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

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

\[ p(y \mid w, \phi_x) = \mathcal{N}(y; \phi_x^T w, \sigma^2 I) = \mathcal{N}(y; f_x, \sigma^2 I) \]

\[ p(f_x \mid y, \phi_x) = \mathcal{N}(f_x; \phi_x^T \mu + \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} (y - \phi_x^T \mu), \phi_x^T \Sigma \phi_x - \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} \phi_x^T \Sigma \phi_x) \]
Reminder: General Linear Regression

An unbounded abundance of choices for features

\[
p(w) = \mathcal{N}(w; \mu, \Sigma) \quad \Rightarrow \quad p(f) = \mathcal{N}(f_x; \phi^T \mu, \phi \Sigma \phi) \\
p(y \mid w, \phi_x) = \mathcal{N}(y; \phi^T \mu, \sigma^2 I) = \mathcal{N}(y; f_x, \sigma^2 I) \\
p(f_x \mid y, \phi_x) = \mathcal{N}(f_x; \phi^T \mu + \phi^T \Sigma \phi (\phi^T \Sigma \phi + \sigma^2 I)^{-1} (y - \phi^T \mu), \phi^T \Sigma \phi - \phi^T \Sigma \phi (\phi^T \Sigma \phi + \sigma^2 I)^{-1} \phi^T \Sigma \phi)^{(2)}
\]
Reminder: General Linear Regression

An unbounded abundance of choices for features

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

\[ p(y \mid w, \phi_x) = \mathcal{N}(y; \phi_x^T w, \sigma^2 I) = \mathcal{N}(y; f_x, \sigma^2 I) \]

\[ p(f_x \mid y, \phi_x) = \mathcal{N}(f_x; \phi_x^T \mu + \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} (y - \phi_x^T \mu), \phi_x^T \Sigma \phi_x - \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} \phi_x^T \Sigma \phi_x) \]
Reminder: General Linear Regression

An unbounded abundance of choices for features

$$p(w) = \mathcal{N}(w; \mu, \Sigma) \implies p(f) = \mathcal{N}(f_x; \phi^T_x \mu, \phi^T_x \Sigma \phi_x)$$

$$p(y \mid w, \phi_x) = \mathcal{N}(y; \phi^T_x w, \sigma^2 I) = \mathcal{N}(y; f_x, \sigma^2 I)$$

$$p(f_x \mid y, \phi_x) = \mathcal{N}(f_x; \phi^T_x \mu + \phi^T_x \Sigma \phi_x (\phi^T_x \Sigma \phi_x + \sigma^2 I)^{-1} (y - \phi^T_x \mu), \phi^T_x \Sigma \phi_x - \phi^T_x \Sigma \phi_x (\phi^T_x \Sigma \phi_x + \sigma^2 I)^{-1} \phi^T_x \Sigma \phi_x)$$
Reminder: General Linear Regression

An unbounded abundance of choices for features

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

\[
p(y \mid w, \phi_x) = \mathcal{N}(y; \phi_x^T w, \sigma^2 I) = \mathcal{N}(y; f_x, \sigma^2 I)
\]

\[
p(f_x \mid y, \phi_x) = \mathcal{N}(f_x; \phi_x^T \mu + \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} (y - \phi_x^T \mu), \phi_x^T \Sigma \phi_x - \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} \phi_x^T \Sigma \phi_x)
\]
Reminder: General Linear Regression

An unbounded abundance of choices for features

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

\[ p(y \mid w, \phi_x) = \mathcal{N}(y; \phi_x^T w, \sigma^2 I) = \mathcal{N}(y; f_x, \sigma^2 I) \]

\[ p(f_x \mid y, \phi_x) = \mathcal{N}(f_x; \phi_x^T \mu + \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} (y - \phi_x^T \mu), \phi_x^T \Sigma \phi_x - \phi_x^T \Sigma \phi_x (\phi_x^T \Sigma \phi_x + \sigma^2 I)^{-1} \phi_x^T \Sigma \phi_x) \]
Can we **Learn** the Features?

**Hierarchical Bayesian Inference**

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

- 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

\[
\phi_i(x; \theta) = \frac{1}{1 + \exp\left(-\frac{x - \theta_1}{\theta_2}\right)}
\]
Can we Learn the Features?

Hierarchical Bayesian Inference

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

- 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

\[ \phi_i(x; \theta) = \frac{1}{1 + \exp\left(-\frac{x - \theta_1}{\theta_2}\right)} \]

- \(\theta_1, \theta_2\) are just unknown parameters!
- So can we infer them just like \(w\)?
Can we Learn the Features?

Hierarchical Bayesian Inference

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

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

\[ \phi_i(x; \theta) = \frac{1}{1 + \exp\left(-\frac{x - \theta_1}{\theta_2}\right)} \]

\(\theta_1, \theta_2\) are just unknown parameters!

So can we infer them just like \(w\)?

Yes, but not as easily: the likelihood

\[ p(y \mid w, \theta) = \mathcal{N}(y; \phi(x; \theta)^T w, \sigma^2) \]

contains a non-linear map of \(\theta\).
Hierarchical Bayesian Inference

Bayesian model adaptation

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

- Model parameters like \( \theta \) are also known as hyper-parameters.
- This is largely a computational, practical distinction:
  - data are observed \( \rightarrow \) condition
  - variables are the things we care about \( \rightarrow \) full probabilistic treatment
  - parameters are the things we have to deal with to get the model right \( \rightarrow \) integrate out
  - hyper-parameters are the top-level, too expensive to properly infer \( \rightarrow \) fit

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

\[ p(\theta \mid y) = \frac{p(y \mid \theta)p(\theta)}{\int p(y \mid \theta')p(\theta') d\theta'} \]
Hierarchical Bayesian Inference
Bayesian model adaptation

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

For Gaussians, the evidence has **analytic form**:

\[ \mathcal{N}(y; \phi_x^T w, \Lambda) \cdot \mathcal{N}(w; \mu, \Sigma) = \mathcal{N}(w; m_{\text{post}}, V_{\text{post}}) \cdot \mathcal{N}(y; \phi_x^T \mu, \phi_x^T \Sigma \phi_x + \Lambda) \]

**BUT:** It’s not a linear function of \( \theta \), so analytic Gaussian inference is not available!

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

Framework:

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

Modelling:
- Directed Graphical Models
- Gaussian Distributions
- Hierarchical models

Computation:
- Monte Carlo
- Linear algebra / Gaussian inference
- Maximum likelihood / Maximum a-posteriori
Finding the “best fit” $\theta$ in Gaussian models

$\hat{\theta} = \arg \max_{\theta} p(y \mid x, \theta) = \arg \max_{\theta} \int p(y \mid f, x, \theta) p(f \mid \theta) \, df$

$= \arg \max_{\theta} \mathcal{N}(y; \phi_{X}^{\theta} \mu, \phi_{X}^{\theta} \Sigma_{\phi_{X}}^{\theta} + \Lambda)$

$= \arg \max_{\theta} \log \mathcal{N}(y; \phi_{X}^{\theta} \mu, \phi_{X}^{\theta} \Sigma_{\phi_{X}}^{\theta} + \Lambda)$

$= \arg \min_{\theta} \log \mathcal{N}(y; \phi_{X}^{\theta} \mu, \phi_{X}^{\theta} \Sigma_{\phi_{X}}^{\theta} + \Lambda)$

$= \arg \min_{\theta} \frac{1}{2} \left( (y - \phi_{X}^{\theta} \mu)^{T} \left( \phi_{X}^{\theta} \Sigma_{\phi_{X}}^{\theta} + \Lambda \right)^{-1} (y - \phi_{X}^{\theta} \mu) + \log \left| \phi_{X}^{\theta} \Sigma_{\phi_{X}}^{\theta} + \Lambda \right| \right) + \frac{N}{2} \log 2\pi$

square error

model complexity / Occam factor
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

\[
\log \phi^{\theta^T} \Sigma \phi^{\theta} + \Lambda
\]

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

\[
\log \left| \phi_x^\theta \Sigma \phi_x^\theta + \Lambda \right|
\]

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

\[ \log \left| \phi_X^\top \Sigma \phi_X + \Lambda \right| \]

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

\[ \log \phi_{X}^{T} \Sigma \phi_{X}^{T} + \Lambda \]

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 $\theta$ that affect the model should ideally be part of the inference process. The evidence

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

(the denominator in Bayes’ theorem) is the (“type-II” or “marginal”) likelihood for $\theta$.

If analytic inference on $\theta$ is intractable (which it usually is), $\theta$ 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 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

\[ L(\theta) = \frac{1}{2} \left( (y - \phi_\theta^T \mu)^T \left( \frac{\phi_\theta^T \Sigma \phi_\theta + \Lambda}{\phi_\theta^T \Sigma \phi_\theta + \Lambda} \right)^{-1} (y - \phi_\theta^T \mu) + \log \left| \phi_\theta^T \Sigma \phi_\theta + \Lambda \right| \right) \]

\[ =: G \]

\[ =: e \]

\[ =: \Delta \]
What does the Optimizer need from us?

Automatic Differentiation

\[
L(\theta) = \frac{1}{2} \left( (y - \phi_X^\theta \mu)^\top \left( \phi_X^\theta \Sigma \phi_X^\theta + \Lambda \right)^{-1} (y - \phi_X^\theta \mu) + \log \left| \phi_X^\theta \Sigma \phi_X^\theta + \Lambda \right| \right) =: c
\]

\[
=: \Delta
\]

\[
=: G
\]

\[
=: e
\]

\[
= m_9 + m_8 = (m_6^\top m_5 m_6) + \log |m_7 + \Lambda|
\]

\[
= \ldots
\]
What does the Optimizer need from us?

Automatic Differentiation — Forward Mode

\[ L(\theta) = \frac{1}{2} \left( (y - \phi^\theta X^\top \mu)^\top \left( \phi^\theta X^\top \Sigma \phi^\theta + \Lambda \right)^{-1} (y - \phi^\theta X^\top \mu) + \log \left| \phi^\theta X^\top \Sigma \phi^\theta + \Lambda \right| \right) \]

\[ \frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial e} \frac{\partial e}{\partial \theta} + \frac{\partial L}{\partial c} \frac{\partial c}{\partial \theta} = \dot{m}_9 \frac{\partial e}{\partial \theta} + \dot{m}_8 \frac{\partial c}{\partial \theta} = \dot{m}_9 \left( \frac{\partial e}{\partial \Delta} \frac{\partial \Delta}{\partial \theta} + \frac{\partial e}{\partial G} \frac{\partial G}{\partial \theta} \right) + \dot{m}_8 \frac{\partial c}{\partial \Delta} \frac{\partial \Delta}{\partial \theta}
\]

\[ = \dot{m}_9 \left( \dot{m}_6 \frac{\partial \Delta}{\partial \theta} + \dot{m}_5 \frac{\partial G}{\partial \theta} \right) + \dot{m}_8 \dot{m}_7 \frac{\partial K}{\partial \theta} = \dot{m}_9 \left( \dot{m}_6 \frac{\partial \Delta}{\partial \phi} \frac{\partial \phi}{\partial \theta} + \dot{m}_5 \frac{\partial G}{\partial \phi} \frac{\partial \phi}{\partial \theta} \right) + \dot{m}_8 \dot{m}_7 \frac{\partial K}{\partial \phi} \frac{\partial \phi}{\partial \theta}
\]

\[ = (\dot{m}_9 \dot{m}_6 \dot{m}_2 + (\dot{m}_9 \dot{m}_5 \dot{m}_4 + \dot{m}_8 \dot{m}_7) \dot{m}_3) \frac{\partial \phi}{\partial \theta} \frac{\partial \theta}{\partial \theta}
\]

\[ = (\dot{m}_9 \dot{m}_6 \dot{m}_2 + (\dot{m}_9 \dot{m}_5 \dot{m}_4 + \dot{m}_8 \dot{m}_7) \dot{m}_3) \dot{m}_1 
\]
What does the Optimizer need from us?

Automatic Differentiation — Forward Mode

\[ L(\theta) = \frac{1}{2} \left( (y - \phi^T \theta) \right)^T \left( \phi^T \sum \phi + \Lambda \right) \left( y - \phi^T \theta \right) + \log \left( \phi^T \sum \phi + \Lambda \right) \]

\[ \dot{m}_9 = \frac{\partial L}{\partial e} = \frac{1}{2} \]
\[ \dot{m}_8 = \frac{\partial L}{\partial c} = \frac{1}{2} \]
\[ [\dot{m}_7]_{ij} = \frac{\partial c}{\partial K_{ij}} = K_{ij}^{-1} \]
\[ [\dot{m}_6]_i = \frac{\partial e}{\partial \Delta_i} = 2[G \Delta]_i \]
\[ [\dot{m}_5]_{ij} = \frac{\partial e}{\partial G_{ij}} = \Delta_i \Delta_j \]
\[ [\dot{m}_4]_{ij, k\ell} = \frac{\partial G_{ij}}{\partial K_{k\ell}} = -G_{ik} G_{j\ell} \]
\[ [\dot{m}_3]_{ij, ab} = \frac{\partial K_{ij}}{\partial \phi_{ab}} = \delta_{ia} [\Sigma \phi]_{bj} + \delta_{jb} [\Sigma \phi]_{kj} \]
\[ [\dot{m}_2]_{i, ab} = \frac{\partial \Delta_i}{\partial \phi_{ab}} = -\delta_{ia} \mu_b \]
\[ [\dot{m}_1]_{ab, \ell} = \frac{\partial \phi_{ab}}{\partial \theta_{\ell}} = \text{your choice!} \]
What does the Optimizer need from us?

Automatic Differentiation — Backward Mode

\[ L(\theta) = \frac{1}{2} (y - \phi^T \mu)^T \left( \phi^T \sum \phi + \Lambda \right)^{-1} (y - \phi^T \mu) + \log \left( \phi^T \sum \phi + \Lambda \right) \]

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

\[ \frac{\partial L}{\partial e} \frac{\partial \Delta}{\partial \phi} =: \bar{m}_4 \frac{\partial \Delta}{\partial \phi} \quad \frac{\partial L}{\partial G} \frac{\partial \Delta}{\partial \phi} =: \bar{m}_5 \frac{\partial \Delta}{\partial \phi} \quad \frac{\partial L}{\partial c} \frac{\partial \Delta}{\partial \phi} =: \bar{m}_6 \frac{\partial \Delta}{\partial \phi} \quad \frac{\partial L}{\partial e} \frac{\partial \Delta}{\partial \phi} =: \bar{m}_7 \frac{\partial \Delta}{\partial \phi} \]

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

But not Bayesian deep networks

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

But not Bayesian deep networks

\[ \hat{f}(x, W) = \arg \min_{W \in \mathbb{R}^D} \|y - \hat{f}(x, W)\|^2 + \alpha^2 \|W\|^2 = \mathcal{L}(W) \]

\[ W_{t+1} = W_t + \tau \nabla \mathcal{L}(W) \]
If we consider multiple layers, we might as well not integrate out the final layer’s weights

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

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

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

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

arg max\, p(w, \theta \mid y) = arg min\, - \log p(w, \theta \mid y)

\[ = \arg min_{w,\theta} - \log p(w, \theta) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^\theta w\|^2 = \arg min_{w,\theta} r(w, \theta) + \sum_{i=1}^{n} \ell_2(y_i; \theta, w) \]

\[ = \arg min_{w,\theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^\theta w\|^2 = \arg min_{w,\theta} r(w, \theta) + \sum_{i=1}^{n} \ell_2(y_i; \theta, w) \]
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

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

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

\[
= \arg \min_{w, \theta} - \log p(w, \theta) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^\theta^\top w\|^2 = \arg \min_{w, \theta} r(w, \theta) + \sum_{i=1}^{n} \ell_2(y_i; \theta, w)
\]

\[
= \arg \min_{w, \theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \|y_i - \phi_i^\theta^\top w\|^2 = \arg \min_{w, \theta} r(w, \theta) + \sum_{i=1}^{n} \ell_2(y_i; \theta, w)
\]

\[
\approx \arg \min_{w, \theta} \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \frac{n}{b} \sum_{\beta=1}^{b} \|y_\beta - \phi_\beta^\theta^\top w\|^2
\]
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

\[
p(w, \theta | y) \propto p(y | w, \phi^\theta)p(w, \theta) = p(w, \theta) \cdot \prod_{i=1}^{n} p(y_i | w, \phi_i^\theta) = p(w, \theta) \cdot \prod_{i=1}^{n} \mathcal{N}(y_i; \phi_i^\theta \top w, \sigma^2)
\]

\[
\arg \max_w p(w, \theta | y) = \arg \min_w - \log p(w, \theta | y)
\]

\[
= \arg \min_w - \log p(w, \theta) + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \| y_i - \phi_i^\theta \top w \|^2 = \arg \min_w r(w, \theta) + \sum_{i=1}^{n} \ell_2(y_i; \theta, w)
\]

\[
= \arg \min_w \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \sum_{i=1}^{n} \| y_i - \phi_i^\theta \top w \|^2 = \arg \min_w r(w, \theta) + \sum_{i=1}^{n} \ell_2(y_i; \theta, w)
\]

\[
\approx \arg \min_w \sum_i w_i^2 + \sum_j \theta_j + \frac{1}{2\sigma^2} \frac{n}{b} \sum_{\beta=1}^{b} \| y_\beta - \phi_\beta^\theta \top w \|^2 \sim \mathcal{N} \left( r + \mathcal{L}(\theta, w), \mathcal{O}(b^{-1}) \right)
\]
Connections and Differences
Bayesian and Deep Learning

- 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 priors, the probabilistic view forces us to address many problems directly, rather than obscuring them with notation and intuitions.

- **Automatic Differentiation (AD)** is an 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.
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.