Nordic probabilistic AI school

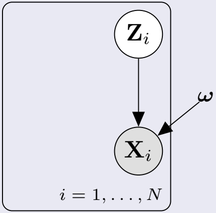
Variational Inference and Optimization

Helge Langseth, Andrés Masegosa, and Thomas Dyhre Nielsen

June 14, 2022

Deep Bayesian Learning – VAE

Model of interest



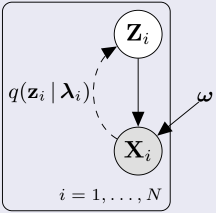
- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i \mid g_{\boldsymbol{\omega}}(\mathbf{z}_i))$, where g is a deep neural network.

$$p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim \text{Bernoulli}(\text{logits} = g_{\boldsymbol{\omega}}(\mathbf{z}_i))$$

- $g_{\boldsymbol{\omega}}(\mathbf{z}_i)$ plays the role of a **DECODER NETWORK**.
- **Goal:** Learn $\boldsymbol{\omega}$ to maximize the model's fit to $\mathcal{D}$.
  - We will cheat and find a **point estimate** for $\boldsymbol{\omega}$.

The Variational Auto Encoder (VAE)

Model of interest



- $p(\mathbf{z}_i)$ is (usually) an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i \,|\, g_{\boldsymbol{\omega}}(\mathbf{z}_i))$, where g is a deep neural network.

$$p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim \text{Bernoulli}(\text{logits} = g_{\boldsymbol{\omega}}(\mathbf{z}_i))$$

- $g_{\boldsymbol{\omega}}(\mathbf{z}_i)$ plays the role of a **DECODER NETWORK**.
- **Goal:** Learn $\boldsymbol{\omega}$ to maximize the model's fit to $\mathcal{D}$.
  - We will cheat and find a **point estimate** for $\boldsymbol{\omega}$.

Variational Inference

- We will need $p_{\boldsymbol{\omega}}(\mathbf{z}_i \,|\, \mathbf{x}_i)$ for each data-point $\mathbf{x}_i$:

$$p_{\boldsymbol{\omega}}(\mathbf{z}_i \,|\, \mathbf{x}_i) = \frac{p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \,|\, g_{\boldsymbol{\omega}}(\mathbf{z}_i))}{\int_{\mathbf{z}_i} p_{\boldsymbol{\omega}}(\mathbf{z}_i) \cdot p_{\boldsymbol{\omega}}(\mathbf{x}_i \,|\, g_{\boldsymbol{\omega}}(\mathbf{z}_i)) \, \mathrm{d}\mathbf{z}_{\mathrm{i}}}.$$

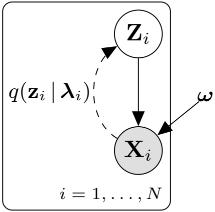
- **Initial plan:** Fit $q(\mathbf{z}_i \,|\, \boldsymbol{\lambda}_i)$ to $p_{\boldsymbol{\omega}}(\mathbf{z}_i \,|\, \mathbf{x}_i)$ using variational inference.

Initial plan:

- Optimize the ELBO

$$\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N) = -\mathbb{E}_q\left[\log \frac{\prod_{i=1}^N q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)}{\prod_{i=1}^N p_{\boldsymbol{\omega}}(\mathbf{z}_i, \mathbf{x}_i)}\right].$$

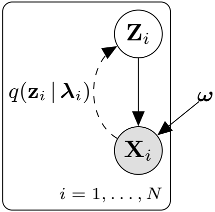
- A natural model for $q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)$ is a Gaussian with parameters $\boldsymbol{\lambda}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$.
- If $\mathbf{Z}_i$ is d-dim and we for simplicity assume diagonal $\boldsymbol{\Sigma}_i$, this still gives $2Nd$ variational parameters to learn.



Initial plan:

- Optimize the ELBO

$$\mathcal{L}(\boldsymbol{\omega}, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_N) = -\mathbb{E}_q \left[\log \frac{\prod_{i=1}^{N} q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)}{\prod_{i=1}^{N} p_{\boldsymbol{\omega}}(\mathbf{z}_i, \mathbf{x}_i)} \right].$$



- A natural model for $q(\mathbf{z}_i \mid \boldsymbol{\lambda}_i)$ is a Gaussian with parameters $\boldsymbol{\lambda}_i = \{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$.
- If $\mathbf{Z}_i$ is d-dim and we for simplicity assume diagonal $\boldsymbol{\Sigma}_i$, this still gives $2Nd$ variational parameters to learn.

A better plan

- Assume $g_{\boldsymbol{\omega}}(\mathbf{z})$ is "smooth": if $\mathbf{z}_i$ and $\mathbf{z}_j$ are "close", then so are $\mathbf{x}_i$ and $\mathbf{x}_j$.

  $\rightsquigarrow$ $\boldsymbol{\lambda}_i$ and $\boldsymbol{\lambda}_j$ should be "close" if $\mathbf{x}_i$ and $\mathbf{x}_j$ are "close".

- **Therefore:** Let's assume there exists a (smooth) function $h(\mathbf{x})$ so that $h(\mathbf{x}_i) = \boldsymbol{\lambda}_i$.
- $h(\cdot)$ is unavailable, so represent it using a deep neural net and learn the weights.
- $h(\mathbf{x}_i)$ plays the role of an **ENCODER NETWORK**.

Amortized inference:
To learn a model $h(\cdot)$, typically a deep neural network, so that $h(\mathbf{x}_i) = \boldsymbol{\lambda}_i$.
$h(\cdot)$ is parameterized with weights, often (abusing notation) denoted by $\boldsymbol{\lambda}$.

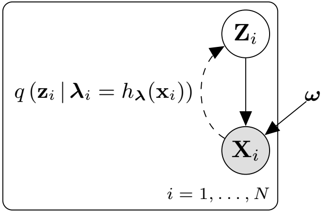
Note! Amortized inference is useful also outside VAEs!

Benefits:

- The $2Nd$ parameters $\{\boldsymbol{\lambda}_i\}_{i=1}^N$ are replaced by the fixed-sized vector $\boldsymbol{\lambda}$.
  - If N is large we may get a simpler learning problem.
- Smoothness of $h(\cdot)$ implies regularization.
- We only change the **parameterization**, not the model itself!

The full VAE approach:

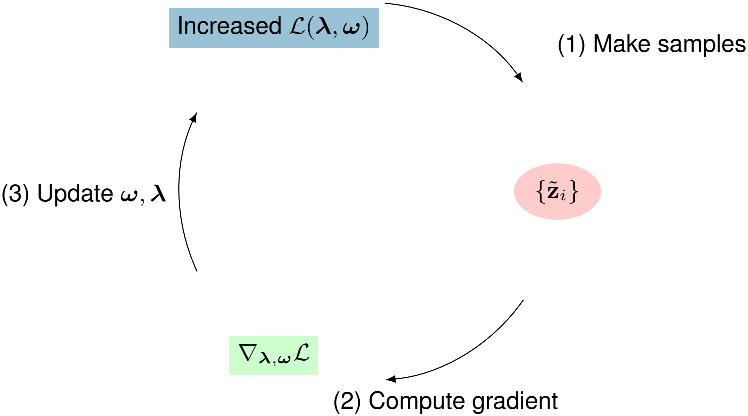
- $p(\mathbf{z}_i)$ is an isotropic Gaussian distribution.
- $p_{\boldsymbol{\omega}}(\mathbf{x}_i|\mathbf{z}_i) \sim$ Bernoulli(logits $= g_{\boldsymbol{\omega}}(\mathbf{z}_i)$), where $g_{\boldsymbol{\omega}}$ is a DNN with weights $\boldsymbol{\omega}$.
- $q(\mathbf{z}_i \,|\, \mathbf{x}_i, \boldsymbol{\lambda}) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where $\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\}$ is given by $h_{\boldsymbol{\lambda}}(\mathbf{x}_i)$. $h_{\boldsymbol{\lambda}}$ is a DNN with weights $\boldsymbol{\lambda}$.



Goal:

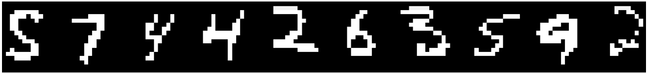
Learn **both** $\boldsymbol{\omega}$ and $\boldsymbol{\lambda}$ by maximizing the ELBO:

$$\mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\omega}) = -\mathbb{E}_q\left[\log \frac{q(\mathbf{z} \,|\, \mathbf{x}, \boldsymbol{\lambda})}{p_{\boldsymbol{\omega}}(\mathbf{z}, \mathbf{x} \,|\, \boldsymbol{\omega})}\right].$$

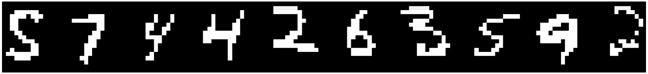


1. For each $\mathbf{x}_i$, sample M (typically 1) $\boldsymbol{\epsilon}$-values.
2. Calculate $\nabla_{\boldsymbol{\lambda},\boldsymbol{\omega}}\, \mathcal{L}(\boldsymbol{\lambda}, \boldsymbol{\omega})$ using the reparameterization-trick.
3. Update parameters using a standard DL optimizer (like Adam).

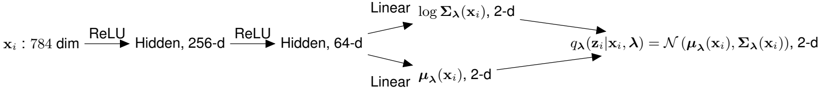
- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit (“0” – “9”).



- The model is learned from $N = 55.000$ training examples.
- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit (“0” – “9”).



- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}\left(\mathbf{0}_2, \mathbf{I}_2\right)$.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.

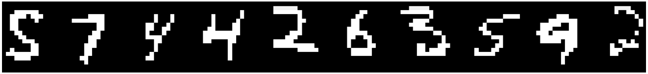


Fun with MNIST – The model

- The model is learned from $N = 55.000$ training examples.

- Each $\mathbf{x}_i$ is a binary vector of 784 pixel values.

- When seen as a 28×28 array, each $\mathbf{x}_i$ is a picture of a handwritten digit ("0" – "9").



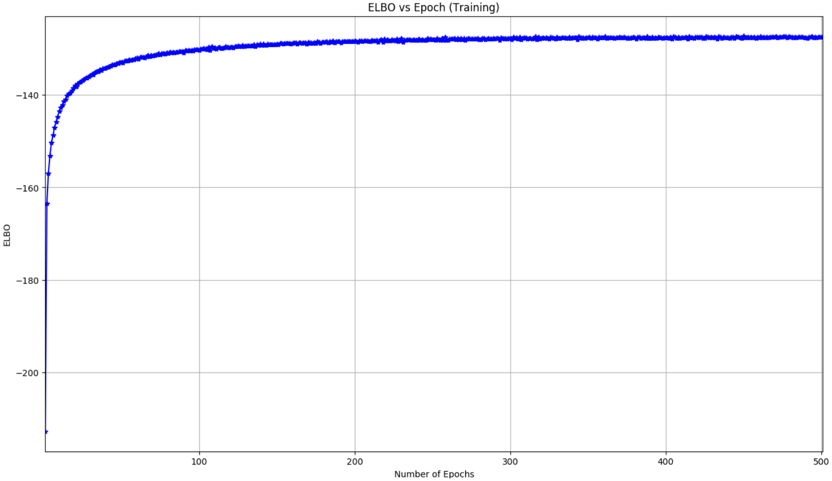
- Encoding is done in **two** dimensions. $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.

- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.

- The **decoder network** $\mathbf{Z} \rightsquigarrow \mathbf{X}$ is a $64 + 256$ neural net with ReLU units.

$$\mathbf{z}_i : 2 \text{ dim} \xrightarrow{\text{ReLU}} \text{Hidden, 64-d} \xrightarrow{\text{ReLU}} \text{Hidden, 256-d} \xrightarrow{\text{Linear}} \text{logit}(\mathbf{p}_i)\text{, 784-d} \longrightarrow p_{\boldsymbol{\omega}}(\mathbf{x}_i \,|\, \mathbf{z}_i, \boldsymbol{\omega}) = \text{Bernoulli}(\mathbf{p}_i)\text{, 784-d}$$

Learning progress; learning rate $\rho = 10^{-4}$, $M = 1$



Note! SGD algorithm uses the negative ELBO as loss.

Trying to reconstruct $\mathbf{x}_i$ by $\mathbb{E}_{p_\omega}\left[\mathbf{X} \,|\, \mathbf{Z} = \mathbb{E}_{q_\lambda}\left[\mathbf{Z} \,|\, \mathbf{x}_i\right]\right]$

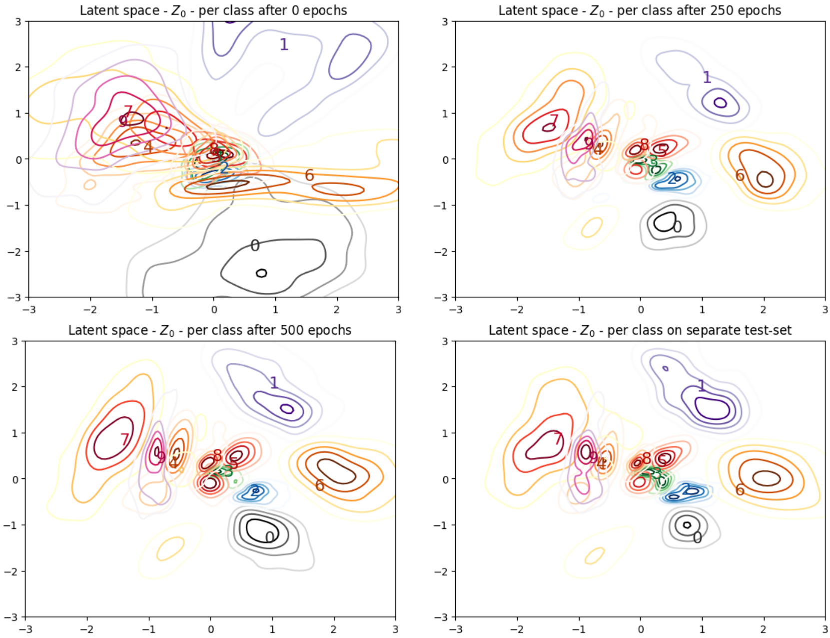
After 1 epoch

After 250 epochs

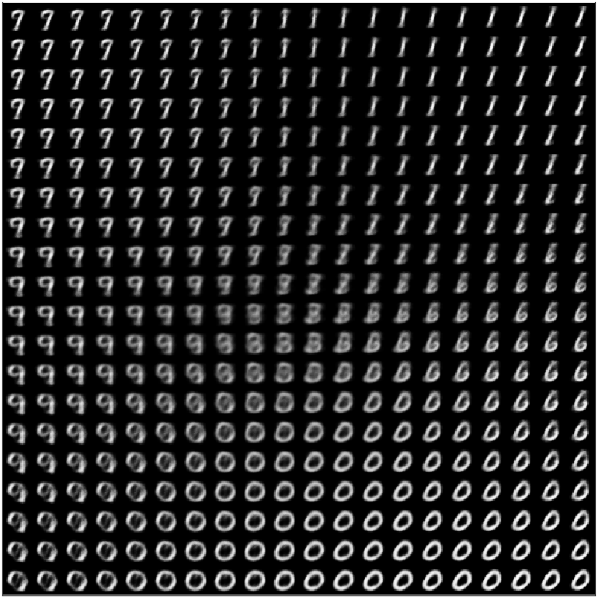
After 500 epoch

Using separate test-set

Averaged distribution over $\mathbf{Z}$ – per class

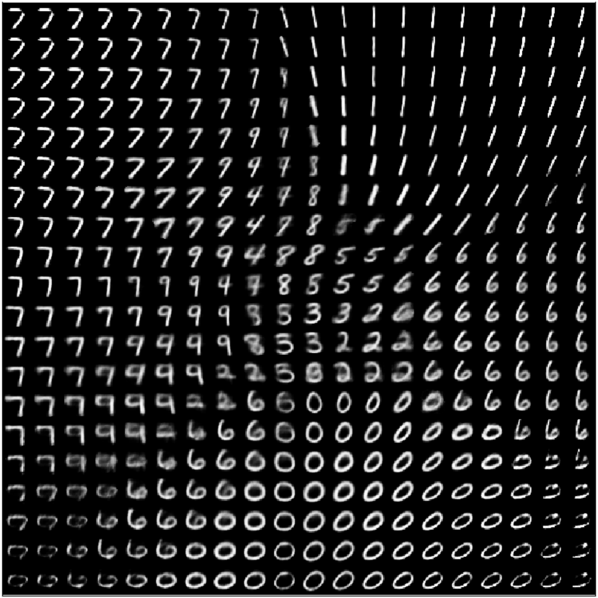


The picture manifold – $\mathbb{E}_{p_\omega}[\mathbf{X} \mid \mathbf{z}]$ for different values of $\mathbf{z}$

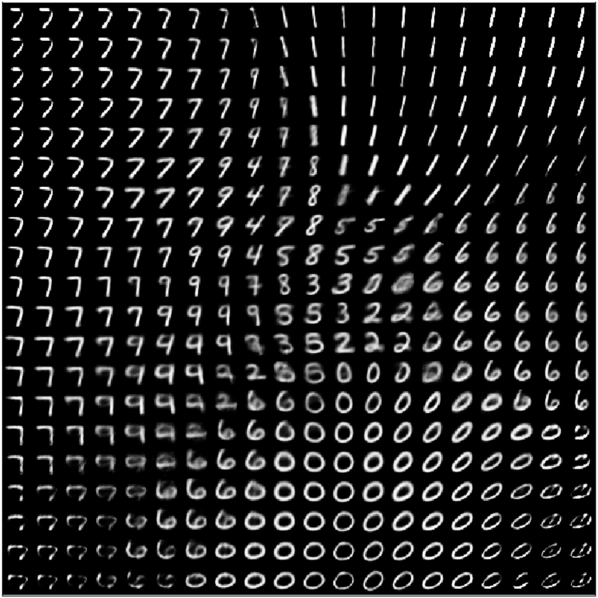


Manifold after 1 epoch

The picture manifold – $\mathbb{E}_{p_\omega}[\mathbf{X} \mid \mathbf{z}]$ for different values of $\mathbf{z}$



Manifold after 250 epochs



Manifold after 500 epochs

Variational Auto-Encoders in Pyro

Pyro specification of an encoder

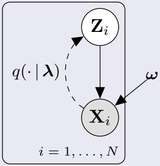
```python
class Decoder(nn.Module):
    def __init__(self, z_dim, hidden_dim):
        super(Decoder, self).__init__()
        # Setup the two linear transformations used
        self.fc1 = nn.Linear(z_dim, hidden_dim)
        self.fc21 = nn.Linear(hidden_dim, 784)
        # Setup the non-linearities
        self.softplus = nn.Softplus()
        self.sigmoid = nn.Sigmoid()

    def forward(self, z):
        # Define the forward computation on the latent z
        # First compute the hidden units
        hidden = self.softplus(self.fc1(z))
        # Return the parameter for the output Bernoulli
        # Each is of size batch_size x 784
        loc_img = self.sigmoid(self.fc21(hidden))
        return loc_img

# define the model p(x|z)p(z)
def model(self, x):
    # register PyTorch module `decoder` with Pyro
    pyro.module("decoder", self.decoder)
    with pyro.plate("data", x.shape[0]):
        # setup hyperparameters for prior p(z)
        z_loc = x.new_zeros(torch.Size((x.shape[0], self.z_dim)))
        z_scale = x.new_ones(torch.Size((x.shape[0], self.z_dim)))
        z = pyro.sample("latent", dist.Normal(z_loc, z_scale).to_event(1))
        # decode the latent code z
        loc_img = self.decoder.forward(z)
        # score against actual images
        pyro.sample("obs", dist.Bernoulli(loc_img).to_event(1),
                    obs=x.reshape(-1, 784))
```

Notes

- The PYRO.MODULE call registers the parameters in the decoder network with Pyro.
- The decoder network is a subclass of NN.MODULE; the class inherits methods such as PARAMETERS() and BACKWARD for calculating gradients.



Pyro specification of a decoder

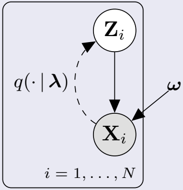
```
class Encoder(nn.Module):
    def __init__(self, z_dim, hidden_dim):
        super(Encoder, self).__init__()
        # Setup the three linear transformations used
        self.fc1 = nn.Linear(784, hidden_dim)
        self.fc21 = nn.Linear(hidden_dim, z_dim)
        self.fc22 = nn.Linear(hidden_dim, z_dim)
        # Setup the non-linearities
        self.softplus = nn.Softplus()

    def forward(self, x):
        # Define the forward computation on the image x
        # First shape the mini-batch to have pixels in
        # the rightmost dimension
        x = x.reshape(-1, 784)
        # then compute the hidden units
        hidden = self.softplus(self.fc1(x))
        # Return a mean vector and a (positive) square
        # root covariance each of size batch_size x z_dim
        z_loc = self.fc21(hidden)
        z_scale = torch.exp(self.fc22(hidden))
        return z_loc, z_scale

# define the guide (i.e. variational distribution) q(z|x)
def guide(self, x):
    # register PyTorch module `encoder` with Pyro
    pyro.module("encoder", self.encoder)
    with pyro.plate("data", x.shape[0]):
        # use the encoder to get the parameters used to define q(z|x)
        z_loc, z_scale = self.encoder.forward(x)
        # sample the latent code z
        pyro.sample("latent", dist.Normal(z_loc, z_scale).to_event(1))
```

Notes

- The encoder and guide follow the same structure as the encoder and model



Code Task: VAEs in Pyro

- Learn how a VAE is coded in Pryo.
- We provide a VAE with a **linear decoder**.
- **Exercise 1: Define a Non-Linear Decoder**
  - A MLP with a hidden layer with non-linearities (e.g. Relu).
- **Exercise 2: Explore the latent space**
  - Moving from linear to non-linear decoders with different capacity.
- Notebook:
  `Day2-Evening/students_VAE.ipynb`.

Conclusions

Recap & Conclusions

- **Bayesian Machine Learning**
  - Represents unobserved quantities using **distributions**
  - Models **epistemic** uncertainty using $p(\boldsymbol{\theta} \mid \mathcal{D})$

- **Bayesian Machine Learning**

- **Variational inference**
  - **Provides** $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$**:** A distributional approximation to $p(\boldsymbol{\theta} \mid \mathcal{D})$
  - **Objective:** $\arg\min_{\boldsymbol{\lambda}} \mathrm{KL}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda}) || p(\boldsymbol{\theta} \mid \mathcal{D})\right) \Leftrightarrow \arg\max_{\boldsymbol{\lambda}} \mathcal{L}\left(q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})\right)$
  - **Mean-field:** Divide and conquer strategy for high-dimensional posteriors
  - **Main caveat:** $q(\boldsymbol{\theta} \mid \boldsymbol{\lambda})$ underestimates the uncertainty of $p(\boldsymbol{\theta} \mid \mathcal{D})$

Recap & Conclusions

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
  - Analytic expressions for some models (i.e., conjugate exponential family)
  - CAVI is very **efficient and stable** if it can be used
  - In principle requires **manual derivation** of updating equations
    - There are **tools** to help (using *variational message passing*)

Recap & Conclusions

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
- **Gradient-based Variational Inference**
  - Provides the tools for VI over **arbitrary** probabilistic models
  - Directly integrates with the tools of deep learning
    - Automatic differentiation, sampling from standard distributions, and SGD
  - Sampling to approximate expectations: **Beware of the variance!**

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
- **Gradient-based Variational Inference**
- **Probabilistic programming languages**
  - PPLs fuel the “build – compute – critique – repeat” - cycle through
    - ease and flexibility of modelling
    - powerful inference engines
    - efficient model evaluations
  - Many available tools (Pyro, TF Probability, Infer.net, Turing.jl, . . .)

- **Bayesian Machine Learning**
- **Variational inference**
- **Coordinate Ascent Variational Inference**
- **Gradient-based Variational Inference**
- **Probabilistic programming languages**
- **What's next?**
  - The "VI toolbox" is reaching maturity
    - From *only* a research area to almost a *prerequisite* for Probabilistic AI
    - . . . yet there are still things to explore further!
  - Today's material should suffice to read (and write!) Prob-AI papers