Deep Probabilistic Models

Part III: Graphical Models, Deep Latent Variables, and Variational Learning

Robert Salomone

AMSI Winter School 2021

Roadmap

- The second half of this course will focus on (Stochastic Gradient) **Variational Learning**.
  - It seems noone can distinguish between the terminology *Variational Approximations*, *Variational Bayes*, *Variational Inference*, and *Variational Learning*.
- However, I use the term **Variational Learning** as we will be training models with gradient descent (Learning) while simultaneously approximating some posterior distributions (Variational Bayes/Inference).
  - This is a **very general framework which encompasses many techniques**.
  - Know however that sometimes the terms are used in different ways. We will use the terminology as follows

$$\text{Variational Learning} \supset \text{Variational Inference/Bayes}.$$

- We need the general framework so we can make sense of **Variational Autoencoders**.
  - However, this way of viewing things in general can be incredibly useful, and will draw connections with many things you may or may not have seen before (e.g., EM algorithm)!

Everybody loves VAEs...

Auto-encoding variational bayes

DP Kingma, M Welling - arXiv preprint arXiv:1312.6114, 2013 - arxiv.org

How can we perform efficient inference and learning in directed probabilistic models, in the presence of continuous latent variables with intractable posterior distributions, and large datasets? We introduce a stochastic variational inference and learning algorithm that scales …

☆ 99 Cited by 15222 Related articles All 33 versions »

People also love Amortized Inference in Deep Latent Variable Models....

Stochastic backpropagation and approximate inference in deep generative models

DJ **Rezende**, S Mohamed… - … conference on machine …, 2014 - proceedings.mlr.press

We marry ideas from deep neural networks and approximate Bayesian inference to derive a generalised class of deep, directed generative models, endowed with a new algorithm for scalable inference and learning. Our algorithm introduces a recognition model to represent …

☆ 99 Cited by 3509 Related articles All 11 versions »

- This paper essentially proposed the exact **same ideas** in the same year!
- As the **Variational Autoencoder** is generally the Deep Latent Variable Model of interest, and only a special case of the framework presented in the paper above, fame has favored the former paper.

So, what is a VAE?

- Despite their popularity, people often have trouble really "**getting**" what is really happening with VAEs.
- Trying to explain them without having a lot of the required background is almost impossible.
- You end up with something that sounds like a bunch of terms mashed up into an abstract...

"*VAEs are a deep generative model, that resembles an autoencoder, but it is Bayesian. The posterior inference is intractable.....neural network overcomes the problem.....manifold learning.....latent space.....encode-decode independent features using variational approximations...disentanglement...second neural net.*"

- This is a shame, because the idea is, in my opinion, quite neat.
  - I'm going to take a very structured approach to build up so understanding VAEs (and extensions) is **easy**.

Roadmap

- To really **get** what is going on with VAEs and why they are called that requires an understanding of several concepts...
    - Directed Graphical Models and Latent Variable Models
    - Bayes Rule (in the context of Directed Graphical Models)
    - **Stochastic Backpropagation**
    - Variational Learning
    - Amortized Variational Inference
- **If we can understand all of the above, Variational Autoencoders are trivial to understand**.
- So, we will take things slowly, and build it up. Along the way, you will learn **several things that are very useful in their own right** (indeed, for some, you have already!).
- They will also help you understand probabalistic programming, which we will need to implement the VAE.
- The above is the focus of the second part of this course.

A Brief Mention: Undirected Graphical Models

- This course would not be complete without at least a very brief mention of **undirected** graphical models
- Examples include: **Auto-Logit Model** (Statistics), **Ising Spin-Glass Model** (Physics), and **Boltzmann Machine** (Machine Learning). Actually, those three things are the exact same model (**!**).
- Another example is the **Exponential Random Graph Model**.
- More generally, the above are examples of something called an **energy-based model**.
- In the context of performing Bayesian inference on $\boldsymbol{\theta}$, a lot of these models are what is called **doubly intractable**, as the normalizing constant of the distribution is unknown.
- Such models are very interesting, but moving forward we are going to restrict ourselves to the directed graphical (acyclic) graphical model framework.
    - The latter is the dominant area of interest in probabilistic machine learning (at least for now!).

Bayes beyond θ.

Bayes Rule: Not just for parameters!

- We will shortly see that being "**Bayesian**" about one's **parameters** has simply one example of **introducing an additional latent variable in a directed graphical model**.

Everything is a graphical model!

- At this winter school, you have been introduced to **Bayesian Statistics**.

$$p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta})$$

- The above is often referred to a **the** posterior, but I want you to think of it as only a posterior.
- Posterior inference in Bayesian statistics is just a special case of posterior inference in a **directed graphical model**.
- A directed graphical model (sometimes called a **Bayesian Network**) is a collection of random objects whose joint distribution is specified in a generative manner through a **directed acyclic graph**.

Directed Graphical Models

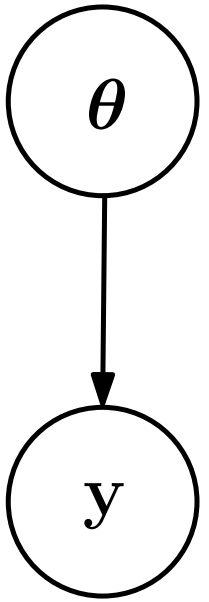
- Here is a graphical model of the prior predictive distribution in the Bayesian statistics setting:

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$$

$$\boldsymbol{Y}|\boldsymbol{\theta} \sim p(\boldsymbol{y}|\boldsymbol{\theta})$$

- This is a model of our belief before we observe anything.
- The graph on the right tells us how to simulate from the distribution, or rather, the way we have specified our joint distribution...

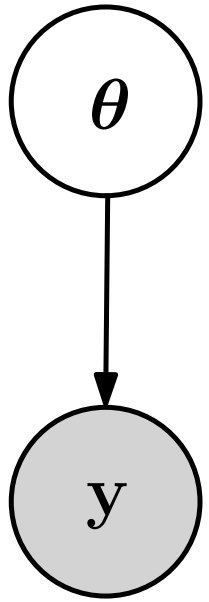
$$p(\boldsymbol{\theta}, \boldsymbol{y}) = p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta})$$



Conditioning

- We represent that we have **observed** the node $\boldsymbol{y}$ by colouring it in.
- Thus, the model on the right represents the distribution of $\boldsymbol{\theta}$ having already observed $\boldsymbol{y}$.
- Note that in all this, $\boldsymbol{\theta}$ is a **latent** variable in our model, **whose purpose is to model of our own uncertainty** regarding the parameter of the data generating process.
- Once we have observed a variable (or variables), we have the posterior distribution over the **latent** (or hidden) variable, via Bayes rule.

$$p(\boldsymbol{\theta}|\boldsymbol{y}) \propto p(\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{\theta})$$



More Generally...

- More generally, we can have a graphical model with variables that are **observed**, as well as **latent** (also called *unobserved*, or *hidden*).
- Henceforth, we will denote our latent variables as $\boldsymbol{Z}$.
    - Note the connection with how we have used the notation $\boldsymbol{Z}$ thus far, that is not entirely coincidental.
- Via a simple application of Bayes' Rule, we may write

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})}{p(\boldsymbol{x})}$$

where $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})\mathrm{d}\boldsymbol{z}$ is the (marginal) **likelihood** of $\boldsymbol{x}$.

- Go from Bayesian Statistics to Bayesian Machine Learning in one easy step! Write $\boldsymbol{Z}$ instead of $\boldsymbol{\theta}$!
    - (That was a joke, but contains a grain of truth!)

Why have latent variables?

Mixed Models: Latent Variables for Modelling Dependence

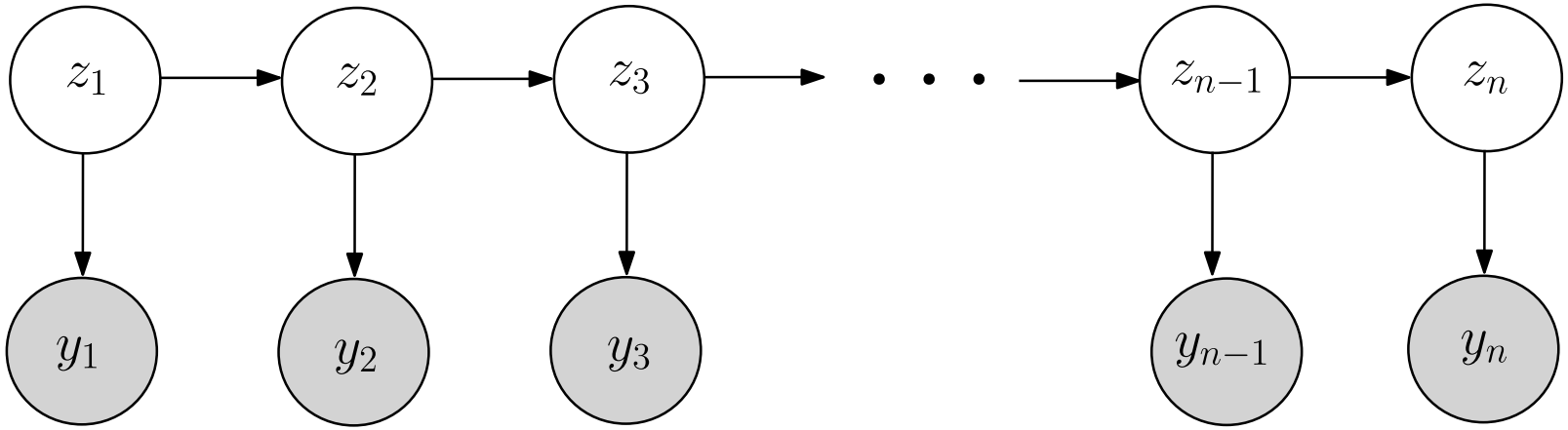
- An example of a model with latent variables you may have seen before is is a **Linear Mixed Model**. It is a simple, yet powerful model of **dependent** data.

$$Y = \mathrm{X}\boldsymbol{\beta} + \mathrm{V}\boldsymbol{Z} + \boldsymbol{\epsilon}.$$

- Above, $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \Sigma_{\boldsymbol{Z}})$ is a latent vector, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathrm{I})$, X is the design matrix (includes the data and possibly a column of ones).
- Then, V is a **second** design matrix that includes information about how the data are related (which observations share certain elements of $\boldsymbol{Z}$ to model their dependent.
- **Classic Example**: Panel (Longitudinal) Data. Each individual has their own latent variable.
- Alternatively it may be modelling at the group/item levels.

State Space Model: latent variables modelling physical phenomena

- Consider the famous (univariate) **state space model**.



In **generative form**:

$$Z_1 \sim p_{Z_1}$$

$$Z_t | Z_{t-1} \sim p_{Z_t|Z_{t-1}}, \quad t = 2, \ldots, n.$$

$$Y_t | Z_t \sim p_{Y_t|X_t}, \quad t = 1, \ldots, n.$$

- Model of some unobserved **latent** process, for which we have noisy **observations**.

$$p(\boldsymbol{z}, \boldsymbol{y}) = \prod_{k=1}^{n} p(z_k|z_{k-1}) p(y_k|z_k)$$

- Here, we may care about $p(\boldsymbol{z}|\boldsymbol{y})$ - for example in **tracking** problems.

Why did I just show you that?

- I showed you that particular example as it is example of a graphical model where we want to be **Bayesian** about a latent variable that isn't $\boldsymbol{\theta}$, but has an important role in the underlying model!
- In Bayesian statistics, $\boldsymbol{\theta}$ is simply a latent (unobserved) variable which is treated with particular importance. That is, the end goal is obtaining $p(\boldsymbol{\theta}|\boldsymbol{y})$.
- More generally, we may have (other) latent variables in our graphical model.
    - Sometimes, we may not even care about $\boldsymbol{\theta}$ so much as achieving other things (e.g., inferring something about latents).
- **I have neglected Discrete variables in the very flexible models I have been showing you in this course thus far, so we will focus on them for a bit!**

A Simple Latent Variable Model

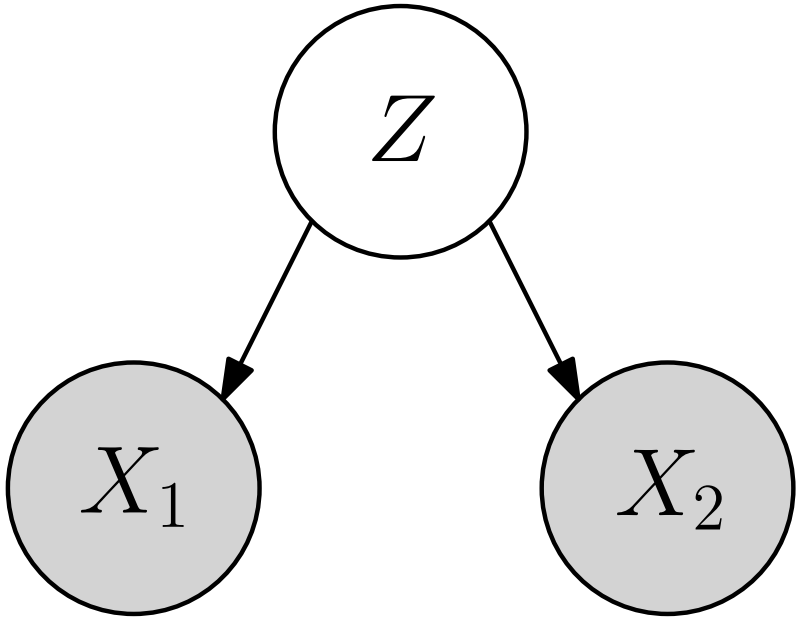
- Consider the following generative process:

$$Z \sim \mathcal{N}(0, 1)$$

$$X_1 | Z \sim \text{Bernoulli}(\textsf{sigmoid}(a_1 + b_1 Z))$$

$$X_2 | Z \sim \text{Bernoulli}(\textsf{sigmoid}(a_2 + b_2 Z))$$

- Here, we have used a latent variable as a **flexible** modelling tool to create a **dependent** Bernoulli model.



Being Lazy: Plate Notation

- **Plate notation** is commonly used in probabalistic machine learning (and statistics) to ease representing the graphs by putting repeated variables in a "plate":
- We just need to add one more

1. Arrows outline the order in which variables are generated (how the distribution factorizes).
2. Random Variables are denoted inside circles
3. Fixed stuff don't have circles.
4. Colored-in variables represent those that are **observed**, ones with white backgrounds correspond to **latent** variables.
5. Anything inside a box is conditionally independent of everything outside it.

(Extended Tutorial Video on Plate Notation)

Example: Deep Bernoulli Model

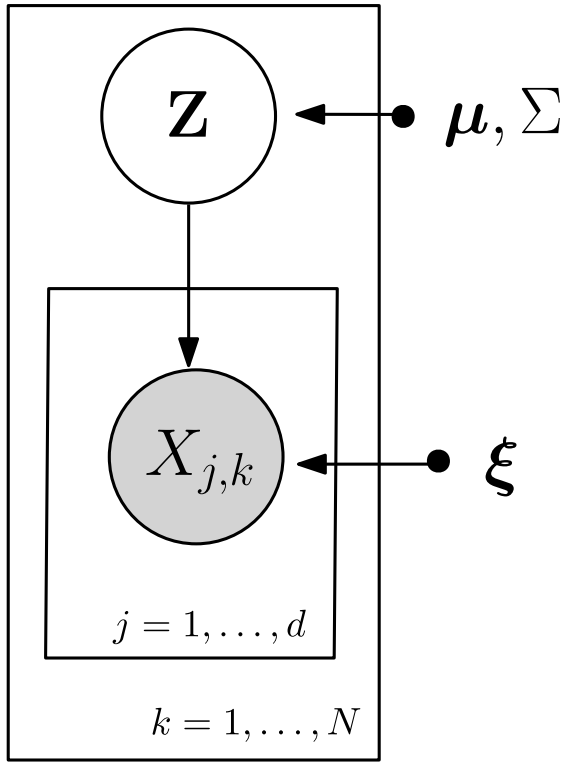
- We can make our model deeper by using multiple layers of latent variables...
  - Our latent variable $\boldsymbol{Z}$ will be of dimension m.
  - Then, for **one** sample, we generate

$$\boldsymbol{Z} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

$$\boldsymbol{p} = g_{\boldsymbol{\xi}}(\boldsymbol{Z}), \quad g_{\xi} : \mathbb{R}^m \to (0,1)^d$$

$$X_i \,|\, \boldsymbol{p} \sim_{\text{ind}} \text{Bernoulli}(p_i), \quad i = 1, \ldots, d.$$

  - Above, $g_{\boldsymbol{\xi}}$ may come from any $\boldsymbol{\xi}$-parametrized class of functions (neural net!).




WE NEED TO GO
DEEPER

Doubly Deep Bernoulli Model: State of the art!

- Again, let $g_{\boldsymbol{\xi}}$ be a function (neural network!) with parameters $\boldsymbol{\xi}$.
- If we would like additional flexibility, we can add another "layer" of latent variables.
- Suppose we have N observations from the following model...

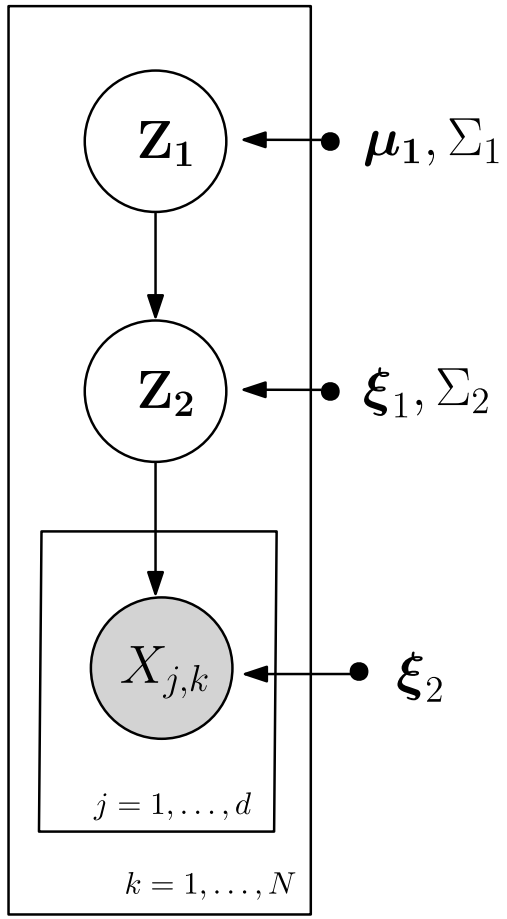
$$\boldsymbol{Z}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$$

$$\boldsymbol{\mu}_2(\boldsymbol{Z}_1) = g_{\boldsymbol{\xi}_1}(\boldsymbol{Z}_1), \quad g_{\xi_1} : \mathbb{R}^{\dim(\boldsymbol{Z}_1)} \to \mathbb{R}^{\dim(\boldsymbol{Z}_2)}$$

$$\boldsymbol{Z}_2 \sim \mathcal{N}(\boldsymbol{\mu}_2(\boldsymbol{Z}_1), \Sigma_2)$$

$$\boldsymbol{p} = g_{\boldsymbol{\xi}_2}(\boldsymbol{Z}_2), \quad g_{\xi_2} : \mathbb{R}^{\dim(\boldsymbol{Z}_2)} \to (0,1)^d$$

$$X_i \,|\, \boldsymbol{p} \sim_{\text{ind}} \text{Bernoulli}(p_i), \quad i = 1, \ldots, d.$$



Deep Probabalistic Modelling

- Methods such as the last few models are the essence of **deep probabalistic modelling**.
  - We can be deep with our use of neural networks (themselves deep).
  - The latent variables can be deep.
- All this deepness should serve a purpose: e.g., flexible models.
- Training the models and inference is then difficult, but we will learn to deal with that.

Takeaways

- Being "**Bayesian**" about $\boldsymbol{\theta}$ is really just adding a latent variable to a graphical model that is put there to model your **epistemic uncertainty**.
- Of course, this can be **very important** to get inferences about $\boldsymbol{\theta}$ for scientific purposes.
    - Including $\boldsymbol{\theta}$ as a latent in a model and specifying a prior also acts as **regularization**.
- More generally, we may not care about $\boldsymbol{\theta}$ so much, so we can be **Bayesian** but not for $\boldsymbol{\theta}$.
    - Example: If we have $\boldsymbol{z} \in \mathbb{R}^{50}$ say but $\boldsymbol{\theta}$ parametrizes for example a deep generative model, it introduces many headaches to deal with the high dimensional posteriors over $\boldsymbol{\theta}$ which we may not be that interested in.
- Latent Variables:
    - Sometimes the latents model data structure (think Mixed Models)
    - Sometimes the posteriors $\boldsymbol{z}|\boldsymbol{x}$ have some desirable interpretation (think State Space Models) and so posterior inference is important.
    - Sometimes the latents are a convenient way to make your model more flexible (think the Doubly Deep Bernoulli model).

Likelihood

- Note that we are not being Bayesian about $\boldsymbol{\theta}$ (all those parameters, e.g., $\boldsymbol{\xi}$, Σ, etc.). If we were, we could just MCMC over $(\boldsymbol{\theta}, \boldsymbol{Z})$.
- Assume we wish to train via maximum likelihood. As we do not observe $\boldsymbol{Z}$, we are interested in the **marginal** likelihood for this model is.

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p_{\boldsymbol{\theta}}(\boldsymbol{z}) \mathrm{d}z$$

- We could try to use stochastic backpropagation, but in general that can't be trusted to work well. Also, that would not give us any information about $\boldsymbol{Z}|\boldsymbol{X}$ (which, generally, we may care about).
- That seems like a $+1$ for "Fully" Bayes!
- However, in general, if the latent variables are low-dimensional and the parameters are high dimensional (as they will be very soon), you want to avoid having **lots** more latent variables that may make inference hard over certain ones we care about.
- So, we may need to be semi-Bayesian going forward. To do so, I will now introduce a general framework.

Maximum (Marginal) Likelihood with Latent Variables: The General Framework

- In light of the previous discussion, and **with apologies to the Reverend Thomas Bayes**, we shall assume the following:
  - Any parameters that one wishes to treat in a Bayesian matter are simply additional latent variables contained in $\boldsymbol{Z}$ that shall hereafter be afforded no privileged position.
- There will also be parameters that we wish to optimize via maximum likelihood, **those** parameters are what we henceforth refer to as $\boldsymbol{\theta}$.
- Let $p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})$ be the joint distribution over the observed variables $\boldsymbol{x}$ and the latent variables $\boldsymbol{z}$ with parameter $\boldsymbol{\theta}$. We seek

$$\boldsymbol{\theta}^{\star} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \left\{ \log p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \right\}.$$

- From this, we will be interested in posteriors of the form $p(\boldsymbol{z}|\boldsymbol{x}\,;\,\boldsymbol{\theta}^{\star})$.

Example

- The previous framework is very general, and it actually includes an interesting class of techniques called **empirical Bayes**.
    - Empirical Bayes can be considered as being "semi-Bayesian": you have a posterior over your parameters $\boldsymbol{\theta}$ but you choose some of your **prior** hyperparameters $\boldsymbol{\eta}$ via optimizing the marginal likelihood.
    - The above sounds a bit dodgy (heretical), but is used all the time in Bayesian model selection.
    - It is also a standard approach for choosing kernel bandwith in **Gaussian Processes**.
    - In some cases, the resulting estimators can have very good theoretical frequentist properties.
- Our framework is more general however, we can optimize over any parameters we see fit, not just "prior" parameters.
    - Sometimes you want to be Bayesian about latent variables in a model, but not necessarily about the parameter $\boldsymbol{\theta}$.

How Bayesian are you?

Empirical Bayes violates the principle that the prior should be chosen independently of the data. However, we can just view it as a computationally cheap approximation to inference in a hierarchical Bayesian model, just as we viewed MAP estimation as an approximation to inference in the one level model $\boldsymbol{\theta} \rightarrow \mathcal{D}$. In fact, we can construct a hierarchy in which the more integrals one performs, the "more Bayesian" one becomes:

Method	Definition
Maximum likelihood	$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\mathcal{D}\|\boldsymbol{\theta})$
MAP estimation	$\hat{\boldsymbol{\theta}} = \operatorname{argmax}_{\boldsymbol{\theta}} p(\mathcal{D}\|\boldsymbol{\theta}) p(\boldsymbol{\theta}\|\boldsymbol{\eta})$
ML-II (Empirical Bayes)	$\hat{\boldsymbol{\eta}} = \operatorname{argmax}_{\boldsymbol{\eta}} \int p(\mathcal{D}\|\boldsymbol{\theta}) p(\boldsymbol{\theta}\|\boldsymbol{\eta}) d\boldsymbol{\theta} = \operatorname{argmax}_{\boldsymbol{\eta}} p(\mathcal{D}\|\boldsymbol{\eta})$
MAP-II	$\hat{\boldsymbol{\eta}} = \operatorname{argmax}_{\boldsymbol{\eta}} \int p(\mathcal{D}\|\boldsymbol{\theta}) p(\boldsymbol{\theta}\|\boldsymbol{\eta}) p(\boldsymbol{\eta}) d\boldsymbol{\theta} = \operatorname{argmax}_{\boldsymbol{\eta}} p(\mathcal{D}\|\boldsymbol{\eta}) p(\boldsymbol{\eta})$
Full Bayes	$p(\boldsymbol{\theta}, \boldsymbol{\eta}\|\mathcal{D}) \propto p(\mathcal{D}\|\boldsymbol{\theta}) p(\boldsymbol{\theta}\|\boldsymbol{\eta}) p(\boldsymbol{\eta})$

Excerpt from **Machine Learning: A Probabalistic Perspective** by Kevin Murphy (2012)

Variational Learning

- In a nutshell, sampling is hard and expensive in high dimensions, so we will replace it by **optimization**.
- Instead of sampling posterior distributions, we just optimize to get some member of an approximating family that is "close" in some sense so we can optimize our parameters while dealing with those pesky latents!

Auxilliary Distribution

- Key trick, introduce some auxilliary family of distributions $q_{\boldsymbol{\phi}}(\boldsymbol{z})$ with *variational parameter* vector $\boldsymbol{\phi}$ that we will use to approximate $p(\boldsymbol{z}|\boldsymbol{x})$. To remind ourself of this fact, we shall use the notation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$.
- We will write,

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z}|\boldsymbol{x})]$$

- Then, for any q, it can be shown that $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) \leq p_{\boldsymbol{\theta}}(\boldsymbol{x})$, with equality is achieved iff $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \equiv p_{\theta}(\boldsymbol{z}|\boldsymbol{x})$.
- The "closer" $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$ is to $\boldsymbol{p}(\boldsymbol{z}|\boldsymbol{x})$ in (reverse) KL divergence, the **tighter** the bound will be.
- Thus, we use $\mathcal{L}$ as our loss function, and if $q_{\phi}(\boldsymbol{z}|\boldsymbol{x})$ approximates $p_{\boldsymbol{\theta}}(\boldsymbol{z}\,|\boldsymbol{x})$ well, we will approximately maximize the likelihood.
- There is actually a connection to the EM algorithm here, see the excellent **slides by Zoubin Gharamani** for further discussion.

The Reduced Case: $\boldsymbol{\theta} = \{\}$, i.e., "Fully Bayesian"

- If one does not choose to optimize over *any* parameters ϕ and have everything in the model be latent --- i.e., we are being "**fully Bayesian**" over all parameter:

$$\begin{aligned}\arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z})] &= \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p(\boldsymbol{x}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z})] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}} \frac{\log p(\boldsymbol{x}, \boldsymbol{Z})}{\log q_{\boldsymbol{\phi}}(\boldsymbol{Z})} \\ &= \arg\min_{\boldsymbol{\phi}} \mathrm{KL}(q_{\boldsymbol{\phi}} || p)\end{aligned}$$

- Thus, in the reduced case, minimizing the negative ELBO is equivalent to minimizing $\mathrm{KL}(q_\phi || p)$.

Backward KL: How good is it?

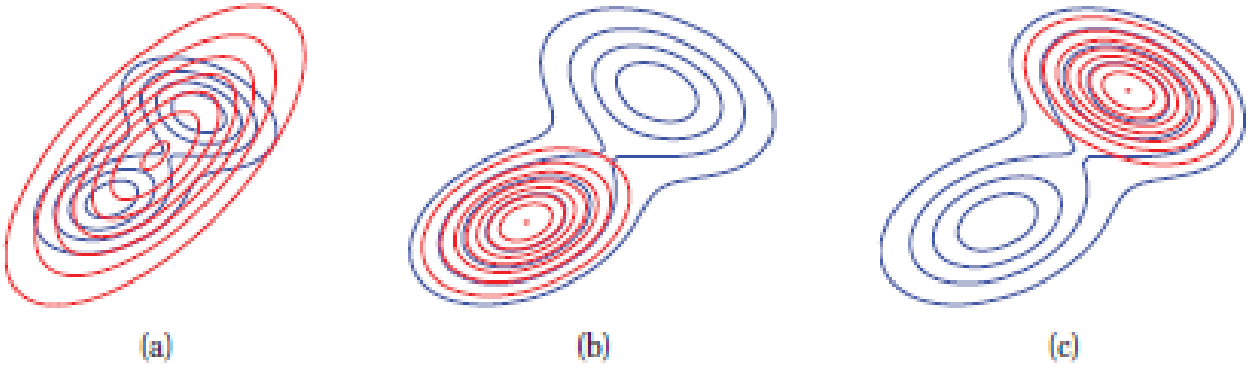


Figure 21.1 Illustrating forwards vs reverse KL on a bimodal distribution. The blue curves are the contours of the true distribution p. The red curves are the contours of the unimodal approximation q. (a) Minimizing forwards KL: q tends to "cover" p. (b-c) Minimizing reverse KL: q locks on to one of the two modes. Based on Figure 10.3 of (Bishop 2006b). Figure generated by `KLfwdReverseMixGauss`.

Figure also from Kevin Murphy's very excellent book (new version available free!)

Stochastic Backpropagation, the Return

- So, we are interested in solving $\boldsymbol{\theta}^\star, \boldsymbol{\phi}^\star$ which is the solution the optimization program

$$\arg\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z})]$$

- The solution to all of lifes problems: **Gradient Descent** ...

$$\nabla_{\boldsymbol{\theta},\boldsymbol{\phi}} \mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{Z}) - \log q_{\boldsymbol{\phi}}(\boldsymbol{Z})]$$

- As we saw in the lecture on GANs, we have the technology we need to estimate the above gradients with **low-variance**.

- It is in our interests to use a family for $q_{\boldsymbol{\phi}}$ that is **reparametrizable** for variance reasons.

Choosing a Family for q_η

- **Desiderata**
  - Flexibility
  - Reparametrizability
  - Ease of Optimization
  - Inductive Bias
  - Parsimony
- Remember, our variational distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ is approximating the true posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$.

Gaussian Variational Families and their Reparametrized Form

- **Gaussian with Diagonal Covariance**

$$q(\cdot \mid \boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \operatorname{diag}(\boldsymbol{\sigma}))$$

$$\boldsymbol{Z} = t_{\phi}(\boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathrm{I}_d).$$

- **Gaussian with General Covariance**: For lower triangular L with positive diagonal,
  - Even a Gaussian in this dimension will have $\mathcal{O}(d^2)$ parameters.

$$q(\cdot \mid \boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma), \quad \Sigma = \mathrm{L}\mathrm{L}^{\top},$$

$$\boldsymbol{Z} = t_{\phi}(\boldsymbol{\epsilon}) = \boldsymbol{\mu} + \mathrm{L}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathrm{I}_d).$$

- **Gaussian with a Factor Covariance Structure** (Ong et al., 2017): For $\mathrm{B} \in \mathbb{R}^{d \times r}$,

$$q(\cdot \mid \boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma), \quad \Sigma = \operatorname{diag}(\boldsymbol{d}) + \mathrm{B}\mathrm{B}^{\top},$$

$$\boldsymbol{Z} = t_{\boldsymbol{\phi}} = \boldsymbol{\mu} + \mathrm{B}\boldsymbol{\epsilon}_1 + \boldsymbol{d} \odot \boldsymbol{\epsilon}_2, \quad \boldsymbol{\epsilon}_1 \sim \mathcal{N}(\mathbf{0}, \mathrm{I}_r), \quad \boldsymbol{\epsilon}_2 \sim \mathcal{N}(\mathbf{0}, \mathrm{I}_d).$$

Flow Variational Families

- In principle, you can use any reparametrizable distribution as a variational family.
- Naturally, Normalizing Flows fall into that category.

$$\boldsymbol{X} = T(\boldsymbol{Z}\,;\,\boldsymbol{\phi})$$

- Flows are most useful for **amortized inference** (more on that tomorrow).

Structured Variational Families

- Depending on the graphical structure of the model, it may make sense to create .
- This is a bit outside the scope of this course, but one nice approach is

Gaussian Variational Families with Sparse Precision Matrices (Tan & Nott, 2017)

$$\boldsymbol{Z} = t_\phi(\boldsymbol{\epsilon}) = \boldsymbol{\mu} + \mathrm{L}^{-1}\boldsymbol{\epsilon}.$$

where L has a certain sparsity structure matching the conditional independence structure of the target.

- There are some recent approaches that take into account graphical structure automatically using probabalistic programming languages:
    - Automatic Structured Variational Inference (Ambrogioni et al., 2021)

"*the variational program has the same control-flow structure, dependence graph, and time complexity as the original model. Local transformations of this kind may be implemented by effect handlers (Plotkin and Pretnar, 2009), a mechanism supported in multiple recent probabilistic programming frameworks*" - ASVI Paper

Sticking the Landing

- In practice, always want to use a reparametrization gradient in the stochastic backpropagation.
- In fact, we can get the variance even lower!
- Interesting trick discovered by Tan & Nott (2018), generalized by Roeder et al. (2017).
- We have that $\mathbb{E}_q\left[\nabla_{\boldsymbol{\phi}} \log q_{\phi}(\boldsymbol{Z})\right] = \mathbf{0}$ (note the connection to ZV-CV), and so we can **remove** the corresponding term from the estimator of

$$\begin{aligned}\nabla_{\boldsymbol{\phi}} &= \nabla_{\boldsymbol{\phi}} \mathbb{E}[\log p(\boldsymbol{x}, \boldsymbol{Z}) - \log q_{\phi}(\boldsymbol{Z})] \\ &= \mathbb{E}_{q_{\boldsymbol{\phi}}}\left[\left(\nabla_{\boldsymbol{z}} \log p(\boldsymbol{Z}, x) - \nabla_{\boldsymbol{z}} \log q_{\boldsymbol{\phi}}(\boldsymbol{Z})\right) \mathrm{J}_{t(\boldsymbol{\epsilon}, \boldsymbol{\phi})}\right] - \mathbb{E}_{q_{\boldsymbol{\phi}}} \nabla_{\phi} \log q_{\boldsymbol{\phi}}(\boldsymbol{z}) \,.\end{aligned}$$

- If $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}) = p(\boldsymbol{z}|\boldsymbol{x})$, an estimator of the **path derivative** (term inside leftmost expectation) will have **zero variance**.
- They call this **sticking the landing** as the stochastic gradient will have lower variance when the optimizer is "landing" near an optima.
- This is tricky to do with automatic differentiation, but **Pyro** implements this all for us in the background.

Sticking the Landing...

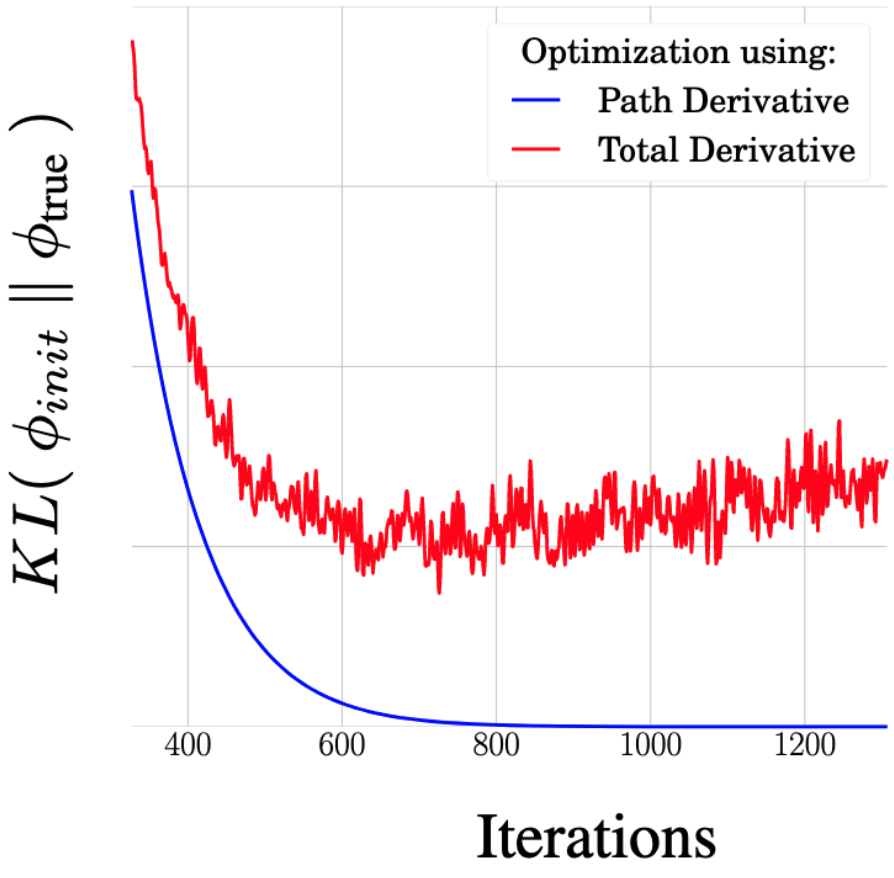


Figure from Roeder et al., (2017), *Sticking the Landing: Simple, Lower-Variance Gradient Estimators for Variational Inference*.

An Implementation

- Let's have a quick look at an implementation of fully Bayesian VI using a Gaussian with Factor Covariance Structure...

```
class VAFC:
    def __init__(self, p, r_cov = 1, logp = [], gr_logp = [], eps=1e-8):
        self.r_cov, self.p =  r_cov, p
        self.logp, self.gr_logp = logp, gr_logp
        self.eps = eps

        self.prm = {'mu':t.zeros(p, requires_grad=True, dtype = t.float64),
                    'B': t.zeros((p,r_cov), requires_grad=True, dtype = t.float64),
                    'sig_th': t.tensor(0.25*t.ones(p, dtype = t.float64),
                                       requires_grad=True)
                    }
```

```
  def t(self, s1,s2):
        # reparametrized form
        Z = (self.prm['mu']  + self.prm['B'] @ s1 + self.prm['sig_th'] * s2)
        return Z
```

Implementation Continued

```
    def sample(self, est_ELBO=False):
        s1 = torch.randn(self.r_cov, dtype = t.float64)
        s2 = torch.randn(self.p, dtype = t.float64)

        theta = self.t(s1,s2)

        if est_ELBO:
            self.lastELBO = self.ELBO(Z)

        return Z
```

```
    def ELBO(self, Z):
        with torch.no_grad():
            DIST = dst.LowRankMultivariateNormal(loc = self.prm["mu"],
                                                 cov_factor = self.prm['B'],
                                                 cov_diag = self.prm["sig_th"]**2)

            ELBO = self.logp(Z) - DIST.log_prob(Z)

            return ELBO
```

```
    def train_step(self, n_BZ = 100, save = True):
        Z = self.sample(est_ELBO=True)

        gr_logp = t.tensor(self.gr_logp(Z).reshape(-1,1))

        with t.no_grad():
            # self.pr_Z computes the precision matrix efficiently
            gr_entropy = -self.pr_Z() @ (Z - self.prm['mu']).reshape(-1,1)

        r = gr_logp - gr_entropy

        Z.backward(-r.flatten()) # most efficient way

        self.optim.step()
```

```
    def full_train(self, num_steps, show = int(500), N=1):
        self.optim = optim.Adadelta(self.prm.values(), rho=0.8, eps=self.eps)
        mn_ELBO = 0

        for i in range(1,num_steps+1):
            self.optim.zero_grad()
            self.train_step()
            mn_ELBO += self.lastELBO
            if i%show == 0 and i>0:
                print(i, float(mn_ELBO/show))
                mn_ELBO = 0
```

Alternatives to the ELBO Loss

- Generally, such methods are referred to as "**Variational Objectives**", and in the general setting aim to make the ELBO bound "**tighter**".
  - Importance Weighted Bound
  - Alternate (and tighter) lower bound on the likelihood

$$\mathcal{L}_k(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots \boldsymbol{Z}_k \sim q(\boldsymbol{z}|\boldsymbol{x})} \left[\log \left(\frac{1}{k} \sum_{i=1}^{k} \frac{p(\boldsymbol{Z}_i, \boldsymbol{x})}{q(\boldsymbol{Z}_i|\boldsymbol{x})} \right) \right]$$

  - Recovers **ELBO loss** for $k = 1$.
    - Reduces to Renyi-Divergence in the "fully-Bayesian" case.
    - **State-of-the-Art** is to employ Doubly Reparametrized Gradient estimators (Tucker et al., 2018) which are an unbiased analogue of the "Sticking the Landing" gradient estimators for alternate objectives.
- Jacknife Variational Objective: Introduces further debiasing via jacknife arguments.
- Thermodynamic Variational Objective: Links VI and Thermodynamic Integration identities to obtain tighter bounds.

An Aside: Mean Field Variational Inference

- Beyond this slide, we will **not** focus on Mean Field Variational Inference, but it is nice to know about it as you may come across it in the literature.
- Here, the objective is to obtain $\arg\min_{q \in \mathcal{Q}} \mathrm{KL}(q||p)$ where the admissible family $\mathcal{Q}$ is all distributions that factorize as

$$q(\boldsymbol{x}) = \prod_k q_k(\boldsymbol{x}_k)$$

i.e., individual elements (or blocks of elements) are independent.

- Advantages: often *extremely fast*, it is neat that you do **not** need to specify a family for q, no gradients needed, posesses a certain mathematical elegance.
- Disadvantages: limited to certain models, not a very flexible family.
  - Requires extremely long and often tedious derivations of the update rules, which are different for every model.
- By contrast, stochastic gradient methods are now dominating the literature, as they are more black-box and generally work well so long as one uses **reparametrization gradients**.

Up Next

- So, that was the crash course on **Directed Graphical Models** and **Variational Learning**.
- As you may have noticed, implementing these things are not simple, and usually code needs to be changed model to model.
  - We will take a look at `Pyro` which makes both specifying models and variational inference a lot easier.
  - We will learn what a variational autoencoder is.
  - We will learn about amortized inference.