

Variational inference

Partly based on material developed together with Helge Langseth

Andrés Masegosa and Thomas Dyhre Nielsen

June 16, 2021

Day 1: Probabilistic programming

- Introduction to probabilistic programming
- Probabilistic programming in Pyro

Day 2: Variational inference

- Recap of variational inference (variational inference as optimization)
- Derivation and implementation of selected examples
 - Bayesian linear regression
 - Factor analysis
 - ...

Day 3: Variational inference – cont'd

- Black box variational inference
- Variational inference in Pyro
- Variational auto-encoders

Black Box Variational Inference

VI inference as optimization

We can minimize (improve the variational approximation)

$$\text{KL}(q_{\lambda}(z), p(z \mid \mathbf{x}))$$

by maximizing the ELBO

$$\mathcal{L}(q) = \mathbb{E}_q \left[\log \frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z})} \right]$$

VI inference as optimization

We can minimize (improve the variational approximation)

$$\text{KL}(q_{\lambda}(z), p(z | \mathbf{x}))$$

by maximizing the ELBO

$$\mathcal{L}(q) = \mathbb{E}_q \left[\log \frac{p(\mathbf{z}, \mathbf{x})}{q(\mathbf{z})} \right]$$

The mean field assumption

We will often use the **mean field assumption**, which states that \mathcal{Q} consists of all distributions that *factorizes* according to the equation

$$q(\mathbf{z}) = \prod_i q_i(z_i)$$

↪ we can treat the variables independently.

Key requirements

We want the approach to be ...

“**Black Box**”: Not requiring tailor-made adaptations by the modeller.

Applicable: Useful independently of the underlying model assumptions.

Efficient: Utilize modelling assumptions, including the mean field assumption, to improve computational speed.

Algorithm: Maximize $\mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z})} \right]$ by gradient ascent

- Initialization:
 - $t \leftarrow 0$;
 - $\hat{\lambda}_0 \leftarrow$ random initialization;
- Repeat until negligible improvement in terms of $\mathcal{L}(q)$:
 - $t \leftarrow t + 1$;
 - $\hat{\lambda}_t \leftarrow \hat{\lambda}_{t-1} + \rho \nabla_{\lambda} \mathcal{L}(q)|_{\hat{\lambda}_{t-1}}$;

The algorithm requires that we can find

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_q \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z})} \right].$$

With a bit of pencil pushing it follows that

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z})} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z}) \right].$$

The algorithm requires that we can find

$$\nabla_{\lambda} \mathcal{L}(q) = \nabla_{\lambda} \mathbb{E}_q \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z})} \right].$$

With a bit of pencil pushing it follows that

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z})} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z}) \right].$$

Properties used for derivation

- ❶ $\nabla_{\lambda} (f(\mathbf{z}, \lambda) \cdot g(\mathbf{z}, \lambda)) = f(\mathbf{z}, \lambda) \cdot \nabla_{\lambda} g(\mathbf{z}, \lambda) + g(\mathbf{z}, \lambda) \nabla_{\lambda} f(\mathbf{z}, \lambda)$
- ❷ $\nabla_{\lambda} f(\mathbf{z}, \lambda) = f(\mathbf{z}, \lambda) \nabla_{\lambda} \log f(\mathbf{z}, \lambda)$
- ❸ $\mathbb{E}_{q_{\lambda}} [\nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \lambda)] = 0$ for a density function $q_{\lambda}(\mathbf{z} | \lambda)$

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z} | \boldsymbol{\lambda})} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \boldsymbol{\lambda}) \right].$$

- We only need access to the un-normalized $p_{\theta}(\mathbf{z}, \mathbf{x})$ – not $p_{\theta}(\mathbf{z} | \mathbf{x})$.

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z} | \lambda)} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \lambda) \right].$$

- We only need access to the un-normalized $p_{\theta}(\mathbf{z}, \mathbf{x})$ – not $p_{\theta}(\mathbf{z} | \mathbf{x})$.

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z} | \lambda)} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \lambda) \right].$$

- $q_{\lambda}(\mathbf{z})$ factorizes under MF, s.t. we can optimize per variable: $q_{\lambda_i}(z_i)$.

- We only need access to the un-normalized $p_{\theta}(\mathbf{z}, \mathbf{x})$ – not $p_{\theta}(\mathbf{z} | \mathbf{x})$.

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z} | \lambda)} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \lambda) \right].$$

- $q_{\lambda}(\mathbf{z})$ factorizes under **MF**, s.t. we can optimize per variable: $q_{\lambda_i}(z_i)$.
- We must calculate $\nabla_{\lambda} \log q(\mathbf{z} | \lambda)$, which is also known as the “score function”. This depends on the distributional family of $q(\cdot)$; can be precomputed for standard distributions.

Example

If $q_{\lambda}(z)$ follows a normal distribution ($\lambda = (\mu, \sigma)$):

$$\frac{1}{\sqrt{2\pi}\sigma^2} \exp \left(-\frac{(z - \mu)^2}{2\sigma^2} \right),$$

then

$$\nabla_{\mu} \log q_{\lambda}(z) = \frac{1}{\sigma^2} (z - \mu)$$

- We only need access to the un-normalized $p_{\theta}(\mathbf{z}, \mathbf{x})$ – not $p_{\theta}(\mathbf{z} | \mathbf{x})$.

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z} | \lambda)} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \lambda) \right].$$

- $q_{\lambda}(\mathbf{z})$ factorizes under **MF**, s.t. we can optimize per variable: $q_{\lambda_i}(z_i)$.
- We must calculate $\nabla_{\lambda} \log q(\mathbf{z} | \lambda)$, which is also known as the “score function”. This depends on the distributional family of $q(\cdot)$; can be precomputed for standard distributions.
- The expectation will be approximated using a sample $\{\mathbf{z}_1, \dots, \mathbf{z}_M\}$ generated from $q(\mathbf{z} | \lambda)$. Hence we require that we can **sample from** $q_{\lambda_i}(\cdot)$.

Calculating the gradient – Things to notice

- We only need access to the un-normalized $p_{\theta}(\mathbf{z}, \mathbf{x})$ – not $p_{\theta}(\mathbf{z} | \mathbf{x})$.

$$\nabla_{\lambda} \mathcal{L}(q) = \mathbb{E}_{q_{\lambda}} \left[\log \frac{p_{\theta}(\mathbf{z}, \mathbf{x})}{q_{\lambda}(\mathbf{z} | \lambda)} \cdot \nabla_{\lambda} \log q_{\lambda}(\mathbf{z} | \lambda) \right].$$

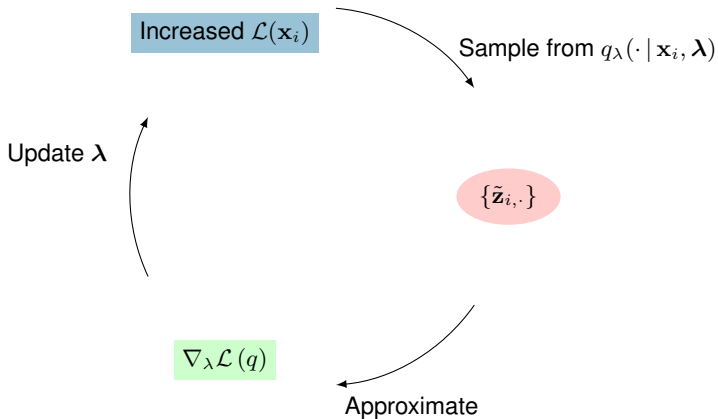
- $q_{\lambda}(\mathbf{z})$ factorizes under **MF**, s.t. we can optimize per variable: $q_{\lambda_i}(z_i)$.
- We must calculate $\nabla_{\lambda} \log q(\mathbf{z} | \lambda)$, which is also known as the “score function”. This depends on the distributional family of $q(\cdot)$; can be precomputed for standard distributions.
- The expectation will be approximated using a sample $\{\mathbf{z}_1, \dots, \mathbf{z}_M\}$ generated from $q(\mathbf{z} | \lambda)$. Hence we require that we can **sample from** $q_{\lambda_i}(\cdot)$.

Calculating the gradient – in summary

We have observed the datapoint \mathbf{x} , and our current estimate for λ_i is $\hat{\lambda}_i$. Then

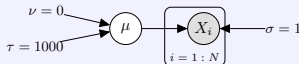
$$\nabla_{\lambda_i} \mathcal{L}(q)|_{\lambda=\hat{\lambda}_i} \approx \frac{1}{M} \sum_{j=1}^M \log \frac{p(z_{i,j}, \mathbf{x})}{q(z_{i,j} | \hat{\lambda}_i)} \cdot \nabla_{\lambda_i} \log q_i(z_{i,j} | \hat{\lambda}_i).$$

where $\{z_{i,1}, \dots, z_{i,M}\}$ are samples from $q_{\lambda_i}(\cdot | \hat{\lambda}_i)$.



Exercise: BBVI in Python

Consider the simple generative model:

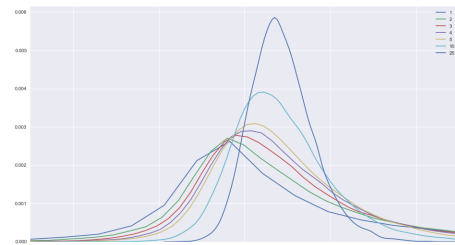


- Derive the BBVI estimate of the gradient for the variational parameters of $q(\mu) = \mathcal{N}(\lambda, 1)$.
- Implement the gradient estimate in the notebook

`students_BBVI.ipynb`

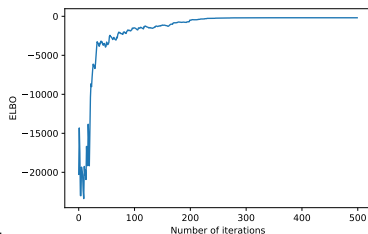
- Perform gradient ascent using your gradient implementation by running the notebook.

Density of gradient estimates



PDF for the gradient calculated at $\lambda = 9$, which is below the optimum ≈ 10 . Several values for M , the sample size used to generate the estimate, are shown.

Evolution of ELBO



Based on gradient estimates using 1 sample

`BBVI-full.ipynb`

- Since the gradient estimate is based on a random sample, it is meaningful to evaluate the estimators' "robustness" in terms of a density function.
- We would hope to see robust estimates, also for small M , and in particular high probability for moving in the correct direction (gradient larger than 0).
- This is not the case, which has lead to a major focus on **variance reduction techniques**: while important we will **not cover them here**.

Probabilistic programming: Variational inference in Pyro

Pyro

Pyro (pyro.ai) is a Python library for probabilistic modeling, inference, and criticism, integrated with PyTorch.

- Modeling:**
 - Directed graphical models
 - Neural networks (via `nn.Module`)
 - ...
- Inference:**
 - Variational inference – including BBVI, SVI
 - Monte Carlo – including Importance sampling and Hamiltonian Monte Carlo
 - ...
- Criticism:**
 - Point-based evaluations
 - Posterior predictive checks
 - ...

... and there are also many other possibilities

`Tensorflow` is integrating probabilistic thinking into its core, `InferPy` is a local alternative, etc.

Pyro models in general

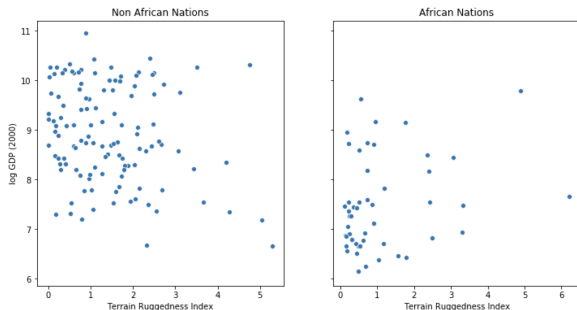
- observations \Leftrightarrow `pyro.sample` with the `obs` argument
- latent random variables \Leftrightarrow `pyro.sample`
- parameters \Leftrightarrow `pyro.param`

Simple example

```
1 #The observations
2 obs = {'sensor': torch.tensor(18.0)}
3
4 def model(obs):
5     temp = pyro.sample('temp', dist.Normal(15.0, 2.0))
6     sensor = pyro.sample('sensor', dist.Normal(temp, 1.0), obs=obs['sensor'])
```

Bayesian linear regression

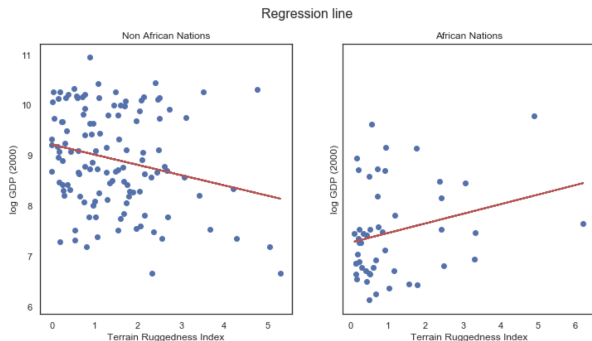
Real Data Example



Relationship between topographic heterogeneity and GDP per capita

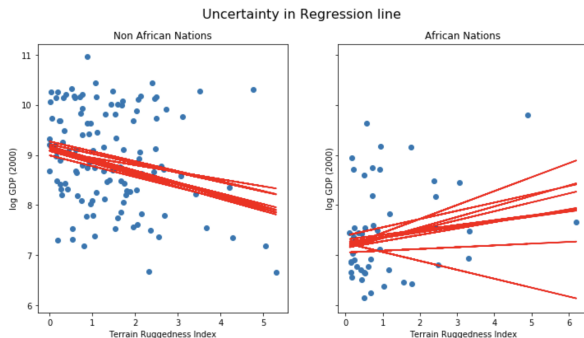
- Terrain ruggedness or bad geography is related to poorer economic performance outside of Africa.
- Rugged terrains have had a reverse effect on income for African nations.

Real Data Example



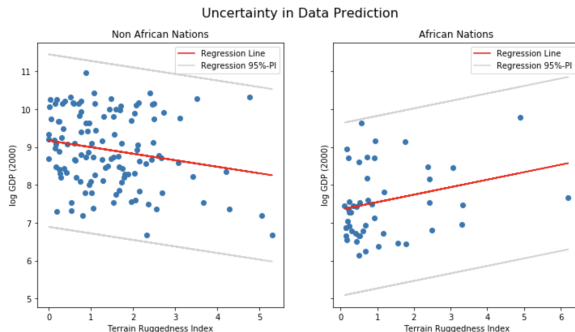
Linear Regression Model

- Negative slope for Non African Nations.
- Positive slope for African Nations.



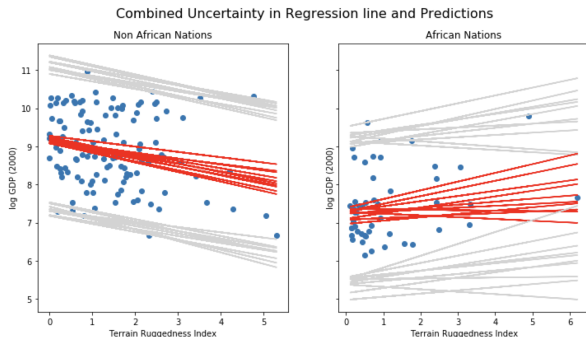
Bayesian Linear Regression Model

- Modeling uncertainty about the linear coefficients (epistemic uncertainty).



Bayesian Linear Regression Model

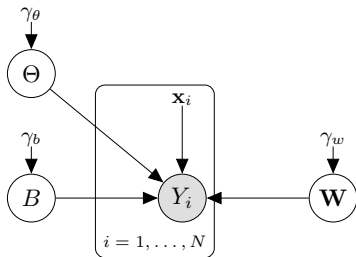
- Modeling uncertainty about the linear coefficients (epistemic uncertainty).
- Modeling data noise (aleatoric uncertainty)



Bayesian Linear Regression Model

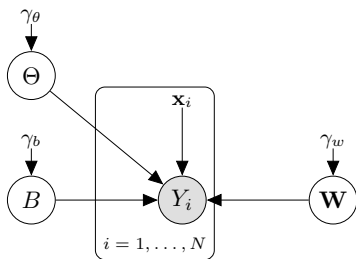
- Modeling uncertainty about the linear coefficients (epistemic uncertainty).
- Modeling data noise (aleatoric uncertainty)

The Bayesian linear regression model



- Num. of data dim: M
- Num. of data inst: N
- $Y_i \mid \{\mathbf{w}, \mathbf{x}_i, b, \theta\} \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i + b, 1/\Theta)$
- $\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \gamma_w^{-1} \mathbf{I}_{M \times M})$
- $B \sim \mathcal{N}(0, \gamma_b^{-1})$
- $\Theta \sim \mathcal{G}(1, 1)$

The Bayesian linear regression model



- Num. of data dim: M
- Num. of data inst: N
- $Y_i \mid \{\mathbf{w}, \mathbf{x}_i, b, \theta\} \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i + b, 1/\Theta)$
- $\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \gamma_w^{-1} \mathbf{I}_{M \times M})$
- $B \sim \mathcal{N}(0, \gamma_b^{-1})$
- $\Theta \sim \mathcal{G}(1, 1)$

The probability model

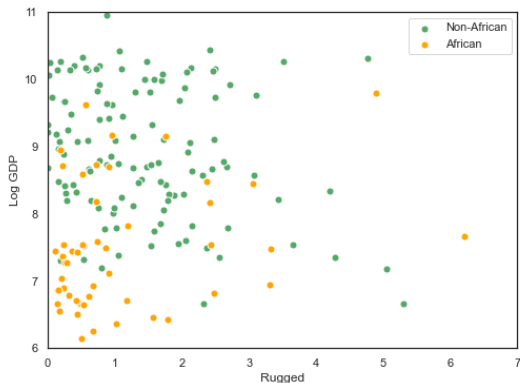
$$p(\cdot \mid \mathbf{x}, \gamma_w, \gamma_b, \gamma_\theta) = \prod_{i=1}^N p(y_i \mid \mathbf{x}_i, \mathbf{w}, b, \theta) p(\mathbf{w} \mid \gamma_w) p(b \mid \gamma_b) p(\theta \mid \gamma_\theta)$$

Notebook: Bayesian Linear Regression in Pyro

Day3/bayesian_linear_regression.ipynb

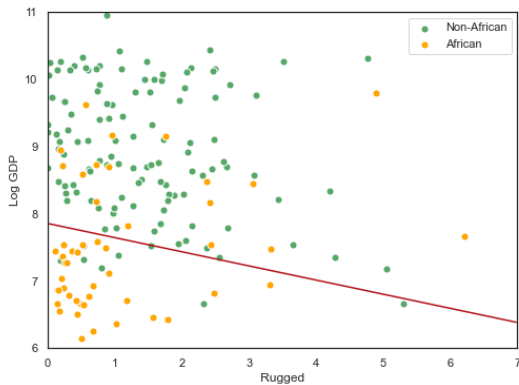
Bayesian logistic regression

Real Data Example



Predict whether a nation is African or not as a function of:

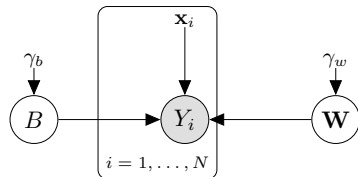
- Terrain ruggedness or bad geography of the nation.
- (Log) GDP of the nation.



Standard Logistic Regression Model

- No model uncertainty.

The Bayesian logistic regression model

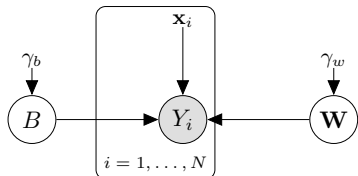


- Num. of data dim: M
- Num. of data inst: N
- $Y_i \mid \{\mathbf{w}, \mathbf{x}_i, b\} \sim \mathcal{B}(\text{logits} = \mathbf{w}^\top \mathbf{x}_i + b)$
- $\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \gamma_w^{-1} \mathbf{I}_{M \times M})$
- $B \sim \mathcal{N}(0, \gamma_b^{-1})$

The probability model

$$p(\cdot \mid \mathbf{x}, \gamma_w, \gamma_b) = \prod_{i=1}^N p(y_i \mid \mathbf{x}_i, \mathbf{w}, b) p(\mathbf{w} \mid \gamma_w) p(b \mid \gamma_b)$$

The Bayesian logistic regression model



- Num. of data dim: M
- Num. of data inst: N
- $Y_i \mid \{\mathbf{w}, \mathbf{x}_i, b\} \sim \mathcal{B}(\text{logits} = \mathbf{w}^\top \mathbf{x}_i + b)$
- $\mathbf{W} \sim \mathcal{N}(\mathbf{0}, \gamma_w^{-1} \mathbf{I}_{M \times M})$
- $B \sim \mathcal{N}(0, \gamma_b^{-1})$

The probability model

$$p(\cdot \mid \mathbf{x}, \gamma_w, \gamma_b) = \prod_{i=1}^N p(y_i \mid \mathbf{x}_i, \mathbf{w}, b) p(\mathbf{w} \mid \gamma_w) p(b \mid \gamma_b)$$

Exercise: Bayesian Logistic Regression in Pyro

- Play around with the first part of the notebook (Section 1)
- Complete the definition of the Bayesian Logistic Regression (Section 2.1).
- Learn the model and Play around with the evaluation (Section 2.2-2.5).

Day3/bayesian_logistic_regression.ipynb

Pyro's Guides

Guides

Definition:

- Guides are **arbitrary stochastic functions**.
- Guides produces samples for those variables of the model which are **not observed**.

Guides

Definition:

- Guides are **arbitrary stochastic functions**.
- Guides produces samples for those variables of the model which are **not observed**.

Guides are used for:

- Define the q **distributions** in variational settings.
- Define **inference networks** as in VAEs.
- Build **proposal distributions** in importance sampling, MCMC.
- ...

Guide requirements

Guide functions must satisfy these two criteria to be valid approximations for a particular model:

- 1 all unobserved (i.e., not conditioned) sample statements that appear in the model appear in the guide.
- 2 the guide has the same input signature as the model (i.e., takes the same arguments)

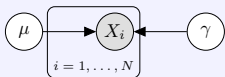
Example

```
1 #The observatons
2 obs = {'sensor': torch.tensor(18.0)}
3
4 def model(obs):
5     temp = pyro.sample('temp', dist.Normal(15.0, 2.0))
6     sensor = pyro.sample('sensor', dist.Normal(temp, 1.0), obs=obs['sensor'])
```

```
1 #The guide
2 def guide(obs):
3     a = pyro.param("mean", torch.tensor(0.0))
4     b = pyro.param("scale", torch.tensor(1.), constraint=constraints.positive)
5     temp = pyro.sample('temp', dist.Normal(a, b))
```

Exercise 1: Pyro implementation for a simple Gaussian model

Day3/student_simple_gaussian_model_pyro.ipynb



- $X_i \mid \{\mu, \gamma\} \sim \mathcal{N}(\mu, 1/\gamma)$
- $\mu \sim \mathcal{N}(0, \tau)$
- $\gamma \sim \text{Gamma}(\alpha, \beta)$

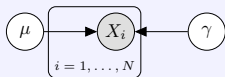
- Implement a pyro **model** and **guide** for the graphical model above.
- Specify appropriate **parameters** τ , α and β (e.g. try to use flat priors).
- Specify suitable **variational approximation** in the form of a Pyro guide.

$$q(\mu, \gamma) = \dots$$

- **Check** the differences with the notebook of Day 2.

Exercise 1: Pyro implementation for a simple Gaussian model

Day3/student_simple_gaussian_model_pyro.ipynb



- $X_i \mid \{\mu, \gamma\} \sim \mathcal{N}(\mu, 1/\gamma)$
- $\mu \sim \mathcal{N}(0, \tau)$
- $\gamma \sim \text{Gamma}(\alpha, \beta)$

- Implement a pyro **model** and **guide** for the graphical model above.
- Specify appropriate **parameters** τ , α and β (e.g. try to use flat priors).
- Specify suitable **variational approximation** in the form of a Pyro guide.

$$q(\mu, \gamma) = \dots$$

- **Check** the differences with the notebook of Day 2.

Exercise 2: Explore Factor Analysis Model with Binomial likelihood

Day3/FA_binomial.ipynb

Variational Auto-Encoders

Limits on the scope of deep learning*

Deep learning thus far . . .

- . . . is data hungry
- . . . has no natural way to deal with hierarchical structure
- . . . is not sufficiently transparent
- . . . has not been well integrated with prior knowledge
- . . . works well as an approximation, but **its answers often cannot be fully trusted**

* Gary Marcus: *Deep Learning: A Critical Appraisal*. arXiv:1801.00631 [cs.AI]

Limits on the scope of deep learning*

Deep learning thus far . . .

- . . . is data hungry
- . . . has no natural way to deal with hierarchical structure
- . . . is not sufficiently transparent
- . . . has not been well integrated with prior knowledge
- . . . works well as an approximation, but **its answers often cannot be fully trusted**

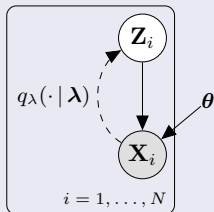
* Gary Marcus: *Deep Learning: A Critical Appraisal*. arXiv:1801.00631 [cs.AI]

Deep Bayesian Learning

A marriage of Bayesian thinking and deep learning is a framework that . . .

- . . . allows explicit modelling.
- . . . has a sound probabilistic foundation.
- . . . balances expert knowledge and information from data.
- . . . avoids restrictive assumptions about modelling families.
- . . . supports efficient inference.

Model of interest

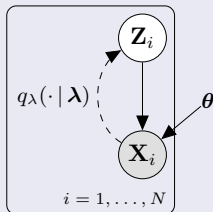


- $p_\theta(\mathbf{z}_i)$ usually is a isotropic Gaussian distribution.
- $p_\theta(\mathbf{x}_i | g_\theta(\mathbf{z}_i))$, where g is deep neural network (DNN).

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

- $g_\theta(\mathbf{z}_i)$ plays the role of a **DECODER NETWORK**.
- We want to learn θ to maximize the model's fit to the data-set $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.

Model of interest



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

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

- $g_\theta(\mathbf{z}_i)$ plays the role of a **DECODER NETWORK**.
- We want to learn θ to maximize the model's fit to the data-set $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$.

Variational Inference:

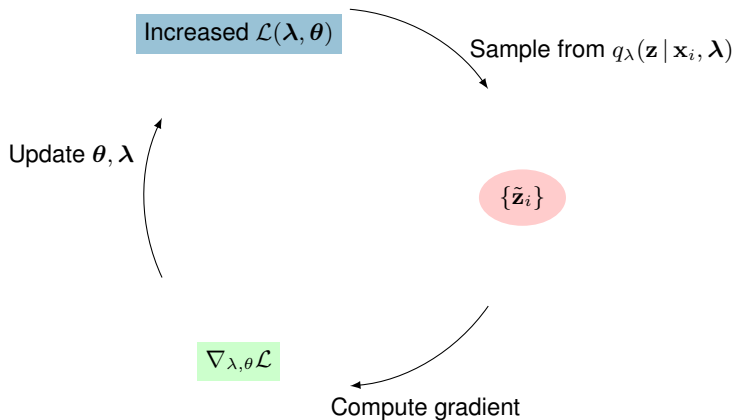
Optimize \mathcal{L} to choose λ and θ , where

$$\mathcal{L}(\lambda, \theta) = -\mathbb{E}_{q_\lambda} \left[\log \frac{q_\lambda(\mathbf{z} | \mathbf{x}, \lambda)}{p_\theta(\mathbf{z}, \mathbf{x} | \theta)} \right]$$

- The variational approximation $q_\lambda(\mathbf{z} | \mathbf{x}, \lambda)$ is parameterized by λ .

$$\mathbf{z}_i | \mathbf{x}_i \sim \mathcal{N}(\mu = h_\lambda(\mathbf{x}_i)[0], \Sigma = h_\lambda(\mathbf{x}_i)[1])$$

- $h_\lambda(\mathbf{x}_i)$ is a DNN which plays the role of a **ENCODER NETWORK**.



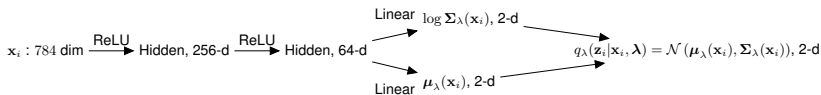
- The model is learned from $N = 55,000$ training examples.
- Each \mathbf{x}_i is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each \mathbf{x}_i is a picture of a handwritten digit (“0” – “9”)



- 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. $p(\mathbf{z}_i) = \mathcal{N}(\mathbf{0}_2, \mathbf{I}_2)$.
- The **encoder network** $\mathbf{X} \rightsquigarrow \mathbf{Z}$.

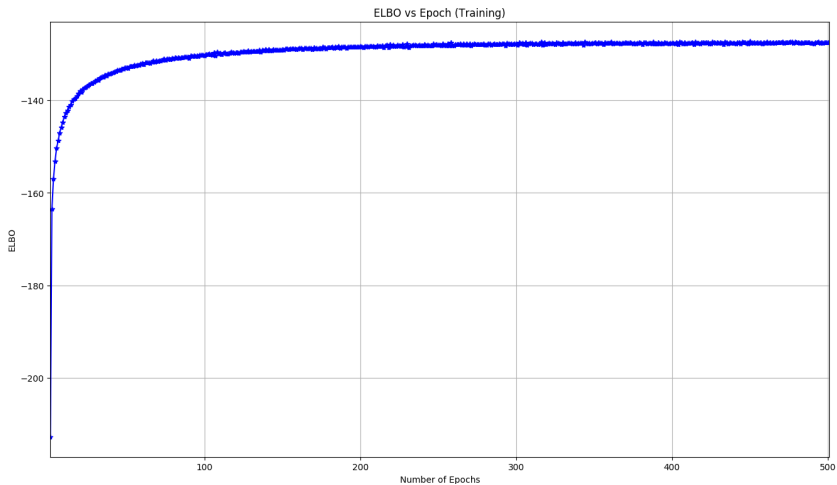


- The model is learned from $N = 55,000$ training examples.
- Each \mathbf{x}_i is a binary vector of 784 pixel values.
- When seen as a 28×28 array, each \mathbf{x}_i is a picture of a handwritten digit (“0” – “9”)



- 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.

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





After 1 epoch



After 250 epochs

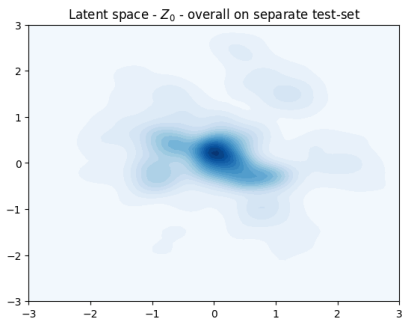
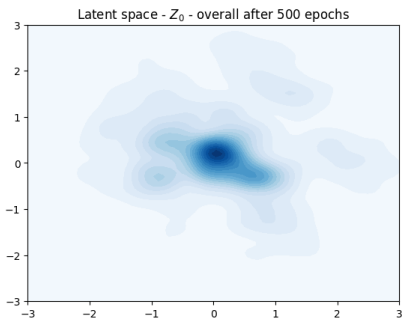
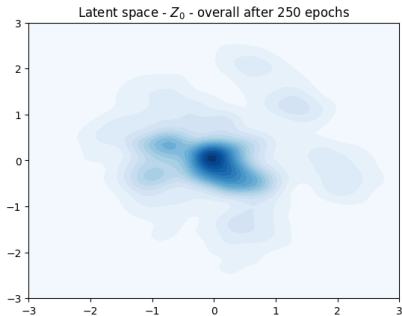
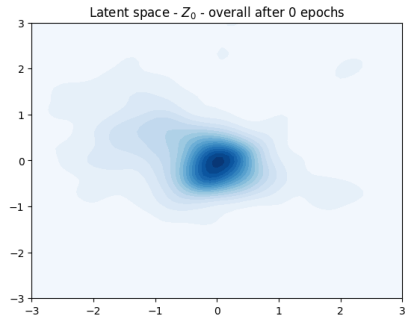


After 500 epoch

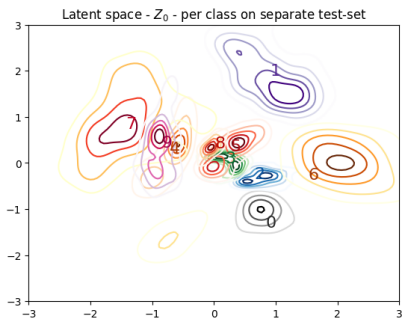
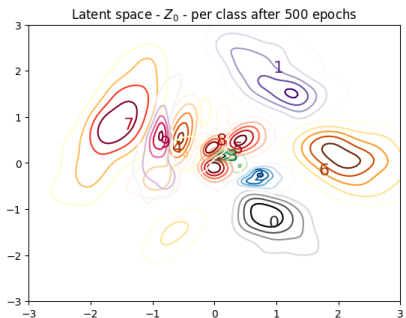
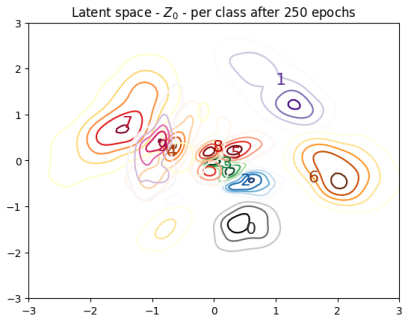
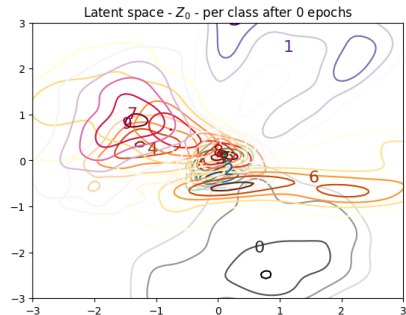


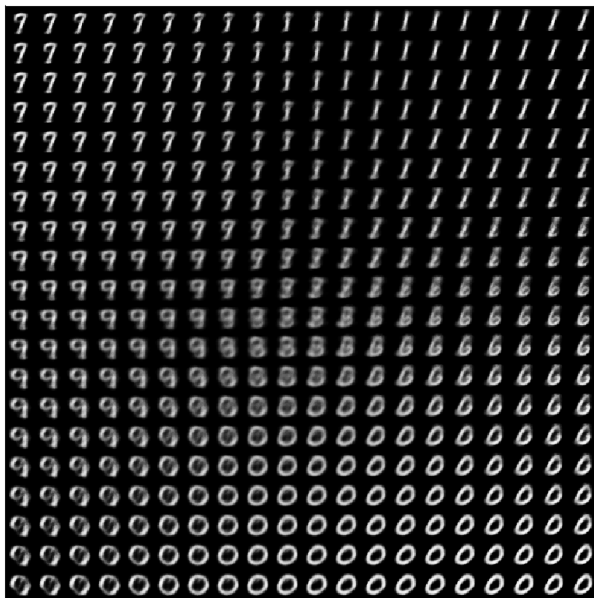
Using separate test-set

Averaged distribution over Z

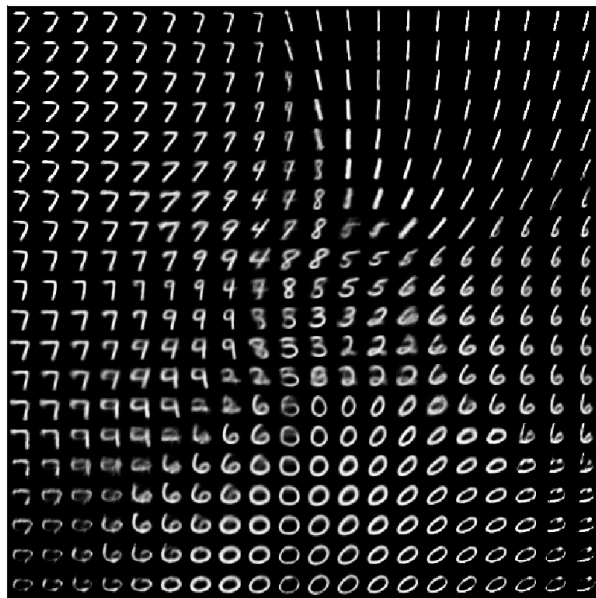


Averaged distribution over Z – per class

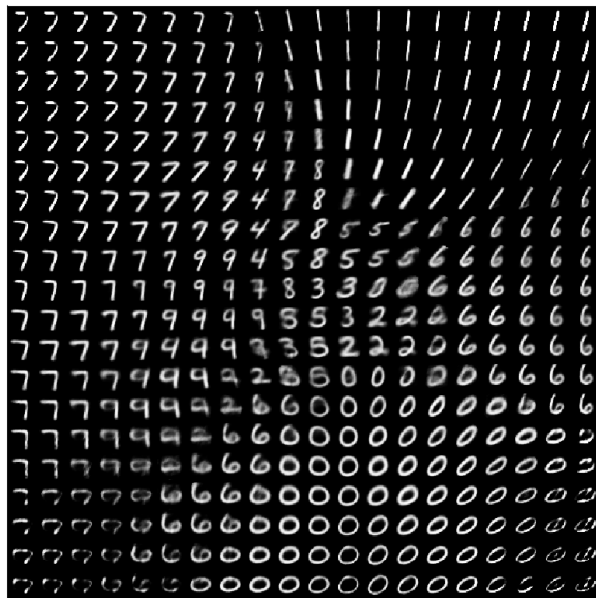




Manifold after 1 epoch



Manifold after 250 epochs



Manifold after 500 epochs

VAE.ipnyb

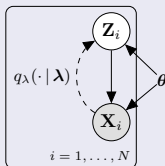

```
class Decoder(nn.Module):
    def __init__(self, z_dim, hidden_dim):
        super(Decoder, self).__init__()
        # Setup the two linear transformations used
        self.fc1 = nn.Linear(z_dim, hidden_dim)
        self.fc2l = nn.Linear(hidden_dim, 784)
        # Setup the non-linearities
        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.fc2l(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.



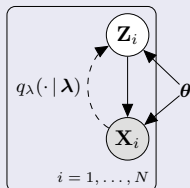
```
class Encoder(nn.Module):
    def __init__(self, z_dim, hidden_dim):
        super(Encoder, self).__init__()
        # Setup the three linear transformations used
        self.fc1 = nn.Linear(784, hidden_dim)
        self.fc21 = nn.Linear(hidden_dim, z_dim)
        self.fc22 = nn.Linear(hidden_dim,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



Conclusions

- **PPLs are the right tool for probabilistic modeling.**
 - Enormous expressibility.
 - Powerful inference engines (BlackBox Variational Inference)

- **PPLs are the right tool for probabilistic modeling.**

- Enormous expressibility.
- Powerful inference engines (BlackBox Variational Inference)

- **Conjugate Exponential Models.**

- Variational Inference is very efficient and stable.
- Requires manual derivation of updating equations.
- There are tools (variational message passing) that avoid that ([Infer.net](#), [Amidst Toolbox](#), etc).

- **PPLs are the right tool for probabilistic modeling.**

- Enormous expressibility.
- Powerful inference engines (BlackBox Variational Inference)

- **Conjugate Exponential Models.**

- Variational Inference is very efficient and stable.
- Requires manual derivation of updating equations.
- There are tools (variational message passing) that avoid that ([Infer.net](#), [Amidst Toolbox](#), etc).

- **Beyond Conjugate Exponential Models.**

- Combine deep learning and probabilistic modeling.
- Black-Box VI is not so efficient and stable.
- But it works well in many cases.