}{\partial K}\right)\frac{\partial K}{\partial \phi} =: (\bar{m}_4 + \bar{m}_7)\frac{\partial K}{\partial \phi}$$

$$\bar{m}_4 = \frac{\partial L}{\partial e}\frac{\partial e}{\partial G}\frac{\partial G}{\partial K} =: \bar{m}_5\frac{\partial G}{\partial K} \qquad \bar{m}_5 = \frac{\partial L}{\partial e}\frac{\partial e}{\partial G} =: \bar{m}_9\frac{\partial e}{\partial G} \qquad \bar{m}_6 = \frac{\partial L}{\partial e}\frac{\partial e}{\partial \Delta} =: \bar{m}_9\frac{\partial e}{\partial \Delta}$$

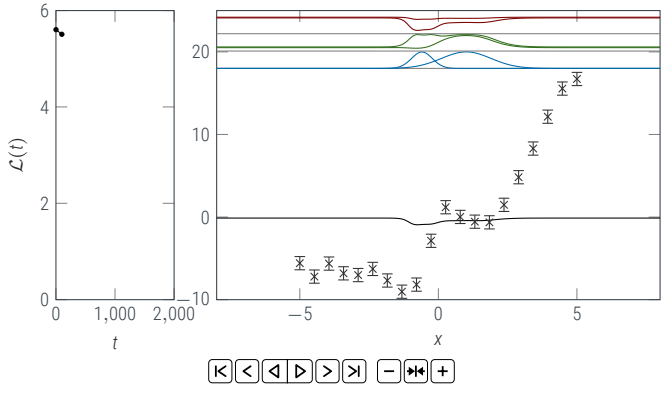
$$\bar{m}_7 = \frac{\partial L}{\partial c}\frac{\partial c}{\partial K} =: \bar{m}_8\frac{\partial c}{\partial K} \qquad \bar{m}_8 = \bar{m}_9 = 1/2$$

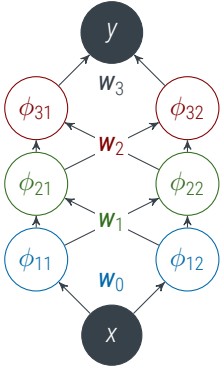
$\bar{w}_i = \frac{\partial L}{\partial \text{subgraph}_i}$ are known as *adjoints*. Traverse graph backward to collect the derivative. This is faster than forward-mode for single-output-many-input functions, but requires storing the above structure (known as a *Wengert list*). (cf. "Backpropagation")





$$\hat{f}(x, W) = \sum_{i=1}^{F} \phi_{3i}(x, w_{\text{lower}}) w_{3i} = \sum_{i} \phi_{3i} \left(\sum_{j} \phi_{2j} \left(\sum_{\ell} \phi_{1\ell}(w_{0\ell} x) w_{1\ell j} \right) w_{2ji} \right) w_{3i}$$





$$\hat{f}(x, \boldsymbol{W}) = \underset{\boldsymbol{W} \in \mathbb{R}^D}{\arg\min} \|\boldsymbol{y} - \hat{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{W})\|^2 + \alpha^2 \|\boldsymbol{W}\|^2 = \mathcal{L}(\boldsymbol{W})$$

$$W_{t+1} = W_t + \tau \nabla \mathcal{L}(W)$$

The connection to Deep Learning

Just go MAP all the way

If we consider multiple layers, we might as well not integrate out the final layer's weights

$$p(w, \theta \mid y) \propto p(y \mid w, \phi^{\theta}) p(w, \theta) = p(w, \theta) \cdot \prod_{i=1}^{n} p(y_i \mid w, \phi_i^{\theta}) = p(w, \theta) \cdot \prod_{i=1}^{n} \mathcal{N}(y_i; \phi_i^{\theta\intercal} w, \sigma^2)$$

The connection to Deep Learning

Just go MAP all the way

If we consider multiple layers, we might as well not integrate out the final layer's weights

$$p(w, \theta \mid y) \propto p(y \mid w, \phi^{\theta}) p(w, \theta) = p(w, \theta) \cdot \prod_{i=1}^{n} p(y_i \mid w, \phi_i^{\theta}) = p(w, \theta) \cdot \prod_{i=1}^{n} \mathcal{N}(y_i; \phi_i^{\theta\intercal} w, \sigma^2)$$

$$\arg\max_{w,\theta} p(w, \theta \mid y) = \arg\min_{w,\theta} -\log p(w, \theta \mid y)$$

The connection to Deep Learning

Just go MAP all the way

If we consider multiple layers, we might as well not integrate out the final layer's weights

$$p(w, \theta \mid y) \propto p(y \mid w, \phi^\theta) p(w, \theta) = p(w, \theta) \cdot \prod_{i=1}^{n} p(y_i \mid w, \phi_i^\theta) = p(w, \theta) \cdot \prod_{i=1}^{n} \mathcal{N}(y_i; \phi_i^{\theta\intercal} w, \sigma^2)$$

$$\begin{aligned}
\arg\max_{w,\theta} p(w, \theta \mid y) &= \arg\min_{w,\theta} -\log p(w, \theta \mid y) \\
&= \arg\min_{w,\theta} -\log p(w, \theta) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^{\theta\intercal} w\|^2 = \arg\min_{w,\theta} r(w, \theta) + \underbrace{\sum_{i=1}^{n} \ell_2(y_i; \theta, w)}_{\mathcal{L}(\theta, w)} \\
&= \arg\min_{w,\theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^{\theta\intercal} w\|^2 = \arg\min_{w,\theta} r(w, \theta) + \underbrace{\sum_{i=1}^{n} \ell_2(y_i; \theta, w)}_{\mathcal{L}(\theta, w)}
\end{aligned}$$

If we consider multiple layers, we might as well not integrate out the final layer's weights

$$p(w, \theta \mid y) \propto p(y \mid w, \phi^\theta) p(w, \theta) = p(w, \theta) \cdot \prod_{i=1}^{n} p(y_i \mid w, \phi_i^\theta) = p(w, \theta) \cdot \prod_{i=1}^{n} \mathcal{N}(y_i; \phi_i^{\theta\intercal} w, \sigma^2)$$

$$\begin{aligned}
\arg\max_{w,\theta} p(w, \theta \mid y) &= \arg\min_{w,\theta} -\log p(w, \theta \mid y) \\
&= \arg\min_{w,\theta} -\log p(w, \theta) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^{\theta\intercal} w\|^2 = \arg\min_{w,\theta} r(w, \theta) + \underbrace{\sum_{i=1}^{n} \ell_2(y_i; \theta, w)}_{\mathcal{L}(\theta, w)} \\
&= \arg\min_{w,\theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^{\theta\intercal} w\|^2 = \arg\min_{w,\theta} r(w, \theta) + \underbrace{\sum_{i=1}^{n} \ell_2(y_i; \theta, w)}_{\mathcal{L}(\theta, w)} \\
&\approx \arg\min_{w,\theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \frac{n}{b} \sum_{\beta=1}^{b} \|y_\beta - \phi_\beta^{\theta\intercal} w\|^2
\end{aligned}$$

The connection to Deep Learning

Just go MAP all the way

If we consider multiple layers, we might as well not integrate out the final layer's weights

$$p(w, \theta \mid y) \propto p(y \mid w, \phi^\theta) p(w, \theta) = p(w, \theta) \cdot \prod_{i=1}^{n} p(y_i \mid w, \phi_i^\theta) = p(w, \theta) \cdot \prod_{i=1}^{n} \mathcal{N}(y_i; \phi_i^{\theta\intercal} w, \sigma^2)$$

$$\begin{aligned}
\arg\max_{w,\theta} p(w, \theta \mid y) &= \arg\min_{w,\theta} -\log p(w, \theta \mid y) \\
&= \arg\min_{w,\theta} -\log p(w, \theta) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^{\theta\intercal} w\|^2 = \arg\min_{w,\theta} r(w, \theta) + \underbrace{\sum_{i=1}^{n} \ell_2(y_i; \theta, w)}_{\mathcal{L}(\theta, w)} \\
&= \arg\min_{w,\theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^{\theta\intercal} w\|^2 = \arg\min_{w,\theta} r(w, \theta) + \underbrace{\sum_{i=1}^{n} \ell_2(y_i; \theta, w)}_{\mathcal{L}(\theta, w)} \\
&\approx \arg\min_{w,\theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \frac{n}{b} \sum_{\beta=1}^{b} \|y_\beta - \phi_\beta^{\theta\intercal} w\|^2 \sim \mathcal{N}\left(r + \mathcal{L}(\theta, w), \mathcal{O}(b^{-1})\right)
\end{aligned}$$

- MAP inference does not capture **uncertainty** on parameters:
  - no posterior uncertainty from not fully identified parameters
  - no model capacity control in the evidence term
- A linear Gaussian regressor is a **single hidden layer** neural network, with quadratic output loss, and fixed input layer (deep networks can of course be treated in the same way). Hyperparameter-fitting corresponds to training the input layer. The usual way to train such network, however, does not include the Occam factor. Data sub-sampling can be used just as in other areas to speed up computations at the cost of reduced computational precision.
- All worries one may have about fitting or hand-picking features for Bayesian regression also apply to deep learning. By highlighting assumptions and piors, the probabilistic view forces us to address many problems directly, rather than obscuring them with notation and intuitions.
- **Automatic Differentiation (AD)** is a algorithmic tool that is just as helpful for Bayesian inference as it is for deep learning

It is possible to construct a point estimate for a Bayesian model, and to construct full posteriors for deep networks. The two domains are not separate, they are just different mental scaffolds. If you're hoping for a theory of deep learning, probability theory is a primary contender.

Summary:

- The features used for Gaussian linear regression can be **learnt** by **hierarchical Bayesian Inference**
- This is usually **intractable**. Instead, **approximate inference** methods are used
- For example, **maximum a-posteriori probability (MAP)** inference fits a point-estimate for feature parameters
- MAP inference is an **optimization** problem, and can thus be performed in the same way as other optimization-based ML approaches, including deep learning. That is, using the same optimizers (e.g. stochastic gradient descent), the same automatic differentiation frameworks (e.g. TensorFlow / pyTorch, etc.) and the same data subsampling techniques.

The different viewpoints (probabilistic / statistical / empirical ("deep")) on Machine Learning often overlap and inform each other. Understanding of Bayesian linear (Gaussian) regression can help us build a better intuition for deep learning, too.

Next lecture: Instead of *learning* a few features, sometimes we can get away with using *infinitely many* features.