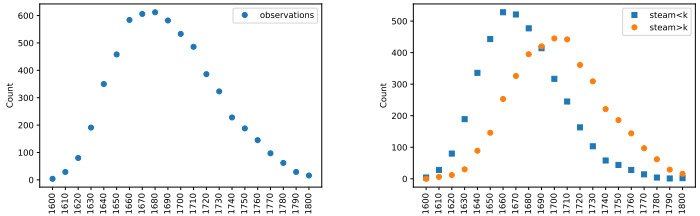
Deep Discrete Latent Variable Models

Wilker Aziz
ILLC @ UvA

Outline

Latent structure

Here I am plotting publication dates (from last class).
Left: observations for Y. Right: observations for $Y|[\text{freq}_{\text{steam}}(x) > k]$.



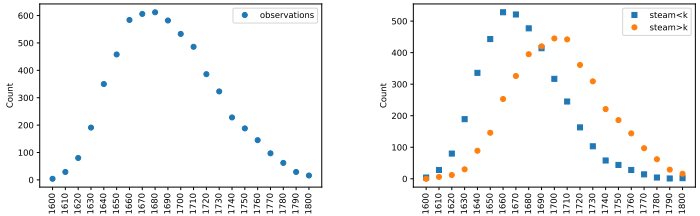
What do we learn from this observation?

Rightmost plot: Suppose we separate our observations (labelled documents) in terms of how often a document contains the word *steam*. Our historians claim that above some threshold k a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society). Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Visualisations: It might strike you as a surprise, but visualising data is very much like modelling data, and therefore is not void of *inductive bias*. We have to be very conscious about what it means to visualise data in a certain way. For example, by choosing to plot curves according to the increased frequency of a certain word, we might have made that aspect artificially important for the phenomenon under consideration (e.g., many other unknown factors might have contributed to the distribution of that word's frequency over the decades in consideration).

Latent structure

Here I am plotting publication dates (from last class).
Left: observations for Y. Right: observations for $Y|[\text{freq}_{\text{steam}}(x) > k]$.



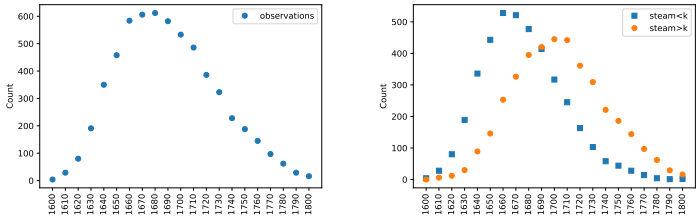
Marginally (left), it looked like our observations could have been drawn from a Poisson distribution.

Rightmost plot: Suppose we separate our observations (labelled documents) in terms of how often a document contains the word *steam*. Our historians claim that above some threshold k a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society). Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Visualisations: It might strike you as a surprise, but visualising data is very much like modelling data, and therefore is not void of *inductive bias*. We have to be very conscious about what it means to visualise data in a certain way. For example, by choosing to plot curves according to the increased frequency of a certain word, we might have made that aspect artificially important for the phenomenon under consideration (e.g., many other unknown factors might have contributed to the distribution of that word's frequency over the decades in consideration).

Latent structure

Here I am plotting publication dates (from last class).
Left: observations for Y. Right: observations for $Y|[\text{freq}_{\text{steam}}(x) > k]$.



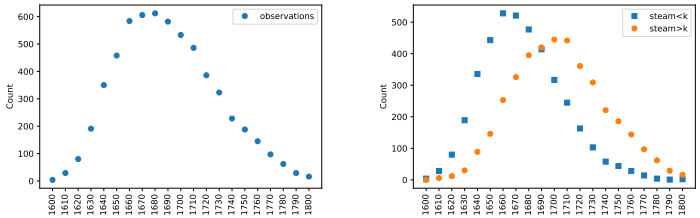
But they might also have been the result of mixing (right) into one population draws from two different Poisson distributions.

Rightmost plot: Suppose we separate our observations (labelled documents) in terms of how often a document contains the word *steam*. Our historians claim that above some threshold k a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society). Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Visualisations: It might strike you as a surprise, but visualising data is very much like modelling data, and therefore is not void of *inductive bias*. We have to be very conscious about what it means to visualise data in a certain way. For example, by choosing to plot curves according to the increased frequency of a certain word, we might have made that aspect artificially important for the phenomenon under consideration (e.g., many other unknown factors might have contributed to the distribution of that word's frequency over the decades in consideration).

Latent structure

Here I am plotting publication dates (from last class).
Left: observations for Y. Right: observations for $Y|[\text{freq}_{\text{steam}}(x) > k]$.



The way a model *views* the data tells us something about latent factors that account for (cause or correlate with) observed variance.

Rightmost plot: Suppose we separate our observations (labelled documents) in terms of how often a document contains the word *steam*. Our historians claim that above some threshold k a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society). Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Visualisations: It might strike you as a surprise, but visualising data is very much like modelling data, and therefore is not void of *inductive bias*. We have to be very conscious about what it means to visualise data in a certain way. For example, by choosing to plot curves according to the increased frequency of a certain word, we might have made that aspect artificially important for the phenomenon under consideration (e.g., many other unknown factors might have contributed to the distribution of that word's frequency over the decades in consideration).

"But hidden states in an NN are latent structure, right?"

Latent structure here has to do with a partitioning of the probability space in terms of intermediate outcomes that depend on one another.

Hidden layers in an NN output deterministic transformations of their observed inputs. They are not statements about statistical dependence.

Example:

$$Y|w,\lambda \sim \text{Poisson}\left(\underbrace{\exp\left(\sum_{i=1}^{D} w_i h_i\right)}_{\text{small NN}}\right)$$

any one draw comes from the exact same Poisson.

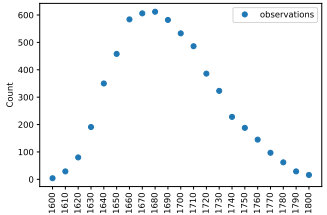
To reveal latent structure that is likely supported by observations, we need to postulate a joint distribution where observations and latent variables interact.

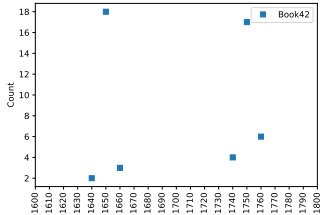
'Interacting' is a matter of **statistical dependence**.

Multimodality

Remember we had some very dedicated historians identify when certain documents had been written?

For a few books, we consulted with multiple experts.





Left: observations for Y across the collection. Right: observations for Y given x is book-42.

a unimodal conditional $Y|\theta, x$ seems an unlikely choice here.

When we plot observations for Y (e.g., left) we see the data *marginally*.

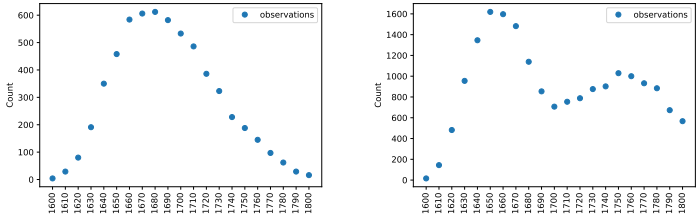
If we intend to model the data conditionally, that plot won't help us pick a likelihood family. That is because our choice should be informed by plots of the kind $Y|x$. If we group our data into bins, where bin membership depends on matching a specific value of x, more often than not our bins will each contain a single data point. Should we conclude that *conditionally* our data can be seen as deterministic? By no means!

Be aware of sneaky modelling assumptions. The combination of *'1 bin per unique document'* and *'one plot per bin'* is a modelling choice (for visualisation purposes, but still). One that suffers from data sparsity so tremendously that it makes a random variable look deterministic. Concluding that we can model the data deterministically is in fact an instance of overfitting (by humans).

Note that sometimes we can *construct* more meaningful $Y|x$ plots that reveal the stochastic nature of the data. For example, if we have direct access to the mechanism by which observations are generated, we can fix $X = x$ and draw Y multiple times (rightmost plot on the slide).

"But NNs can, in principle, learn anything, right?"

Not quite. We predict a *probability distribution* by parameterising probability *mass or density functions*. Thus our models are limited by the expressiveness of the mass/density functions we choose.



Left: data look unimodal and could have been drawn from a Poisson.
Right: data look bimodal and could not have been drawn from any Poisson.

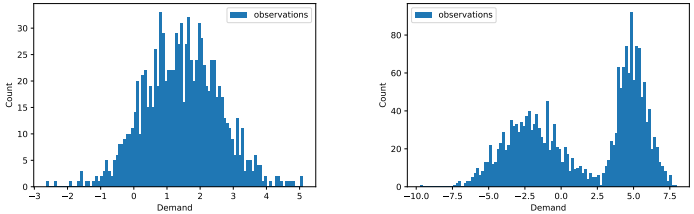
Does this mean that we are better off predicting outcomes (in data space) rather than distributions?

No, not really. Say we have some predictor $x \in \mathcal{X}$ and some response $y \in \mathcal{Y}$. We can train an NN to *deterministically* map any one x to a single y. This excludes problems where the response exhibits variance (too large a class to be overlooked!). If instead we train an NN to stochastically map any one x to some $y \in \mathcal{Y}$ with certain probability (even if unknown), we have essentially specified a probability distribution, but via an *implicit* mechanism. That is, we've parameterised a *random generator* rather than a prescribed parametric mass/density function. Such *implicit models* can learn very (even arbitrarily) flexible distributions, but they pose many technical challenges. For example, mass/density assessments are intractable, if your algorithm for parameter estimation requires those, you will have a hard time; if your data are discrete (NLP is full of examples), and you rely on derivatives for parameter estimation, you will be in big trouble.

So we will go on with prescribed distributions, and if we need more flexible distributions (right), we will have to construct them carefully. For example, via marginalisation of latent random variables.

"But NNs can, in principle, learn anything, right?"

Not quite. We predict a *probability distribution* by parameterising probability *mass or density functions*. Thus our models are limited by the expressiveness of the mass/density functions we choose.



Left: data look unimodal and could have been drawn from a Gaussian.
Right: data look bimodal and could not have been drawn from any Gaussian.

Does this mean that we are better off predicting outcomes (in data space) rather than distributions?

No, not really. Say we have some predictor $x \in \mathcal{X}$ and some response $y \in \mathcal{Y}$. We can train an NN to *deterministically* map any one x to a single y. This excludes problems where the response exhibits variance (too large a class to be overlooked!). If instead we train an NN to stochastically map any one x to some $y \in \mathcal{Y}$ with certain probability (even if unknown), we have essentially specified a probability distribution, but via an *implicit* mechanism. That is, we've parameterised a *random generator* rather than a prescribed parametric mass/density function. Such *implicit models* can learn very (even arbitrarily) flexible distributions, but they pose many technical challenges. For example, mass/density assessments are intractable, if your algorithm for parameter estimation requires those, you will have a hard time; if your data are discrete (NLP is full of examples), and you rely on derivatives for parameter estimation, you will be in big trouble.

So we will go on with prescribed distributions, and if we need more flexible distributions (right), we will have to construct them carefully. For example, via marginalisation of latent random variables.

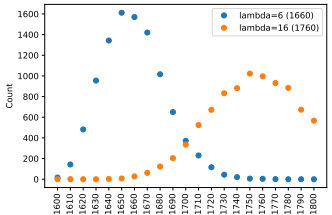
Can we combine simple distributions?

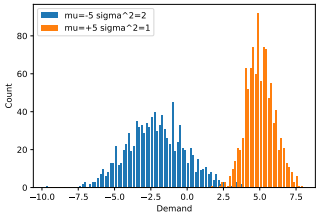
We can however *mix* K members of each family to get a good fit:

For example, with $K = 2$

$$Z|b \sim \text{Bernoulli}(b)$$
$$Y|\lambda, b \sim \text{Poisson}(\lambda_z)$$

$$Z|b \sim \text{Bernoulli}(b)$$
$$Y|\mu, \sigma, b \sim \mathcal{N}(\mu_z, \sigma_z^2)$$





This is known as a **mixture model**. The specific ones on the slide combines two conditional distributions, namely, $Y|\theta, Z = 0$ and $Y|\theta, Z = 1$. The model mixes its conditional *components* stochastically, a process controlled by a distribution over components, whose probabilities $p(z|\theta)$ are known as *mixing weights*. That is, with probability $p(z|\theta)$ the component $Y|\theta, Z = z$ generates a draw in $\mathcal{Y}$. In this example, $p(Z = 1|\theta) = b$ and $p(Z = 0|\theta) = 1 - b$.

For $K > 2$ components, $Z|\pi \sim \text{Cat}(\pi_1, \ldots, \pi_K)$, thus $p(z|\pi) = \pi_z$.

Mixture model

A **mixture model** assigns probability

$$p(z, y|\theta) = p(z|\theta)p(y|z, \theta)$$

to joint outcomes in the sample space $\mathcal{Z} \times \mathcal{Y}$. That is, it prescribes a *joint distribution* over observed and unobserved random variables.

A mixture model encodes the assumption that data points are each drawn from one of a finite number of independent distributions.

The latent variable Z captures this *unobserved component assignment*. It is governed by a distribution we call the *prior*. Oftentimes this is as simple as a uniform distribution over the sample space $\mathcal{Z}$.

Given an observation y drawn from the mixture, we can *infer* a distribution over component assignments by basic probability calculus, this very famous result is known as Bayes rule:

$$p(z|y, \theta) = \frac{p(z, y|\theta)}{p(y|\theta)} = \frac{p(y|z, \theta)p(z|\theta)}{\sum_{z' \in \mathcal{Z}} p(y|z', \theta)p(z'|\theta)}$$

Note that this *posterior distribution* $p(z|y, \theta)$ involves the *marginal distribution* $p(y|\theta)$, which we discuss next.

Prescribing multimodal distributions

The **marginal** distribution of the mixture model is potentially multimodal. It assigns probability

$$p(y|\theta) = \sum_{z=1}^{K} p(z|\theta)p(y|z,\theta)$$

to an outcome $y \in \mathcal{Y}$ by *marginalisation* of assignments $z \in \mathcal{Z}$ of the latent random variable.

A draw from the marginal of the mixture model is a draw from *one*, and *only one*, of its components (selected by drawing Z). Don't confuse mixing, in the mixture model sense, with interpolating.

Say we have a dataset $\mathcal{D}$ of N i.i.d. observations for Y. Maximum likelihood estimation of θ depends on the log-likelihood function, which in turn depends on assessments of the *marginal likelihood* of each observation:

$$\begin{aligned}\mathcal{L}_{\mathcal{D}}(\theta) &= \sum_{s=1}^{N} \log p(y^{(s)}|\theta) \\ &= \sum_{s=1}^{N} \log \sum_{z^{(s)}=1}^{K} p(y^{(s)}, z^{(s)}|\theta) \\ &= \sum_{s=1}^{N} \log \sum_{z^{(s)}=1}^{K} p(y^{(s)}|z^{(s)},\theta)p(z^{(s)}|\theta)\end{aligned}$$

This scales linearly in the number of *components*.

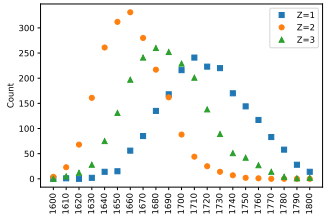
Interpolation is something like this: $\sum_{i=}^{n} w_i y_i$. Note this is note defined for discrete variables. For numerical variables it is often not in the sample space: an interpolation of Poisson draws can be a real value; an interpolation of unit-norm vectors typically has norm less than 1.

Posterior component assignment

Given an observation y we can infer a distribution over component assignments via Bayes rule

$$p(z|y,\theta) = \frac{p(z,y|\theta)}{p(y|\theta)} = \frac{p(y|z,\theta)p(z|\theta)}{\sum_{z' \in \mathcal{Z}} p(y|z',\theta)p(z'|\theta)}$$

MMs are one of the first options when it comes to organising massive collections of unlabelled data into smaller groups (clustering).



Here is a mixture model (3 Poisson components) of our historical data.

Components in a mixture model are not *labelled* with self-evident information such as *'pre-steam-engine'* and *'post-steam-engine'*, but sometimes by inspecting likely component assignments we can recognise some salient features data bring data points together under a certain component. We can also use it to target annotation efforts, for example, to avoid under-representing certain decades (in our running example).

Labelling components with self-evident information can be done by experts with assistance of posterior queries or even semi-automatically by extending mixture models in interesting ways. See LDA (Blei et al., 2003), for example.

Predictors are welcome

It is also possible to use mixture models in conditional models:

$$p(z, y|x, \theta) = p(z|x, \theta)p(y|x, z, \theta)$$

and we may use x in different ways, e.g.

$$p(z, y|x, \theta) = p(z|\theta)p(y|x, z, \theta)$$
$$p(z, y|x, \theta) = p(z|x, \theta)p(y|z, \theta)$$

Whether to use predictors to parameterise mixing weights, the conditional model, or both will depend on the application.

'Parameterising mixing weights' means specifying a distribution over K mixture components, e.g.

$$Z|\theta, x \sim \text{Cat}(g(x; \theta))$$

An alternative to giving control of mixing weights to a neural network, or fixing the weights to something superficially intuitive (like a uniform distribution), is to prescribe a *prior* distribution over the mixing coefficients. This would get you very close to Bayesian realms. Do you know any distribution which has the space of K-dimensional probability vectors as support?

Semi-supervised learning

Suppose some documents are annotated and others are not (as in the example), and say we model generatively.

For labelled documents, we observe (x, y) whose likelihood is

$$p(x, y|\theta) = p(y|\theta)p(x|y, \theta)$$

and the likelihood of an unlabelled document x is

$$p(x|\theta) = \sum_{y \in \mathcal{Y}} p(y|\theta)p(x|y, \theta)$$

Under the assumption that $\mathcal{Y}$ is a finite set, the likelihood of an observed document is the marginal likelihood of a **mixture model**!

This is a very special mixture model for its components are *labelled* with self-evident information, they are decades!

A generative model of this kind can be thought of as a classifier (the *modelling a task* point of view), after all, we need only apply Bayes rule to obtain a conditional $p(y|x_*, \theta)$ that can power a decision rule for a novel document x_*. And indeed, there are cases where this formulation improves classification performance.

But, above all, a generative model of this kind is a model of all of our observations (the *modelling a random experiment* point of view). Our observations are indeed a collection of documents, where some documents (very few) are labelled for decade. Isn't it rather arbitrary that we decided to call the labelled instances *training data* and ignore all unlabelled instances (the vast majority of our observations)? Isn't it odd that most of the data has not impact on any of our statistical considerations? This generally happens when we are *task*-driven more than data-driven.

Besides powering a classification rule, the generative formulation could be used to shed light onto vocabulary shifts over the decades. One way to specify the component $p(x|y, \theta)$ is to assume it generates a document by drawing words independently given a decade-specific parameter θ_y. That is, $X_i|\theta, y \sim \text{Cat}(\theta_y)$ for $i = 1, \ldots, |x|$.

Competition or cooperation?

In a mixture model the components *compete* to generate a data point. This means they cannot *cooperate* to account for some observed variance.

Sometimes, however, we want to stipulate the presence of a number of latent factors that together contribute to our observations distributing the way they do. Think of it in terms of clustering: sometimes we need overlapping clusters, or rather, *attributes*.

For example, our documents are scientific documents, and the period in consideration covers the European Scientific Revolution, as it came to be named. A number of inventions and new ideas marked this period. Documents were likely influenced by subsets of those ideas, rather than any singe idea in particular.

Like in mixture models we can recognise two roles for the class of models we are about to develop.

They can serve task-driven goals and power models that can predict attributes of an input (e.g., attributes of product, aspects of review, morphological features of a word).

They can serve knowledge-seeking goals and power inferences about latent structure that account (cause or correlate with) observed variance (e.g., in what latent aspects/dimensions are data points related).

Latent Factor Models

We have a prior over D-dimensional factor vectors (typically binary vectors)

$$Z_d|b \sim \text{Bernoulli}(b_d) \qquad d = 1, \ldots, D$$

A sample $z = \langle z_1, \ldots, z_D \rangle$ is used to parameterises a choice of likelihood.

For example, if we Y is Poisson distributed, we might have

$$Y|w, z \sim \text{Poisson}\left(\exp\left(\sum_{d=1}^{D} w_d z_d\right)\right)$$

This induces a marginal likelihood

$$p(y|b, w) = \sum_{z \in \{0,1\}^D} p(y, z|w, b) = \sum_{z \in \{0,1\}^D} p(z|b)p(y|z, w)$$

You can think of a latent factor model as specifying an exponential number of components, literally, up to 2^D components, since each z in its sample space can potentially lead to a different Poisson parameter. But the components are *not* independent, they are all predicted by the same log-linear model (in the example, the logarithm of the Poisson parameter is a linear combination of z, i.e., D random binary attributes).

Whereas the marginal likelihood of the MM depends linearly on the number of components, the marginal likelihood of the latent factor model depends exponentially on the number of factors.

Once again, predictors are welcome. They can be used to parameterise the distribution over factors, i.e., $p(z|x, \theta)$ and/or the conditional $p(y|x, z, \theta)$.

Summary

Mixture model ('learning clusters')

Latent factor model ('learning attributes or overlapping clusters')

Applications:

- unsupervised learning (e.g., word alignments, LDA, IBP)
- semi-supervised learning (e.g., generative classifiers, disentanglement learning)
- transparency (e.g., latent rationales)

Examples:

- word alignments (Brown et al., 1993; Vogel et al., 1996; Rios et al., 2018)
- LDA (Blei et al., 2003)
- IBP (Ghahramani and Griffiths, 2006)
- semi-supervised deep generative models (Kingma et al., 2014; Zhou and Neubig, 2017)
- latent rationales (Lei et al., 2016; Bastings et al., 2019)

Outline

Latent Variable Models

When talking about some generic model I will follow this convention

- X is a random variable taking on values in a sample space $\mathcal{X}$
- $x \in \mathcal{X}$ is an observation
- Z is a *discrete* random variable taking on values in a sample space $\mathcal{Z}$
- $z \in \mathcal{Z}$ is a latent assignment
- the joint distribution factorises as $p(x, z|\theta) = p(z|\theta)p(x|z, \theta)$
- $p(z|\theta)$ is called the *prior*
- $p(x|z, \theta)$ is called the *likelihood*
- $p(x|\theta)$ is the *marginal likelihood* (or *evidence*)
- and $p(z|x, \theta) = \frac{p(x,z|\theta)}{p(x|\theta)}$ is the posterior
- anything in the model can be parameterised by NNs

Please don't get confused, x here is not a predictor. To avoid clutter, I will be omitting any potential predictors. If at one point I need to introduce input-output type variables again, for example, when discussing a specific application, I will say so explicitly.

Throughout, we shall assume we have N i.i.d. observations. With deterministic parameters θ, we can make all our arguments in terms of a single observation x. Recall, the likelihood-function $\mathcal{L}_{\mathcal{D}}(\theta)$ is just $\sum_{x \sim \mathcal{D}} \log p(x|\theta)$.

By the way, can you draw a plate diagram for our generic latent variable model?

Many models admit exact marginals

Examples (and the algorithms for marginalisation)

- Mixture models (enumeration)
- HMMs (forward algorithm)
- CFGs (inside algorithm)
- Spanning-tree random fields (matrix-tree theorem)

Tractable marginalisation depends on the conditional independence assumptions of a model (e.g., in an HMM a hidden state is independent of all but its preceding state), not on how that model's probability distributions are parameterised (e.g., a transition distribution in the HMM may be stored in a table, predicted by a log-linear model or by an NN).

Marginalisation algorithms are generally harder to parallelise on GPUs.

Recall that to use NNs in probabilistic models we converged to two constraints on the log-likelihood function:

- differentiability with respect to parameters
- and tractability

If $p(z|\theta)$ and $p(x|z,\theta)$ are differentiable functions of their parameters, there is no impediment to gradient-based parameter estimation. Can you show that to yourself? Hint: expand $\nabla_\theta \log p(x|\theta)$.

Tractability depends on whether $p(x|\theta) = \sum_{z \in \mathcal{Z}} p(x, z|\theta)$, or its logarithm, can be evaluated in feasible time. Though it may seem so, this is not always a matter of cardinality of $\mathcal{Z}$.

For example, there is a Catalan number of trees in a CFG, yet because of the strong independence assumptions in the model, the marginal likelihood is computable in cubic-time (w.r.t. sequence length) via the inside algorithm. Similarly, there is an exponential number of state sequences in an HMM, but its marginal likelihood is computable in linear-time (w.r.t. sequence length) via the forward algorithm.

Neural {MM, HMM, CFG, CRF, ... }

An NN-parameterisation of a classic discrete LVM, for which exact marginals are tractable, still needs to preserve all of that model's statistical assumptions about unobserved random variables.

We won't necessarily achieve a more complex distribution.

Though we may condition on complex data more effectively.

A neural HMM could look like:

$$p(x|\theta) = \sum_{z \in \{1,\dots,K\}^{|x|}} \prod_{i=1}^{|x|} \underbrace{p(z_i|z_{i-1}, x_{<i}, \theta)}_{\mathrm{Cat}(z_i|g(x_{<i}, z_{i-1}; \theta))} \underbrace{p(x_i|z_i, x_{<i})}_{\mathrm{Cat}(x_i|f(x_{<i}, z_i; \theta))}$$

the entire history of already generated words is available for conditioning

NNs allow us to condition on complex observations, like a long history of words $x_{<i}$ in unsupervised part-of-speech tagging.

We cannot, as easily, exploit that power to relax statistical conditional independence assumptions, for those assumptions are crucial in order to maintain *exact and tractable* access to marginal probabilities.

Think of it this way, what makes the HMM the HMM is the first-order (or n-order) Markov assumption $Z_i \perp Z_j | Z_{i-1}$ for j other than i and $i-1$. Relaxing that turns the HMM into something else, for which exact inference is likely impossible. See Wang et al. (2018) for a neural HMM.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) =$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta)$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta)$$

$$= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) =$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta)$$

$$= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) = \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} p(z, x|\theta) \nabla_\theta \log p(z, x|\theta)$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.
- Sums are fine, but let's use the log identity $f' = f(\log f)'$

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$
\begin{aligned}
&= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta) \\
&= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) = \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} p(z, x|\theta) \nabla_\theta \log p(z, x|\theta) \\
&= \sum_{z \in \mathcal{Z}} \frac{p(z, x|\theta)}{p(x|\theta)} \nabla_\theta \log p(z, x|\theta) =
\end{aligned}
$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.
- Sums are fine, but let's use the log identity $f' = f(\log f)'$
- The marginal is constant for $z \in \mathcal{Z}$, so distribute it over the sum.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$
\begin{aligned}
&= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta) \\
&= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) = \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} p(z, x|\theta) \nabla_\theta \log p(z, x|\theta) \\
&= \sum_{z \in \mathcal{Z}} \frac{p(z, x|\theta)}{p(x|\theta)} \nabla_\theta \log p(z, x|\theta) = \sum_{z \in \mathcal{Z}} p(z|x, \theta) \nabla_\theta \log p(z, x|\theta)
\end{aligned}
$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.
- Sums are fine, but let's use the log identity $f' = f(\log f)'$
- The marginal is constant for $z \in \mathcal{Z}$, so distribute it over the sum.
- This gives us a recognisable object! Joint probability, divided by evidence, that's the posterior! And we have a weighted average, coefficients given by a pmf, and we sum over the entire support $\mathcal{Z}$.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$
\begin{aligned}
&= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta) \\
&= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) = \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} p(z, x|\theta) \nabla_\theta \log p(z, x|\theta) \\
&= \sum_{z \in \mathcal{Z}} \frac{p(z, x|\theta)}{p(x|\theta)} \nabla_\theta \log p(z, x|\theta) = \sum_{z \in \mathcal{Z}} p(z|x, \theta) \nabla_\theta \log p(z, x|\theta) \\
&= \mathbb{E}_{p(z|x,\theta)} \left[\nabla_\theta \log p(z, x|\theta) \right]
\end{aligned}
$$

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.
- Sums are fine, but let's use the log identity $f' = f(\log f)'$
- The marginal is constant for $z \in \mathcal{Z}$, so distribute it over the sum.
- This gives us a recognisable object! Joint probability, divided by evidence, that's the posterior! And we have a weighted average, coefficients given by a pmf, and we sum over the entire support $\mathcal{Z}$.
- We have an expectation! The gradient of the log-marginal of x is the expected gradient of the log joint probability of x and z, where x is observed and z is a draw from the posterior distribution $Z|x, \theta$. Dependency on Z makes the gradient of log-joint $G(Z) = \nabla_\theta \log P(Z, X = x)$ a random variable.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$
\begin{aligned}
&= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta) \\
&= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) = \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} p(z, x|\theta) \nabla_\theta \log p(z, x|\theta) \\
&= \sum_{z \in \mathcal{Z}} \frac{p(z, x|\theta)}{p(x|\theta)} \nabla_\theta \log p(z, x|\theta) = \sum_{z \in \mathcal{Z}} p(z|x, \theta) \nabla_\theta \log p(z, x|\theta) \\
&= \mathbb{E}_{p(z|x,\theta)} \left[\nabla_\theta \log p(z, x|\theta) \right]
\end{aligned}
$$

Autodiff performs **exact posterior inference** for us!

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.
- Sums are fine, but let's use the log identity $f' = f(\log f)'$
- The marginal is constant for $z \in \mathcal{Z}$, so distribute it over the sum.
- This gives us a recognisable object! Joint probability, divided by evidence, that's the posterior! And we have a weighted average, coefficients given by a pmf, and we sum over the entire support $\mathcal{Z}$.
- We have an expectation! The gradient of the log-marginal of x is the expected gradient of the log joint probability of x and z, where x is observed and z is a draw from the posterior distribution $Z|x, \theta$. Dependency on Z makes the gradient of log-joint $G(Z) = \nabla_\theta \log P(Z, X = x)$ a random variable.
- The gradient of the log-marginal $\nabla_\theta \log p(x|\theta)$ is deterministic, it is the expected value $\mathbb{E}_{Z|X=x,\theta}[G(Z)]$. You evaluate the marginal, autodiff evaluates the expectation.

Gradient-based MLE

What happens when we autodiff the quantity $\log p(x|\theta)$, which we computed exactly and tractably?

Let's inspect this gradient ourselves $\nabla_\theta \log p(x|\theta)$

$$
\begin{aligned}
&= \frac{1}{p(x|\theta)} \nabla_\theta p(x|\theta) = \frac{1}{p(x|\theta)} \nabla_\theta \sum_{z \in \mathcal{Z}} p(z, x|\theta) \\
&= \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} \nabla_\theta p(z, x|\theta) = \frac{1}{p(x|\theta)} \sum_{z \in \mathcal{Z}} p(z, x|\theta) \nabla_\theta \log p(z, x|\theta) \\
&= \sum_{z \in \mathcal{Z}} \frac{p(z, x|\theta)}{p(x|\theta)} \nabla_\theta \log p(z, x|\theta) = \sum_{z \in \mathcal{Z}} p(z|x, \theta) \nabla_\theta \log p(z, x|\theta) \\
&= \mathbb{E}_{p(z|x,\theta)} \left[\nabla_\theta \log p(z, x|\theta) \right]
\end{aligned}
$$

Autodiff performs **exact posterior inference** for us!

Gradient of log-marginal

- It all starts with the derivative of log, followed by chain rule again.
- The next step requires marginalisation.
- Now we need the gradient of a big sum.
- Derivatives are linear, so we can sum gradients instead.
- Sums are fine, but let's use the log identity $f' = f(\log f)'$
- The marginal is constant for $z \in \mathcal{Z}$, so distribute it over the sum.
- This gives us a recognisable object! Joint probability, divided by evidence, that's the posterior! And we have a weighted average, coefficients given by a pmf, and we sum over the entire support $\mathcal{Z}$.
- We have an expectation! The gradient of the log-marginal of x is the expected gradient of the log joint probability of x and z, where x is observed and z is a draw from the posterior distribution $Z|x, \theta$. Dependency on Z makes the gradient of log-joint $G(Z) = \nabla_\theta \log P(Z, X = x)$ a random variable.
- The gradient of the log-marginal $\nabla_\theta \log p(x|\theta)$ is deterministic, it is the expected value $\mathbb{E}_{Z|X=x,\theta}[G(Z)]$. You evaluate the marginal, autodiff evaluates the expectation.

Summary

Many discrete LVMs admit tractable marginalisation

Assessing the gradient of the log-marginal probability of an observation corresponds to assessing an expectation under the posterior distribution over latent variables. Think of it this way:

- we need posterior inference to compute the gradient
- and we need the gradient for parameter estimation
- with exact marginals, autodiff assesses the gradient thus abstracting posterior inference away

What happens when we cannot solve $\sum_{z \in \mathcal{Z}} p(x, z|\theta)$?

Many interesting models are such that the exact marginal is intractable. We've seen, for example, the case where $p(x, z|\theta)$ is a latent factor model.

Autodiff cannot differentiate a quantity that cannot be assessed. So if we cannot compute the exact log-marginal probability of an observation, we won't get automatic posterior inference for free. We will have to resort to rather explicit approaches to approximate inference.

Outline

Latent factor document model

Let us consider a latent factor model for document modelling:

- a document $x = (x_1, \ldots, x_n)$ consists of n i.i.d. categorical draws from that model
- the categorical distribution in turn depends on binary latent factors $z = (z_1, \ldots, z_D)$ which are also i.i.d.

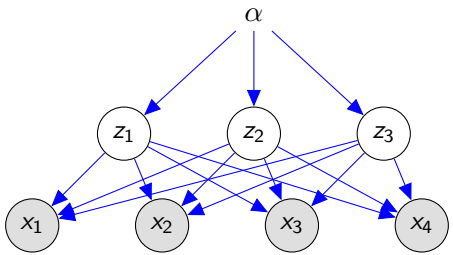
$$\begin{aligned} Z_j &\sim \text{Bernoulli}(\alpha) && (1 \leq d \leq D) \\ X_i|z &\sim \text{Categorical}\left(f(z;\theta)\right) && (1 \leq i \leq n) \end{aligned}$$

Here $0 < \alpha < 1$ specifies a Bernoulli prior
and $f(\cdot;\theta)$ is a function computed by an NN, e.g.:

$$\begin{aligned} f(z;\theta) &= \text{softmax}(Wz + b) \\ \theta &= \{W, b\} \end{aligned}$$

To keep the model simple we will assume $X_i \perp X_j | Z$ for $i \neq j$. We could, however, relax this conditional independence if we wanted. For example, we could model $X_i|\theta, z, x_{<i} \sim \text{Cat}(f(z, x_{<i}; \theta))$.

Graphical model



Joint distribution: independent latent variables

Suppose, for example, $D = 3$ and $n = 4$.

I omit θ from the graphical model, but every $X_i|z$ depends on it.

Intractable Marginals

In the latent factor model, marginalisation takes time $\mathcal{O}(2^D)$

$$p(x|\alpha,\theta) = \sum_{z\in\{0,1\}^D} p(z|\alpha)p(x|z,\theta)$$
$$= \sum_{z\in\{0,1\}^D} \prod_{d=1}^{D} \text{Bern}(z_d|\alpha) \prod_{i=1}^{n} \text{Cat}(x_i|f(z;\theta))$$

As a consequence, we cannot assess $\log p(x|\alpha,\theta)$ nor its gradient.

But we know that

$$\nabla_\theta \log p(x|\alpha,\theta) = \mathbb{E}_{p(z|x,\alpha,\theta)}[\nabla_\theta \log p(x,z|\alpha,\theta)]$$

Revision: in a previous version we missed the ∇_θ operator on the right-hand side.

Unfortunately, we cannot count on autodiff to solve the expectation for us in this case.

Perhaps we can estimate the gradient?

Gradient estimates?

Monte Carlo to the rescue?

$$\nabla_\theta \log p(x|\theta) = \mathbb{E}_{p(z|x,\theta)}\left[\nabla_\theta \log p(x,z|\theta)\right]$$
$$\overset{\text{MC}}{\approx} \frac{1}{K}\sum_{k=1}^{K} \nabla_\theta \log p(x, z^{(k)}|\theta) \quad \text{where } z^{(k)} \sim p(z|x,\theta)$$

But the posterior is not available either!

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

I am omitting α, the prior parameter. For simplicity, assume it is fixed.

We know that MC estimation gives us an unbiased estimate of the gradient. But MC requires sampling independently from the distribution of the random variable $Z|X = x, \theta$, that is the posterior distribution in this case.

We may know the family to which the posterior belongs:

- The posterior of the *mixture model* is a Categorical distribution (just like the prior), we also know an algorithm to assess the posterior Categorical parameter: marginalisation by enumeration). Then, $Z|\theta, x \sim \text{Cat}(\pi)$ where $\pi_z = \frac{p(x,z|\theta)}{\sum_{z'=1}^{K} p(x,z'|\theta)}$.
- The posterior of the latent factor model is a Gibbs distribution (think of it as a generalisation of the Categorical distribution to combinatorial sample spaces, like the space of binary vectors), and we know the algorithm to assess its parameter, again, it's marginalisation by enumeration. This time, however, enumeration is intractable: $Z|\theta, x \sim \text{Gibbs}(\pi)$ where $\pi_z = \frac{p(x,z|\theta)}{\sum_{z' \in \{0,1\}^D} p(x,z'|\theta)}$.

Sometimes **we do not even known the family**. For example, if $Z \in \mathbb{N}^D$ and $Z_d \sim \text{Poisson}(\alpha)$, we have latent factor model with ordinal attributes. The posterior family is *unknown*. It is not Gibbs, because the sample space is unbounded, it's not a product of Poisson distributions, because Z_d are correlated in the posterior.

The Basic Problem

We want to compute the posterior over latent variables $p(z|x, \theta)$. This involves computing the marginal likelihood

$$p(x|\theta) = \sum_{z \in \mathcal{Z}} p(x, z|\theta)$$

which is generally **intractable**. This problem motivates the use of **approximate inference** techniques.

You may think all we really need is the marginal and we shouldn't bother about the posterior. I'd argue there's barely a difference. Probabilistic inference (i.e., computations involving probability calculus) is the core of the problem. Sometimes we see it as the need for a marginal, sometimes as the need for a posterior.

Strategy

Variational Inference

- Accept that $p(z|x,\theta)$ is not computable.
- Approximate it by an auxiliary distribution $q(z)$ that is computable!
- Choose $q(z)$ as close as possible to $p(z|x,\theta)$ to obtain a faithful approximation.

We are going to derive VI's objective from two points of view

- first, we will concentrate on the intractable log-marginal, and attempt to bound it by a tractable quantity;
- then we will concentrate on the intractable posterior, and attempt to learn a tractable approximation to it;

The two views will turn out intimately related.

This is the outline for **variational inference** (VI; Jordan et al., 1999; Blei et al., 2017).

There are alternatives to VI, but they are not covered in this course. Here are some pointers:

- Markov chain Monte Carlo (MCMC). Here is an excellent material by Michael Betancourt: `https://betanalpha.github.io/assets/case_studies/markov_chain_monte_carlo.html`.
- Expectation propagation (EP; Minka, 2001; Vehtari et al., 2020)

Outline

VI derivation I

Deriving VI from the log-evidence:

$$\log p(x|\theta) = \log \sum_{z \in \mathcal{Z}} p(x, z|\theta)$$

The evidence lower-bound (ELBO):

- Let's start from the log marginal.

VI derivation I

Deriving VI from the log-evidence:

$$\log p(x|\theta) = \log \sum_{z \in \mathcal{Z}} p(x, z|\theta)$$
$$= \log \sum_{z \in \mathcal{Z}} q(z) \frac{p(x, z|\theta)}{q(z)}$$

The evidence lower-bound (ELBO):

- Let's start from the log marginal.
- And introduce $q(z)$ such that $q(z) > 0$ if $p(z|x, \theta) > 0$.

VI derivation I

Deriving VI from the log-evidence:

$$\begin{aligned}\log p(x|\theta) &= \log \sum_{z\in\mathcal{Z}} p(x,z|\theta) \\ &= \log \sum_{z\in\mathcal{Z}} q(z)\frac{p(x,z|\theta)}{q(z)} \\ &= \log \mathbb{E}_{q(z)}\left[\frac{p(x,z|\theta)}{q(z)}\right]\end{aligned}$$

The evidence lower-bound (ELBO):

- Let's start from the log marginal.
- And introduce $q(z)$ such that $q(z) > 0$ if $p(z|x,\theta) > 0$.
- And note we got an expectation w.r.t. $q(z)$, and recall that, unlike $p(z|x,\theta)$, we know $q(z)$, as we chose it.

VI derivation I

Deriving VI from the log-evidence:

$$\begin{aligned}
\log p(x|\theta) &= \log \sum_{z \in \mathcal{Z}} p(x, z|\theta) \\
&= \log \sum_{z \in \mathcal{Z}} q(z) \frac{p(x, z|\theta)}{q(z)} \\
&= \log \mathbb{E}_{q(z)} \left[\frac{p(x, z|\theta)}{q(z)} \right] \\
&\overset{\text{JI}}{\geq} \mathbb{E}_{q(z)} \left[\log \frac{p(x, z|\theta)}{q(z)} \right]
\end{aligned}$$

The evidence lower-bound (ELBO):

- Let's start from the log marginal.
- And introduce $q(z)$ such that $q(z) > 0$ if $p(z|x, \theta) > 0$.
- And note we got an expectation w.r.t. $q(z)$, and recall that, unlike $p(z|x, \theta)$, we know $q(z)$, as we chose it.
- And Jensen's inequality allows us to push the log through the expectation.

VI derivation I

Deriving VI from the log-evidence:

$$\begin{aligned}\log p(x|\theta) &= \log \sum_{z\in\mathcal{Z}} p(x,z|\theta) \\ &= \log \sum_{z\in\mathcal{Z}} q(z)\frac{p(x,z|\theta)}{q(z)} \\ &= \log \mathbb{E}_{q(z)}\left[\frac{p(x,z|\theta)}{q(z)}\right] \\ &\overset{\text{JI}}{\geq} \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right]\end{aligned}$$

This is the *lowerbound on the log-evidence*, also known as ELBO. Crucially, it **does not require the true posterior**!

The evidence lower-bound (ELBO):

- Let's start from the log marginal.
- And introduce $q(z)$ such that $q(z) > 0$ if $p(z|x,\theta) > 0$.
- And note we got an expectation w.r.t. $q(z)$, and recall that, unlike $p(z|x,\theta)$, we know $q(z)$, as we chose it.
- And Jensen's inequality allows us to push the log through the expectation.

VI derivation I

Let's gain insight about this bound

$$\log p(x|\theta) \geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right]$$

What can be said about $q(z)$?

- Let's start from the ELBO.

VI derivation I

Let's gain insight about this bound

$$\log p(x|\theta) \geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right]$$
$$= \mathbb{E}_{q(z)}\left[\log \frac{p(z|x,\theta)p(x|\theta)}{q(z)}\right]$$

What can be said about $q(z)$?

- Let's start from the ELBO.
- Now let's factorise the joint probability using the marginal and the posterior (these are clearly not available to us, but they will help us understand what is going on).

VI derivation I

Let's gain insight about this bound

$$\begin{aligned}\log p(x|\theta) &\geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right] \\ &= \mathbb{E}_{q(z)}\left[\log \frac{p(z|x,\theta)p(x|\theta)}{q(z)}\right] \\ &= \log p(x|\theta) + \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)}\end{aligned}$$

What can be said about $q(z)$?

- Let's start from the ELBO.
- Now let's factorise the joint probability using the marginal and the posterior (these are clearly not available to us, but they will help us understand what is going on).
- The log-marginal is constant w.r.t. z, thus its expected valued under $q(z)$ is itself, i.e., $\log p(x|\theta)$.

VI derivation I

Let's gain insight about this bound

$$\begin{aligned}\log p(x|\theta) &\geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right] \\ &= \mathbb{E}_{q(z)}\left[\log \frac{p(z|x,\theta)p(x|\theta)}{q(z)}\right] \\ &= \log p(x|\theta) + \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\ &= \log p(x|\theta) - \sum_{z\in\mathcal{Z}} q(z) \log \frac{q(z)}{p(z|x,\theta)}\end{aligned}$$

What can be said about $q(z)$?

- Let's start from the ELBO.
- Now let's factorise the joint probability using the marginal and the posterior (these are clearly not available to us, but they will help us understand what is going on).
- The log-marginal is constant w.r.t. z, thus its expected valued under $q(z)$ is itself, i.e., $\log p(x|\theta)$.
- We can apply a property of logs to rearrange the fraction.

VI derivation I

Let's gain insight about this bound

$$
\begin{aligned}
\log p(x|\theta) &\geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right] \\
&= \mathbb{E}_{q(z)}\left[\log \frac{p(z|x,\theta)p(x|\theta)}{q(z)}\right] \\
&= \log p(x|\theta) + \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \log p(x|\theta) - \sum_{z\in\mathcal{Z}} q(z) \log \frac{q(z)}{p(z|x,\theta)} \\
&= \log p(x|\theta) - \underbrace{\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)}_{\geq 0}
\end{aligned}
$$

What can be said about $q(z)$?

- Let's start from the ELBO.
- Now let's factorise the joint probability using the marginal and the posterior (these are clearly not available to us, but they will help us understand what is going on).
- The log-marginal is constant w.r.t. z, thus its expected valued under $q(z)$ is itself, i.e., $\log p(x|\theta)$.
- We can apply a property of logs to rearrange the fraction.
- Which gives us the KL divergence from $p(z|x,\theta)$ to $q(z)$. Recall, $\mathrm{KL}\left(q \,||\, p\right) \geq 0$ and equality holds only if $q = p$.

VI derivation I

Let's gain insight about this bound

$$\begin{aligned}
\log p(x|\theta) &\geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right] \\
&= \mathbb{E}_{q(z)}\left[\log \frac{p(z|x,\theta)p(x|\theta)}{q(z)}\right] \\
&= \log p(x|\theta) + \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \log p(x|\theta) - \sum_{z\in\mathcal{Z}} q(z) \log \frac{q(z)}{p(z|x,\theta)} \\
&= \log p(x|\theta) - \underbrace{\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)}_{\geq 0}
\end{aligned}$$

We have derived a lower bound on the log-evidence whose gap is exactly $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$.

What can be said about $q(z)$?

- Let's start from the ELBO.
- Now let's factorise the joint probability using the marginal and the posterior (these are clearly not available to us, but they will help us understand what is going on).
- The log-marginal is constant w.r.t. z, thus its expected valued under $q(z)$ is itself, i.e., $\log p(x|\theta)$.
- We can apply a property of logs to rearrange the fraction.
- Which gives us the KL divergence from $p(z|x,\theta)$ to $q(z)$. Recall, $\mathrm{KL}\left(q \,||\, p\right) \geq 0$ and equality holds only if $q = p$.

It looks like $q(z)$ should be as close as possible to $p(z|x,\theta)$!

VI derivation I

Let's gain insight about this bound

$$\begin{aligned}
\log p(x|\theta) &\geq \mathbb{E}_{q(z)}\left[\log \frac{p(x,z|\theta)}{q(z)}\right] \\
&= \mathbb{E}_{q(z)}\left[\log \frac{p(z|x,\theta)p(x|\theta)}{q(z)}\right] \\
&= \log p(x|\theta) + \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \log p(x|\theta) - \sum_{z\in\mathcal{Z}} q(z) \log \frac{q(z)}{p(z|x,\theta)} \\
&= \log p(x|\theta) - \underbrace{\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)}_{\geq 0}
\end{aligned}$$

We have derived a lower bound on the log-evidence whose gap is exactly $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$.

What can be said about $q(z)$?

- Let's start from the ELBO.
- Now let's factorise the joint probability using the marginal and the posterior (these are clearly not available to us, but they will help us understand what is going on).
- The log-marginal is constant w.r.t. z, thus its expected valued under $q(z)$ is itself, i.e., $\log p(x|\theta)$.
- We can apply a property of logs to rearrange the fraction.
- Which gives us the KL divergence from $p(z|x,\theta)$ to $q(z)$. Recall, $\mathrm{KL}\left(q \,||\, p\right) \geq 0$ and equality holds only if $q = p$.

It looks like $q(z)$ should be as close as possible to $p(z|x,\theta)$!

Outline

VI derivation II

Derive VI by optimisation:

$$\arg\max_{q(z)} - \mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$$

The previous derivation suggests that we should attempt to choose $q(z)$ such that the gap relative to the true posterior $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$ is as small as possible.

- Let's state that objective explicitly and seek some optimum $q(z)$.

VI derivation II

Derive VI by optimisation:

$$\begin{aligned}&\arg\max_{q(z)} - \mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right) \\ &= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)}\end{aligned}$$

The previous derivation suggests that we should attempt to choose $q(z)$ such that the gap relative to the true posterior $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$ is as small as possible.

- Let's state that objective explicitly and seek some optimum $q(z)$.
- We do not have access to the true posterior probability of any z, thus let's decompose it via Bayes rule.

VI derivation II

Derive VI by optimisation:

$$\begin{aligned}
&\arg\max_{q(z)} - \mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right) \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)p(x|\theta)}
\end{aligned}$$

The previous derivation suggests that we should attempt to choose $q(z)$ such that the gap relative to the true posterior $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$ is as small as possible.

- Let's state that objective explicitly and seek some optimum $q(z)$.
- We do not have access to the true posterior probability of any z, thus let's decompose it via Bayes rule.
- Bayes rule reveals the marginal. Note that $p(x|\theta)$ does not depend on z.

VI derivation II

Derive VI by optimisation:

$$
\begin{aligned}
&\arg\max_{q(z)} - \mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right) \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)p(x|\theta)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)} - \overbrace{\log p(x|\theta)}^{constant}
\end{aligned}
$$

The previous derivation suggests that we should attempt to choose $q(z)$ such that the gap relative to the true posterior $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$ is as small as possible.

- Let's state that objective explicitly and seek some optimum $q(z)$.
- We do not have access to the true posterior probability of any z, thus let's decompose it via Bayes rule.
- Bayes rule reveals the marginal. Note that $p(x|\theta)$ does not depend on z.
- Nor it depends on our choice of $q(z)$.

VI derivation II

Derive VI by optimisation:

$$
\begin{aligned}
&\arg\max_{q(z)} - \mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right) \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)p(x|\theta)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)} - \overbrace{\log p(x|\theta)}^{\textit{constant}} \\
&= \arg\max_{q(z)} \underbrace{\sum_{z\in\mathcal{Z}} q(z) \log p(x,z|\theta)}_{\mathbb{E}_{q(z)}[\log p(x,z|\theta)]} - \underbrace{\sum_{z\in\mathcal{Z}} q(z) \log q(z)}_{-\mathbb{H}(q(z))}
\end{aligned}
$$

The previous derivation suggests that we should attempt to choose $q(z)$ such that the gap relative to the true posterior $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$ is as small as possible.

- Let's state that objective explicitly and seek some optimum $q(z)$.
- We do not have access to the true posterior probability of any z, thus let's decompose it via Bayes rule.
- Bayes rule reveals the marginal. Note that $p(x|\theta)$ does not depend on z.
- Nor it depends on our choice of $q(z)$.
- The final objective involves only joint probabilities (always tractable, by assumption), and $q(z)$.

VI derivation II

Derive VI by optimisation:

$$
\begin{aligned}
&\arg\max_{q(z)} - \mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right) \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(z|x,\theta)}{q(z)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)p(x|\theta)} \\
&= \arg\max_{q(z)} \sum_{z\in\mathcal{Z}} q(z) \log \frac{p(x,z|\theta)}{q(z)} - \overbrace{\log p(x|\theta)}^{\textit{constant}} \\
&= \arg\max_{q(z)} \underbrace{\sum_{z\in\mathcal{Z}} q(z) \log p(x,z|\theta)}_{\mathbb{E}_{q(z)}[\log p(x,z|\theta)]} - \underbrace{\sum_{z\in\mathcal{Z}} q(z) \log q(z)}_{-\mathbb{H}(q(z))}
\end{aligned}
$$

The previous derivation suggests that we should attempt to choose $q(z)$ such that the gap relative to the true posterior $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$ is as small as possible.

- Let's state that objective explicitly and seek some optimum $q(z)$.
- We do not have access to the true posterior probability of any z, thus let's decompose it via Bayes rule.
- Bayes rule reveals the marginal. Note that $p(x|\theta)$ does not depend on z.
- Nor it depends on our choice of $q(z)$.
- The final objective involves only joint probabilities (always tractable, by assumption), and $q(z)$.

ELBO

The evidence lowerbound (ELBO) is the optimisation objective in **variational inference**.

$$\begin{aligned}&\arg\max_{q(z)} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{q(z)}\left[\log p(x,z|\theta)\right]+\mathbb{H}\left(q(z)\right)\right]\\ =&\arg\max_{q(z)} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{q(z)}\left[\log p(x|z,\theta)\right]-\mathrm{KL}\left(q(z)\,||\,p(z)\right)\right]\end{aligned}$$

The ELBO circumvents intractable posterior inference by optimisation: we search the approximate posterior that is closest to the true posterior in terms of $\mathrm{KL}\left(q(z)\,||\,p(z|x,\theta)\right)$. For example, if $q(z|\lambda)$ is in a certain parametric family, we search for its parameter.

ELBO highlights

- we get to design $q(z)$, so for example, while the true posterior of a latent factor model depends on an intractable marginalisation, the approximate posterior $q(z)$ might simply combine D independent Bernoulli distributions (one per latent factor);
- as we get to pick $q(z)$, we get to choose a family that's convenient, for example, one for which we can obtain independent samples;
- tractable samples from $q(z)$ means that we can obtain MC estimates of the ELBO;
- that's because for some given $z \in \mathcal{Z}$, the ELBO only requires assessing the joint probability $\log p(x,z|\theta)$ and $\log q(z)$;
- ideally, we would choose a family for which the entropy is also tractable.

Can you show to yourself that the second expression is true?

- if $p(z)$ and $q(z)$ are in the same (exponential) family, chances are the term $\mathrm{KL}\left(q(z)\,||\,p(z)\right)$ is known in closed form;

Designing a tractable approximation

Mean field approximation:

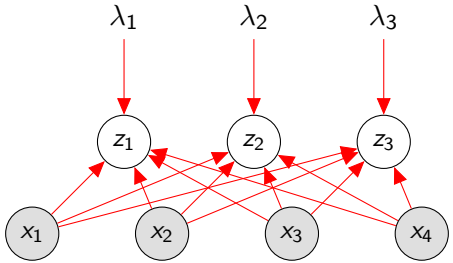
- make **all** latent variables independent under $q(z)$.
- pick a parametric family with tractable pmf.

For example, in the latent factor model this takes the form:

$$q(z|\lambda) = \prod_{d=1}^{D} q(z_d|\lambda_d)$$
$$= \prod_{d=1}^{D} \text{Bern}(z_d|\lambda_d)$$

where λ is a vector that specifies D Bernoulli parameters.

Mean Field Latent Factor Model Inference



$$Z_d|\lambda \sim \text{Bernoulli}(\lambda_d)$$

Instead of inferring the true posterior $Z|X = x, \theta$, a computation that takes assessing the marginal probability $p(x|\theta)$, and thus requires all 2^D assessments of the joint probability $p(x, z|\theta)$, **we optimise** exactly D parameters. One per Bernoulli factor in the posterior approximation $q(z|\lambda)$.

Clearly, such independence assumption is a strong simplification. In some cases we need to design *structured* approximate posteriors, that is, approximations that can correlate latent variables (we will hear more about those later).

Amortised variational inference

Amortise the cost of inference using NNs

$$q(z_1, \ldots, z_D|\lambda, x) = \prod_{d=1}^{D} q_\lambda(z_d|\lambda, x)$$

still mean field

$$Z_d|\lambda, x \sim \text{Bernoulli}(b_d)$$

but with a shared set of parameters

- where $b_1^D = \text{NN}(x; \lambda)$

The true posterior $Z|X = x, \theta$ follows from conditioning on observed x.

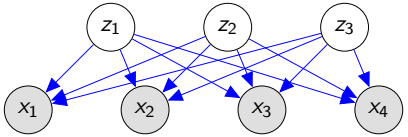
With NNs, we can condition on complex data efficiently, thus it seems like an interesting idea to jointly parameterise the independent factors of the posterior approximation $q(z|x, \lambda)$.

This leads to fewer parameters (more latent variables will not demand more parameters) and has a potentially useful by-product: an *inference model*, that is, a *model* of the latent variable.

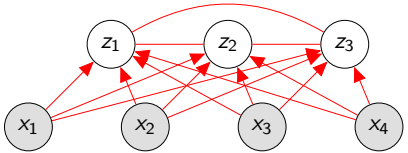
Recall our notion of model: a mechanism to predict the outcomes of a random experiment. So far, we've been attempting to model outcomes of some joint distribution $p(x, z|\theta)$. In variational inference, we introduce a rather unusual model, i.e., $q(z|x, \lambda)$, it predicts another model's posterior inferences.

Overview

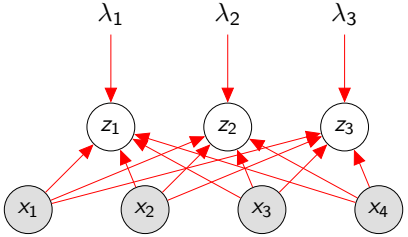
Joint distribution



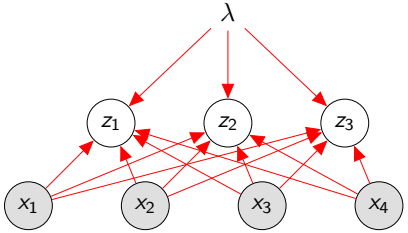
Posterior



Mean field



Amortised VI



Joint distribution: latent variables are independent a priori. This is a model assumption.

Posterior: latent variables are correlated. That is because for any $z \in \mathcal{Z}$ the value $p(z|x,\theta)$ depends on $p(x,z'|\theta)$ for all $z' \in \mathcal{Z}$ via $p(x|\theta)$.

Mean field approximation: we postulate a simple distribution over latent variables, e.g., where every variable is controlled by an independent distribution. The parameters of these distributions are chosen to maximise the ELBO.

Amortised VI: we design a probabilistic model of the latent variables. That is, we design a tractable model that maps from observations to an approximation of the true posterior distribution. This inference model is typically parameterised by an inference (neural) network, its parameters are too estimated to maximise the ELBO.

Summary

- Posterior inference is often **intractable** because the marginal likelihood (or **evidence**) $p(x|\theta)$ cannot be computed efficiently.
- Variational inference approximates the posterior $p(z|x,\theta)$ with a simpler distribution $q(z)$.
- The variational objective is the **evidence lower bound (ELBO)**:

$$\mathbb{E}_{q(z)}\left[\log p(x,z|\theta)\right] + \mathbb{H}\left(q(z)\right)$$

- The solution to the ELBO minimises $\mathrm{KL}\left(q(z) \,||\, p(z|x,\theta)\right)$

There's an interesting special case of VI which is likely familiar to you. When $q(z) = p(z|x,\theta)$ we recover EM. Check the (optional) Appendix.

Summary

- We design $q(z)$ to be simple
- A common approximation is the **mean field** approximation which assumes that all latent variables are independent:

$$q(z|\lambda) = \prod_{d=1}^{D} q(z_d|\lambda_d)$$

- In amortised VI, we condition on a data point x to parameterise a collection of variational factors $\prod_{d=1}^{D} q(z_d|x, \lambda)$ and we typically use NNs for that.

Outline

Variational Inference Learning (NVIL)

Train a probabilistic model with NN likelihood using amortised variational inference.

$$\lambda^*, \theta^* = \arg\max_{\lambda,\theta} \mathbb{E}_{x\sim\mathcal{D}}\left[\underbrace{\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\log p(x,z|\theta)}{q(z|x,\lambda)}\right]}_{\text{ELBO}_x(\lambda,\theta)}\right]$$

Approach parameter estimation via stochastic gradient-based optimisation.

Now we discuss the concrete case of training deep discrete latent variable models with amortised variational inference.

The main difference with respect to VI as we saw is that we will be learning the inference model $q(z|x,\lambda)$ along with the joint distribution $p(x,z|\theta)$.

Concretely, we will use gradient-based optimisation to update λ and θ towards a (local) maximum of the ELBO.

The $\text{ELBO}_{\mathcal{D}}(\lambda,\theta)$, just like the log-likelihood function $\mathcal{L}_{\mathcal{D}}(\theta)$, factorises as a sum over i.i.d. observations.

Generative model

Again, let's take the latent factor document model as an example:

- a document $x = (x_1, \ldots, x_n)$ consists of n i.i.d. categorical draws from that model
- the categorical distribution in turn depends on binary latent factors $z = (z_1, \ldots, z_D)$ which are also i.i.d.

$$\begin{aligned} Z_d &\sim \text{Bernoulli}\,(\alpha) && (1 \leq d \leq D) \\ X_i|z &\sim \text{Categorical}\,(f(z;\theta)) && (1 \leq i \leq n) \end{aligned}$$

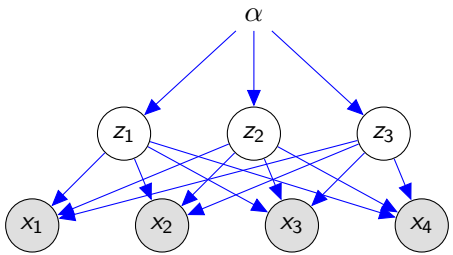
Here $0 < \alpha < 1$ specifies a Bernoulli prior (assume fixed) and $f(\cdot;\theta)$ is a function computed by an NN

$$\begin{aligned} f(z;\theta) &= \text{softmax}(Wz + b) \\ \theta &= \{W, b\} \end{aligned}$$

We've chosen a very shallow NN for the likelihood. It's just an affine projection and a softmax (a log-linear model).

Nothing prevents us from using a more complex likelihood, both in terms of parameterisation (e.g., a deeper FFNN) and statistical assumptions (e.g., a factorisation of the sequence x without Markov assumptions).

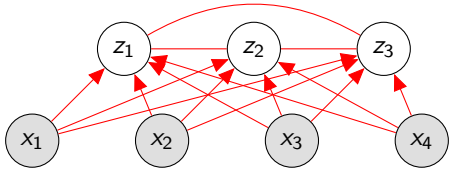
Example Model



Joint distribution: independent latent variables

I omit θ from the graphical model, but recall that every $X_i|\theta, z$ in the joint distribution depends on it. Moreover, every $Z_d|\theta, x$ in the true posterior distribution also depends on it.

Example Model

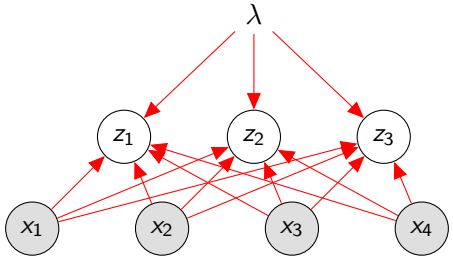


Posterior: latent variables are marginally dependent.

For our variational distribution we are going to assume that they are not (recall: mean field assumption).

I omit θ from the graphical model, but recall that every $X_i|\theta, z$ in the joint distribution depends on it. Moreover, every $Z_d|\theta, x$ in the true posterior distribution also depends on it.

Mean Field Inference



The inference network needs to predict D Bernoulli parameters b_1^D. Any neural network with sigmoid output will do that job.

The inference model is *independent* of θ.

That is the whole point, rather than actually inferring the true posterior, we want to independently estimate a model to perform approximate inference.

Inference Model

Model

$$q(z|x,\lambda) = \prod_{d=1}^{D} \text{Bern}(z_d|b_d)$$
$$\text{where } b_1^D = g(x;\lambda)$$

Example architecture (inference network)

$$h = \frac{1}{n}\sum_{i=1}^{n} E_{x_i} \qquad b_1^D = \text{sigmoid}(Mh + c)$$

$\lambda = \{E, M, c\}$

Some will call $q(z|x,\lambda)$ the inference network or recognition network. To be consistent with the vocabulary we've developed so far, I prefer calling the distribution $Z|X = x, \lambda$ an *inference model*. The *inference network* then is the NN architecture that parameterises the inference model. The term *recognition network* comes from the literature around Wake-Sleep (WS; Hinton et al., 1995), a heuristic form of VI that we discuss in the (optional) Appendix.

In this example, the inference network is very shallow: we embed the words using an embedding matrix E, combine them into an average h, project that to D real values via an affine transformation $Mh + c$, and use elementwise sigmoid to map each of those to he interval $(0, 1)$, necessary for the Bernoulli distributions. Nothing prevents us from using a more complex architecture, for example: we could encode the entire document using an LSTM and use the LSTM's last hidden state instead of the average of embeddings.

Making the inference model more complex, for example to correlate the latent assignments, is harder. But, if we knew a more complex model whose pmf is tractable (to assess and to sample from) we could use it instead. Can you think of any?

Objective

Let's concentrate on a single observation $x \in \mathcal{D}$:

$$\begin{aligned}\text{ELBO}_x(\lambda, \theta) &= \mathbb{E}_{q(z|x,\lambda)}\left[\log \frac{p(x, z|\theta)}{q(z|x, \lambda)}\right] \\ &= \mathbb{E}_{q(z|x,\lambda)}\left[\log p(x, z|\theta)\right] + \mathbb{H}\left(q(z|x, \lambda)\right) \\ &= \mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z, \theta)\right] - \text{KL}\left(q(z|x, \lambda) \,||\, p(z)\right)\end{aligned}$$

Parameter estimation

$$\arg\max_{\theta,\lambda} \; \mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z, \theta)\right] - \text{KL}\left(q(z|x, \lambda) \,||\, p(z)\right)$$

Here I list all 3 forms of the ELBO for a single data point. Generally, we need to pick a family for which we can sample from $Z|X = x, \lambda$ and assess the probability of a sample.

1. The first form is convenient when that is precisely all we can do.
2. The second form is convenient when in addition to that we can assess the entropy $\mathbb{H}(Z|X = x, \lambda)$.
3. The last form is convenient when $p(z)$ and $q(z|x, \lambda)$ are in the same exponential family.

Let's focus on the 3rd form for now, as it suffices to illustrate all challenges and the solutions we will develop. We have two terms, let's call them *the expected likelihood term* and the *KL term*, for brevity.

Our **goal** for the rest of this section is to compute ∇_θ and ∇_λ, or at least unbiased estimates thereof, as we need those for optimisation.

KL term

KL between D independent Bernoulli distributions is tractable

$$\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z|\alpha)\right) = \sum_{d=1}^{D} \mathrm{KL}\left(q(z_d|x,\lambda) \,||\, p(z_d|\alpha)\right)$$

$$= \sum_{d=1}^{D} \mathrm{KL}\left(\mathrm{Bernoulli}\left(b_d\right)\right) \,||\, \mathrm{Bernoulli}\left(\alpha\right))$$

$$= \sum_{d=1}^{D} b_d \log \frac{b_d}{\alpha} + (1-b_d) \log \frac{1-b_d}{1-\alpha}$$

In our example, the prior is a product of D independent Bernoulli distributions. Similarly, the inference model is a product of D independent distributions. This means that the KL term is a sum of D independent KL terms. Moreover, each $\mathrm{KL}\left(Z_d|X=x,\lambda \,||\, Z_d|\alpha\right)$ is known analytically, since both distributions are in the same exponential family (i.e., the Bernoulli family).

Being able to solve this expression in closed-form and with a computation that scales linearly in D means that there's no challenge in representing the KL term in a computation graph, and autodiff will be able to differentiate it with respect to λ (and even with respect to α should our prior not be fixed).

Updating the generative model

$$\frac{\partial}{\partial \theta}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z)\right)}^{\text{constant wrt } \theta}\right)$$

Updating the generative model is actually rather simple

- The second term is constant in this case, and poses no challenge. Even if it depend on θ, that is, if the prior depended on θ, as long as we can evaluate the KL term, autodiff would differentiate it for us. The first term seems less obvious, after all, we cannot solve the expected value in closed-form (it would take a sum over $z \in \mathcal{Z}$. Avoiding this sum is the whole point.

Updating the generative model

$$\frac{\partial}{\partial\theta}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda)\;||\;p(z)\right)}^{\text{constant wrt }\theta}\right)$$

$$= \underbrace{\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\theta}\log p(x|z,\theta)\right]}_{\text{expected gradient :)}}$$

Updating the generative model is actually rather simple

- The second term is constant in this case, and poses no challenge. Even if it depend on θ, that is, if the prior depended on θ, as long as we can evaluate the KL term, autodiff would differentiate it for us. The first term seems less obvious, after all, we cannot solve the expected value in closed-form (it would take a sum over $z \in \mathcal{Z}$. Avoiding this sum is the whole point.
- But note that the distribution we take expectations with respect to is the inference model $q(z|x,\lambda)$, which does not depend on θ. As derivatives are linear, we compute an expected derivative instead of differentiating an expected value.

Updating the generative model

$$\frac{\partial}{\partial\theta}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda)\,||\,p(z)\right)}^{\text{constant wrt }\theta}\right)$$

$$= \underbrace{\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\theta}\log p(x|z,\theta)\right]}_{\text{expected gradient :)}}$$

$$\overset{\text{MC}}{\approx} \frac{1}{S}\sum_{s=1}^{S}\frac{\partial}{\partial\theta}\log p(x|z^{(s)},\theta) \quad \text{where } z^{(s)} \sim q(z|x,\lambda)$$

Updating the generative model is actually rather simple

- The second term is constant in this case, and poses no challenge. Even if it depend on θ, that is, if the prior depended on θ, as long as we can evaluate the KL term, autodiff would differentiate it for us. The first term seems less obvious, after all, we cannot solve the expected value in closed-form (it would take a sum over $z \in \mathcal{Z}$. Avoiding this sum is the whole point.
- But note that the distribution we take expectations with respect to is the inference model $q(z|x,\lambda)$, which does not depend on θ. As derivatives are linear, we compute an expected derivative instead of differentiating an expected value.
- Expected values are great for we know how to estimate them without bias. More often than not we use a single sample per observation.

Updating the generative model

$$\frac{\partial}{\partial\theta}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda)\,||\,p(z)\right)}^{\text{constant wrt }\theta}\right)$$

$$= \underbrace{\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\theta}\log p(x|z,\theta)\right]}_{\text{expected gradient :)}}$$

$$\overset{\text{MC}}{\approx} \frac{1}{S}\sum_{s=1}^{S}\frac{\partial}{\partial\theta}\log p(x|z^{(s)},\theta) \quad \text{where } z^{(s)} \sim q(z|x,\lambda)$$

Monte Carlo (MC) estimation gives us a gradient estimate with a computation that does not depend on the size of $\mathcal{Z}$.

Updating the generative model is actually rather simple

- The second term is constant in this case, and poses no challenge. Even if it depend on θ, that is, if the prior depended on θ, as long as we can evaluate the KL term, autodiff would differentiate it for us. The first term seems less obvious, after all, we cannot solve the expected value in closed-form (it would take a sum over $z \in \mathcal{Z}$. Avoiding this sum is the whole point.
- But note that the distribution we take expectations with respect to is the inference model $q(z|x,\lambda)$, which does not depend on θ. As derivatives are linear, we compute an expected derivative instead of differentiating an expected value.
- Expected values are great for we know how to estimate them without bias. More often than not we use a single sample per observation.

Updating the inference model

$$\frac{\partial}{\partial \lambda}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z)\right)}^{\text{analytical}}\right)$$

Updating the inference model is not as simple

- The KL term is tractable to assess, thus autodiff will handle it, and we don't need to worry about the exact form of the gradient.

Updating the inference model

$$\frac{\partial}{\partial \lambda}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z)\right)}^{\text{analytical}}\right)$$

$$= \frac{\partial}{\partial \lambda}\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \underbrace{\frac{\partial}{\partial \lambda}\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z)\right)}_{\text{analytical computation}}$$

Updating the inference model is not as simple

- The KL term is tractable to assess, thus autodiff will handle it, and we don't need to worry about the exact form of the gradient.
- The first term requires an intractable sum over $z \in \mathcal{Z}$ which we mean to avoid. Unfortunately this time we cannot simply 'push' the derivative inside as the expectation is taken w.r.t. $q(z|x,\lambda)$, which clearly depends on λ.

Updating the inference model

$$\frac{\partial}{\partial \lambda}\left(\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \overbrace{\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z)\right)}^{\text{analytical}}\right)$$

$$= \frac{\partial}{\partial \lambda}\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] - \underbrace{\frac{\partial}{\partial \lambda}\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z)\right)}_{\text{analytical computation}}$$

The first term again requires approximation by sampling, but there is a problem

Updating the inference model is not as simple

- The KL term is tractable to assess, thus autodiff will handle it, and we don't need to worry about the exact form of the gradient.
- The first term requires an intractable sum over $z \in \mathcal{Z}$ which we mean to avoid. Unfortunately this time we cannot simply 'push' the derivative inside as the expectation is taken w.r.t. $q(z|x,\lambda)$, which clearly depends on λ.

MC is not differentiable

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q_\lambda(z|x)} \left[\log p_\theta(x|z) \right]$$
$$= \frac{\partial}{\partial \lambda} \sum_z q(z|x, \lambda) \log p(x|z, \theta)$$

Unfortunately, we cannot turn to MC either, as we can only MC estimate expected values, and the derivative of the expected likelihood term does not seem to be an expected value.

MC is not differentiable

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q_\lambda(z|x)} \left[\log p_\theta(x|z) \right]$$
$$= \frac{\partial}{\partial \lambda} \sum_z q(z|x,\lambda) \log p(x|z,\theta)$$
$$= \underbrace{\sum_z \frac{\partial}{\partial \lambda} (q(z|x,\lambda)) \log p(x|z,\theta)}_{\text{not an expectation}}$$

Unfortunately, we cannot turn to MC either, as we can only MC estimate expected values, and the derivative of the expected likelihood term does not seem to be an expected value.

- Writing the expected likelihood explicitly we can see that though we can sum derivatives, as differentiation is linear, we cannot hope to evaluate all $|\mathcal{Z}|$ terms in our lifetime.

MC is not differentiable

$$\begin{aligned}
&\frac{\partial}{\partial \lambda} \mathbb{E}_{q_\lambda(z|x)} \left[\log p_\theta(x|z)\right] \\
&= \frac{\partial}{\partial \lambda} \sum_z q(z|x,\lambda) \log p(x|z,\theta) \\
&= \underbrace{\sum_z \frac{\partial}{\partial \lambda} (q(z|x,\lambda)) \log p(x|z,\theta)}_{\text{not an expectation}}
\end{aligned}$$

- MC estimator is non-differentiable

Unfortunately, we cannot turn to MC either, as we can only MC estimate expected values, and the derivative of the expected likelihood term does not seem to be an expected value.

- Writing the expected likelihood explicitly we can see that though we can sum derivatives, as differentiation is linear, we cannot hope to evaluate all $|\mathcal{Z}|$ terms in our lifetime.
- This shows that in general we cannot differentiate an MC estimate.

MC is not differentiable

$$\begin{aligned}
&\frac{\partial}{\partial \lambda} \mathbb{E}_{q_\lambda(z|x)} \left[\log p_\theta(x|z)\right] \\
&= \frac{\partial}{\partial \lambda} \sum_z q(z|x,\lambda) \log p(x|z,\theta) \\
&= \underbrace{\sum_z \frac{\partial}{\partial \lambda} (q(z|x,\lambda)) \log p(x|z,\theta)}_{\text{not an expectation}}
\end{aligned}$$

- MC estimator is non-differentiable
- Differentiating the expression does not yield an expectation: cannot approximate via MC

Unfortunately, we cannot turn to MC either, as we can only MC estimate expected values, and the derivative of the expected likelihood term does not seem to be an expected value.

- Writing the expected likelihood explicitly we can see that though we can sum derivatives, as differentiation is linear, we cannot hope to evaluate all $|\mathcal{Z}|$ terms in our lifetime.
- This shows that in general we cannot differentiate an MC estimate.

Score Function Estimator

We can again use the log identity for derivatives

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q_\lambda(z|x)} [\log p_\theta(x|z)]$$
$$= \sum_z \frac{\partial}{\partial \lambda} (q(z|x, \lambda)) \log p(x|z, \theta)$$

It turns out we've already seen this form of gradient when we derived the general form of $\nabla_\theta \log p(x|\theta)$ for models with tractable marginals.

Score Function Estimator

We can again use the log identity for derivatives

$$\begin{aligned}
&\frac{\partial}{\partial \lambda} \mathbb{E}_{q_\lambda(z|x)} \left[\log p_\theta(x|z)\right] \\
&= \sum_z \frac{\partial}{\partial \lambda} (q(z|x,\lambda)) \log p(x|z,\theta) \\
&= \sum_z q(z|x,\lambda) \frac{\partial}{\partial \lambda} (\log q(z|x,\lambda)) \log p(x|z,\theta)
\end{aligned}$$

It turns out we've already seen this form of gradient when we derived the general form of $\nabla_\theta \log p(x|\theta)$ for models with tractable marginals.

- We can use the log identity for derivatives (i.e., $f' = f(\log f)'$) to re-express the sum as an expectation with respect to $q(z|x,\lambda)$.

Score Function Estimator

We can again use the log identity for derivatives

$$\begin{aligned}
&\frac{\partial}{\partial\lambda}\mathbb{E}_{q_\lambda(z|x)}\left[\log p_\theta(x|z)\right] \\
&= \sum_z \frac{\partial}{\partial\lambda}(q(z|x,\lambda))\log p(x|z,\theta) \\
&= \sum_z q(z|x,\lambda)\frac{\partial}{\partial\lambda}(\log q(z|x,\lambda))\log p(x|z,\theta) \\
&= \underbrace{\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right]}_{\text{expected gradient :)}}
\end{aligned}$$

It turns out we've already seen this form of gradient when we derived the general form of $\nabla_\theta \log p(x|\theta)$ for models with tractable marginals.

- We can use the log identity for derivatives (i.e., $f' = f(\log f)'$) to re-express the sum as an expectation with respect to $q(z|x,\lambda)$.
- This estimator is known as the *score function estimator*.

Score Function Estimator

We can again use the log identity for derivatives

$$\begin{aligned}
&\frac{\partial}{\partial\lambda}\mathbb{E}_{q_\lambda(z|x)}\left[\log p_\theta(x|z)\right] \\
&= \sum_z \frac{\partial}{\partial\lambda}(q(z|x,\lambda))\log p(x|z,\theta) \\
&= \sum_z q(z|x,\lambda)\frac{\partial}{\partial\lambda}(\log q(z|x,\lambda))\log p(x|z,\theta) \\
&= \underbrace{\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right]}_{\text{expected gradient :)}}
\end{aligned}$$

We turned the derivative of an expectation into the expected value of a derivative!

It turns out we've already seen this form of gradient when we derived the general form of $\nabla_\theta \log p(x|\theta)$ for models with tractable marginals.

- We can use the log identity for derivatives (i.e., $f' = f(\log f)'$) to re-express the sum as an expectation with respect to $q(z|x,\lambda)$.
- This estimator is known as the *score function estimator*.

Score Function Estimator

We can now build an MC estimator

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q(z|x,\lambda)} \left[\log p(x|z,\theta) \right]$$
$$= \mathbb{E}_{q(z|x,\lambda)} \left[\log p(x|z,\theta) \frac{\partial}{\partial \lambda} \log q(z|x,\lambda) \right]$$

And, as always, expected gradients can be estimated free of bias via MC.

Score Function Estimator

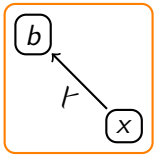
We can now build an MC estimator

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q(z|x,\lambda)} \left[\log p(x|z,\theta) \right]$$
$$= \mathbb{E}_{q(z|x,\lambda)} \left[\log p(x|z,\theta) \frac{\partial}{\partial \lambda} \log q(z|x,\lambda) \right]$$
$$\overset{\text{MC}}{\approx} \frac{1}{S} \sum_{s=1}^{S} \log p(x|z^{(s)},\theta) \frac{\partial}{\partial \lambda} \log q(z^{(s)}|x,\lambda)$$

where $z^{(s)} \sim q(z|x,\lambda)$

And, as always, expected gradients can be estimated free of bias via MC.

Computation Graph



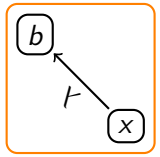
inference network

Let's put everything together in a computation graph

- we map an observation x to the parameters b of our inference model, this uses an NN with parameters λ;

Computation Graph

$z \sim \text{Bernoulli}(b)$

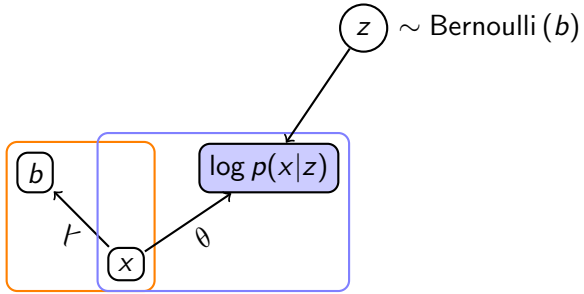


inference network

Let's put everything together in a computation graph

- we map an observation x to the parameters b of our inference model, this uses an NN with parameters λ;
- with b we can parameterise Bernoulli distributions (in our example), from which we know how to obtain independent samples;

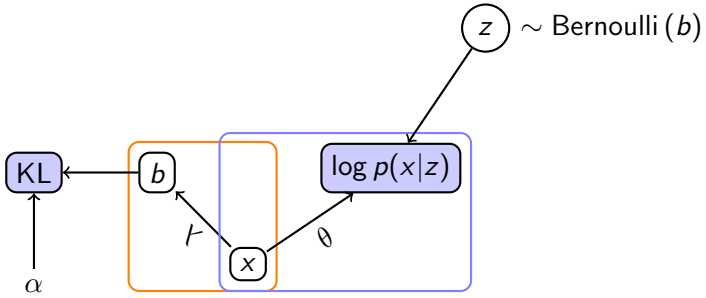
Computation Graph



Let's put everything together in a computation graph

- we map an observation x to the parameters b of our inference model, this uses an NN with parameters λ;
- with b we can parameterise Bernoulli distributions (in our example), from which we know how to obtain independent samples;
- besides, we have our main neural network, which maps from z to the log-probability $\log p(x|z, \theta)$, this is a quantity that depends on θ and whose gradient we need in order to update the generative model;

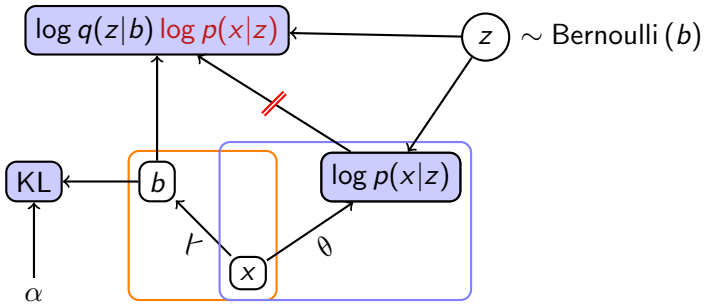
Computation Graph



Let's put everything together in a computation graph

- we map an observation x to the parameters b of our inference model, this uses an NN with parameters λ;
- with b we can parameterise Bernoulli distributions (in our example), from which we know how to obtain independent samples;
- besides, we have our main neural network, which maps from z to the log-probability $\log p(x|z, \theta)$, this is a quantity that depends on θ and whose gradient we need in order to update the generative model;
- with b and the prior parameter α, we can assess $\mathrm{KL}\left(q(z|x, \lambda) \,||\, p(z|\alpha)\right)$, whose gradient we need in order to update the inference model;

Computation Graph



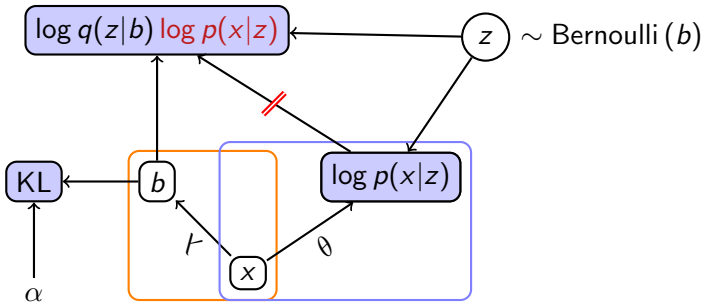
inference network

generative network

Let's put everything together in a computation graph

- we map an observation x to the parameters b of our inference model, this uses an NN with parameters λ;
- with b we can parameterise Bernoulli distributions (in our example), from which we know how to obtain independent samples;
- besides, we have our main neural network, which maps from z to the log-probability $\log p(x|z, \theta)$, this is a quantity that depends on θ and whose gradient we need in order to update the generative model;
- with b and the prior parameter α, we can assess $\mathrm{KL}\left(q(z|x, \lambda) \,||\, p(z|\alpha)\right)$, whose gradient we need in order to update the inference model;
- finally, to update the inference model we also need the score function estimator, which is $\log p(x|z, \theta) \nabla_\lambda \log q(z|x, \lambda)$; to obtain that gradient using autodiff we need to get a gradient for $\log q(z|x, \lambda)$ and scale it by the log-likelihood $\log p(x|z, \theta)$; we can do that if we can achieve that by multiplying $\log q(z|x, \lambda)$ and a 'detached' (constant) version of $\log p(x|z, \theta)$;

Computation Graph



inference network

generative network

Let's put everything together in a computation graph

- we map an observation x to the parameters b of our inference model, this uses an NN with parameters λ;
- with b we can parameterise Bernoulli distributions (in our example), from which we know how to obtain independent samples;
- besides, we have our main neural network, which maps from z to the log-probability $\log p(x|z, \theta)$, this is a quantity that depends on θ and whose gradient we need in order to update the generative model;
- with b and the prior parameter α, we can assess $\mathrm{KL}\left(q(z|x, \lambda) \,||\, p(z|\alpha)\right)$, whose gradient we need in order to update the inference model;
- finally, to update the inference model we also need the score function estimator, which is $\log p(x|z, \theta) \nabla_\lambda \log q(z|x, \lambda)$; to obtain that gradient using autodiff we need to get a gradient for $\log q(z|x, \lambda)$ and scale it by the log-likelihood $\log p(x|z, \theta)$; we can do that if we can achieve that by multiplying $\log q(z|x, \lambda)$ and a 'detached' (constant) version of $\log p(x|z, \theta)$;

Stochastic surrogate objectives

A computation node whose gradient estimates the gradient we want:

$$\log p(x|z,\theta) - \mathrm{KL}\left(q(z|x,\lambda \mid\mid p(z|\alpha)\right) + \underbrace{\log p(x|z,\not{\theta})}_{\text{'detached'}} \log q(z|x,\lambda)$$

Can you verify $\nabla_{\theta,\lambda}$ of the surrogate objective yields the correct partials?

Implementation goal: we want a forward pass whose backward estimates $\nabla_{\lambda,\theta} \mathrm{ELBO}_x(\lambda,\theta)$.

That is, a quantity whose gradient w.r.t. λ, θ as computed by an automatic differentiation algorithm yields the correct partial derivatives for the generative and the inference model.

To implement this efficiently, we resort to the notion of a 'detached' computation node. That is, a node whose value is interpreted as a constant (its outputs are disconnected from NN parameters during back-propagation). For brevity, we will denote this by crossing the parameter out (e.g., $\not{\theta}$).

Score Function Estimator: Variance

$$\frac{\partial}{\partial \lambda} \mathbb{E}_{q(z|x,\lambda)} \left[\log p(x|z,\theta) \right] = \mathbb{E}_{q(z|x,\lambda)} \left[\log p(x|z,\theta) \frac{\partial}{\partial \lambda} \log q(z|x,\lambda) \right]$$

Empirically this estimator often exhibits high variance.

- the magnitude of $\log p(x|z,\theta)$ varies widely
- the model likelihood does not contribute to direction of gradient (it only scales the gradient)

We can get gradient estimates and they are unbiased, but they are too noisy to be useful out of the box.

How can we reduce the variance of an estimator?

Score Function Estimator: Variance

We could:

- sample more (better MC estimates)
- use variance reduction techniques (e.g. baselines and control variates)

Sampling more is not a very efficient way to reduce variance, as the variance drops with the square root of the number of samples.

Perhaps we can do better with less computation?

Score Function Estimator: Variance

Idea: standardise the "reward" $r(z) := \log p(x|z,\theta)$ to have a mean at 0 and a variance of 1

- Keep a moving average of the mean and variance $\log p(x|z,\theta)$: $\hat{\mu}$ and $\hat{\sigma^2}$.
- $\hat{r}(z) = \frac{\log p(x|z,\theta) - \hat{\mu}}{\hat{\sigma^2}}$

It can be shown that

$$\begin{aligned}\frac{\partial}{\partial\lambda}\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] &= \mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right] \\ &= \mathbb{E}_{q(z|x,\lambda)}\left[\hat{r}(z)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right]\end{aligned}$$

To understand why this is true, we need to learn more about control variates (Greensmith et al., 2004). You can see the (optional) Appendix.

In reinforcement learning, $\hat{\mu}$ is also known as a baseline. Score function estimation along with baselines is what is known as REINFORCE (Williams, 1992).

Score Function Estimator: Variance

- We can show that using these *baselines* does not bias the estimator.
- We can add more advanced *control variates* and other *baselines* to further reduce variance.
- More about this in the (optional) Appendix.

Back to the KL term

We can easily relax our constraints about the tractability of the KL term. In general, we could have the approximate posterior and the prior in different families, and the prior could even depend on θ.

Recall that

$$\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z|\theta)\right) = \mathbb{E}_{q(z|x,\lambda)}\left[\log \frac{q(z|x,\lambda)}{p(z|\theta)}\right]$$

If this quantity is not tractable we can work with gradient estimates of it:

$$\nabla_\theta \,\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z|\theta)\right) = \mathbb{E}_{q(z|x,\lambda)}\left[-\nabla_\theta \log p(z|\theta)\right]$$

$$\nabla_\lambda \,\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z|\theta)\right) = \mathbb{E}_{q(z|x,\lambda)}\left[\log \frac{q(z|x,\lambda)}{p(z|\theta)} \nabla_\lambda \log q(z|x,\lambda)\right]$$

By rewriting the KL term as an expectation we can see that its gradient w.r.t. θ is indeed the expected value of a gradient, which we can MC-estimate directly.

For the gradient w.r.t. λ, we again need to use the score function estimator, which re-expressed the gradient as an expected value, for which then MC estimation is possible.

It is an interesting exercise to show to yourself that the expression for $\nabla_\lambda \,\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z|\theta)\right)$ indeed holds.

Pros and Cons

Pros:

- Applicable to all distributions
- Many libraries come with samplers for common distributions

Cons:

- High Variance!

Unfortunately, for discrete latent variables there is not alternative. Combating the cons takes studying and deploying variance reduction techniques such as control variates (Gu et al., 2016; Tucker et al., 2017; Grathwohl et al., 2018), Rao-Blackwellization (Liu et al., 2019), as well as other techniques developed in reinforcement learning literature (Rennie et al., 2017; Schulman et al., 2017).

Mohamed et al. (2019) present an extensive survey.

NVIL's original paper (Mnih and Gregor, 2014). The same ideas power black-box inference outside the context of deep learning (Ranganath et al., 2014). Mnih and Rezende (2016) present an extension based on multiple-sample MC estimates.

What next?

1. check the quizzes on Canvas
2. check the list of exercises
3. come to the live session with questions

Next week we talk about deep latent variable models with continuous random variables.

Outline

Implicit distributions

We can specify a stochastic map by using a (deterministic) NN and a source of random numbers with probability density function $s(\epsilon)$. For each (x, ϵ) the mapping is deterministic, but the noise source induces a random variable $Y|\theta, x$. The **implicit likelihood** assigned to an outcome y given x is $p(y|x, \theta) = \int_{\{\epsilon: f(x,\epsilon;\theta)=y\}} s(\epsilon)\mathrm{d}\epsilon$.

In words, we must 'integrate the density of the noise source for every possible way you can map x to y.'

KL divergence

The Kullback-Leibler divergence (or relative entropy) measures the divergence of a distribution q from a distribution p.

- $\mathrm{KL}\left(q(z) \,||\, p(z)\right) = \mathbb{E}_{q(z)}\left[\log \frac{q(z)}{p(z)}\right]$
- $\mathrm{KL}\left(q(z) \,||\, p(z)\right) = \int q(z) \log \frac{q(z)}{p(z)} \mathrm{d}z$ (continuous)
- $\mathrm{KL}\left(q(z) \,||\, p(z)\right) = \sum_z q(z) \log \frac{q(z)}{p(z)}$ (discrete)

KL divergence - Properties

Properties

- $\mathrm{KL}\left(q(z) \,||\, p(z)\right) \geq 0$ with equality iff $q(z) = p(z)$.
- $-\mathrm{KL}\left(q(z) \,||\, p(z)\right) = \mathbb{E}_{q(z)}\left[\log \frac{p(z)}{q(z)}\right] \leq 0.$
- We want: $supp(q) \subseteq supp(p)$; otherwise $\mathrm{KL}\left(q(z) \,||\, p(z)\right) = \infty$

Wake-Sleep Algorithm

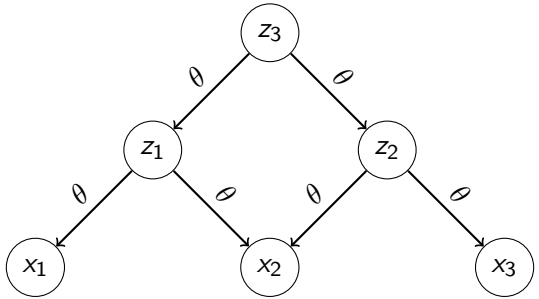
- Generalise latent variables to neural networks.
- Train generative neural model.
- Use variational inference! (kind of)
- Hinton et al. (1995)

Wake-Sleep Architecture

2 neural networks:

- A generation network to model the data (the one we want to optimise) – parameters: θ
- An inference (recognition) network (to model the latent variable) – parameters: λ
- Original setting: binary hidden units
- Training is performed in a "hard EM" fashion

Generator



The 'generator' in wake-sleep is a generative model parameterised by NNs. In the original paper they had an NN with stochastic binary hidden units.

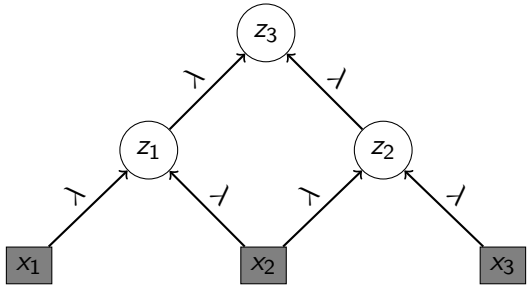
For example, this NN has 3 layers:

- The top one parameterises a distribution over 1 binary random variable, i.e., $Z_3|\theta_0 \sim \text{Bern}(f^{(3)}(\theta_3))$.
- The middle one conditions on a sampled z_3 and parameterises a distribution over 2 binary random variables, i.e., $Z_d|\theta, Z = z_3 \sim \text{Bern}(f_d^{(2)}(z_3; \theta_2))$ for $d = 1, 2$.
- The bottom one conditions on sampled $\langle z_1, z_2 \rangle$ and parameterises a distribution over 3 observed random variables. For example, if x is a document we might make an independence assumption: $X_i|\theta, Z_1 = z_1, Z_2 = z_2 \sim \text{Cat}(f^{(1)}(z_1, z_2; \theta_1))$.

The true posterior is clearly intractable, it takes assessing $p(x|\theta) = \sum_{z \in \mathcal{Z}} p(x, z|\theta)$ and $\mathcal{Z}$ is the space of all possible configuration of binary assignments.

I omit arrows from z_2 to x_1 and from z_1 to x_3 to keep the drawing cleaner.

Recognition Network



The recognition network is much like our inference models. It predicts a distribution over Z_1, Z_2, Z_3 given x using an independent model with parameters λ.

This is an NN that predicts as many rvs as there are latent variables in the original model. Think of it as a conditional model of the latent variable.

- We condition on x and parameterise a distribution over two binary random variables, i.e.: $Z_d|\lambda, x \sim \text{Bern}(g^{(1)}(x; \lambda_1))$ for $d = 1, 2$.
- We then condition on sampled $\langle z_1, z_2 \rangle$ and parameterise a distribution over one binary random variable $Z_3|\lambda z_1, z_2 \sim \text{Bern}(g^{(2)}(z_1, z_2; \lambda_2))$

The recognition network specifies an approximate posterior distribution which assumes layer-wise independence, that is, $Z_d^{(\ell)}$ in a layer ℓ is independent on all but the latent variables in the layer below.

I omit arrows from x_2 to z_2 and from x_3 to z_1 to keep the drawing cleaner.

Wake-sleep Training

Wake Phase

- Use inference network to sample hidden unit setting z from $q(z|x, \lambda)$
- Update generation parameters θ to maximize joint log-likelihood of data and latents $p(x, z|\theta)$

Wake-sleep Training

Wake Phase

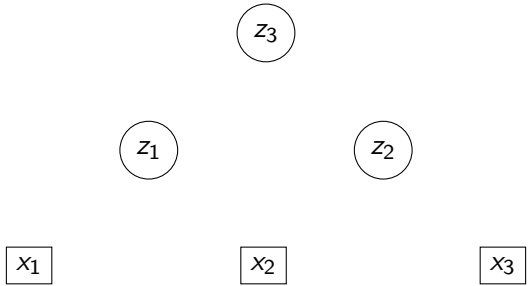
- Use inference network to sample hidden unit setting z from $q(z|x, \lambda)$
- Update generation parameters θ to maximize joint log-likelihood of data and latents $p(x, z|\theta)$

Sleep Phase

- Produce dream sample $z, \tilde{x}$ from the joint distribution
- Update inference parameters λ to maximize probability of latent state $q(z|\tilde{x}, \lambda)$

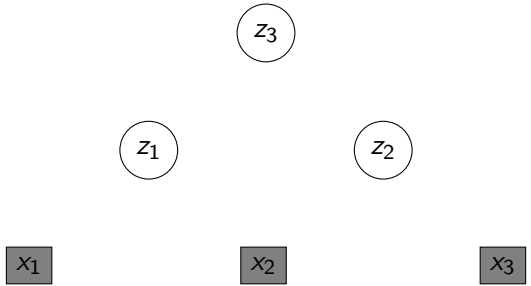
Wake Phase Sampling

Sampling $z \sim q(z|x, \lambda)$



Wake Phase Sampling

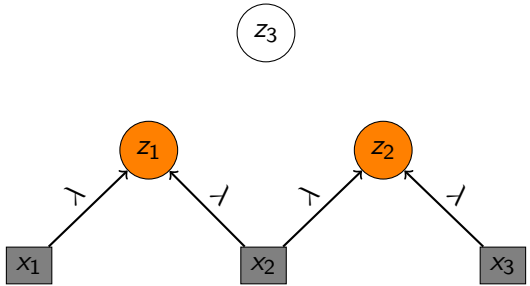
Sampling $z \sim q(z|x, \lambda)$



- Observe x

Wake Phase Sampling

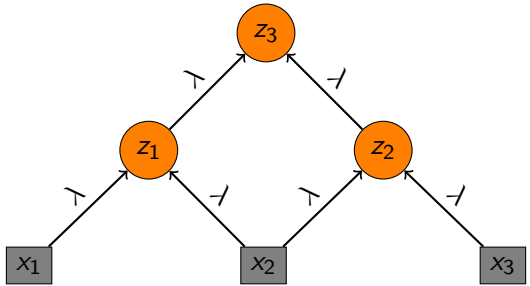
Sampling $z \sim q(z|x, \lambda)$



- Observe x
- Parameterise distributions $Z_d|\theta, X = x$ and sample latent variables z_1, z_2

Wake Phase Sampling

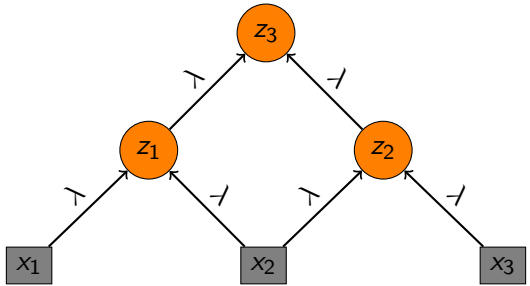
Sampling $z \sim q(z|x, \lambda)$



- Observe x
- Parameterise distributions $Z_d|\theta, X = x$ and sample latent variables z_1, z_2
- Condition on z_1, z_2, parameterise distribution $Z_3|\theta, Z_1 = z_1, Z_2 = z_2$ and sample latent variable z_3.

Wake Phase Sampling

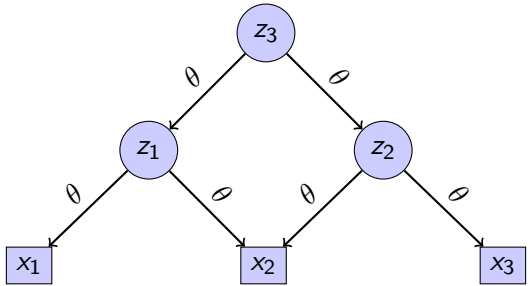
Sampling $z \sim q(z|x, \lambda)$



- Observe x
- Parameterise distributions $Z_d|\theta, X = x$ and sample latent variables z_1, z_2
- Condition on z_1, z_2, parameterise distribution $Z_3|\theta, Z_1 = z_1, Z_2 = z_2$ and sample latent variable z_3.

Wake Phase Update

Compute $\log p(x, z|\theta)$ and update θ



With the sample z we got from the recognition network we can compute the joint probability of z and the observation x. This means we do not need to sample from $p(x, z|\theta)$. The alternative to sampling from the recognition model, would be to fix the observation x and sample from the induced true posterior $p(z|x, \theta)$, which is clearly intractable.

Thus the recognition model plays a role identical to that of the inference model in variational inference.

As in VI, because we sampled from $q(z|x, \lambda)$ it is easy to compute a gradient estimate w.r.t. θ.

The situation is much more difficult w.r.t. λ, as we saw in the section about NVIL. To circumvent difficulties with gradient estimation for λ, in Wake-Sleep, we change the optimisation objective in order to update the recognition model. In particular, we update the recognition model as to maximise the probability of some 'dream data' which we obtain by sampling from the generative model.

Sleep Phase Sampling

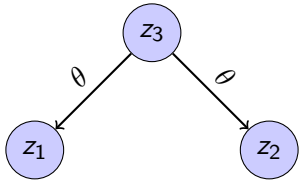
Sampling $(z, \tilde{x}) \sim p(x, z|\theta)$

We do a stochastic forward pass through the generative model sampling our random variables.

- We sample z_3 from the distribution at the top layer.

Sleep Phase Sampling

Sampling $(z, \tilde{x}) \sim p(x, z|\theta)$

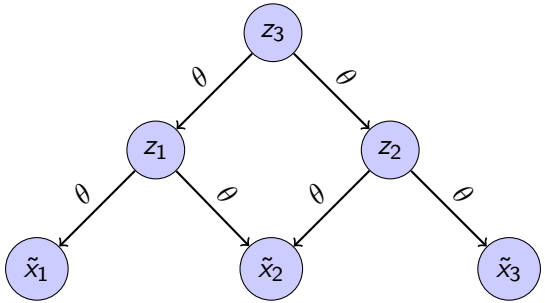


We do a stochastic forward pass through the generative model sampling our random variables.

- We sample z_3 from the distribution at the top layer.
- Then condition on z_3 to parameterise the distribution $Z_1, Z_2|\theta, Z_3 = z_3$, from which we sample z_1 and z_2.

Sleep Phase Sampling

Sampling $(z, \tilde{x}) \sim p(x, z|\theta)$

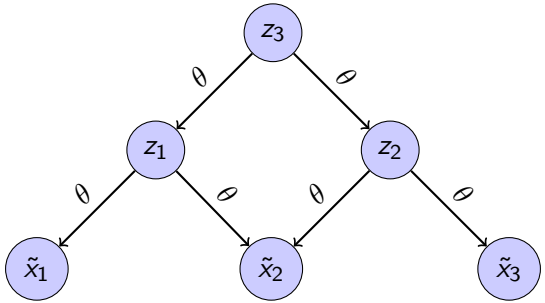


We do a stochastic forward pass through the generative model sampling our random variables.

- We sample z_3 from the distribution at the top layer.
- Then condition on z_3 to parameterise the distribution $Z_1, Z_2|\theta, Z_3 = z_3$, from which we sample z_1 and z_2.
- We condition on z_1 and z_2 to parameterise our output distributions over data space $X_i|\theta, Z_1 = z_1, Z_2 = z_2$, from where we **sample** data. This is crucial, our sample $\tilde{x}$ is not an actual observation (we mark it with tilde to help you track its influence).

Sleep Phase Sampling

Sampling $(z, \tilde{x}) \sim p(x, z|\theta)$

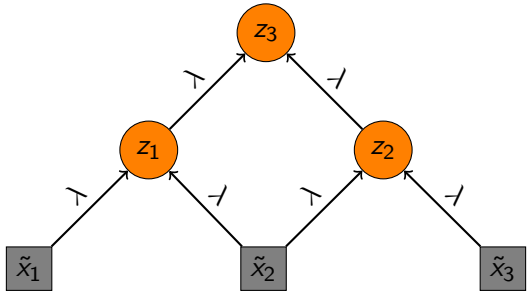


We do a stochastic forward pass through the generative model sampling our random variables.

- We sample z_3 from the distribution at the top layer.
- Then condition on z_3 to parameterise the distribution $Z_1, Z_2|\theta, Z_3 = z_3$, from which we sample z_1 and z_2.
- We condition on z_1 and z_2 to parameterise our output distributions over data space $X_i|\theta, Z_1 = z_1, Z_2 = z_2$, from where we **sample** data. This is crucial, our sample $\tilde{x}$ is not an actual observation (we mark it with tilde to help you track its influence).

Sleep Phase Update

Compute $\log q(z|\tilde{x}, \lambda)$ and update λ



The last ingredient is to assess the likelihood of the sampled z given the sampled $\tilde{x}$ under the recognition model and update λ as to maximise it.

Wake Phase Objective

Objective

$$\arg\min_{\theta} \ \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(q(z|x,\lambda) \,||\, p(z|x,\theta)\right)\right]$$

$$= \arg\max_{\theta} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{ELBO}_x(\theta,\lambda) - \log p(x|\theta)\right]$$

Approximation: optimize the lower-bound alone.

The wake-phase really is identical to VI. It makes the exact same approximation, namely, that optimising a lowerbound on the log-evidence is a good idea.

Wake Phase Objective

Objective

$$\arg\max_{\theta} \; \mathbb{E}_{x\sim\mathcal{D}}\left[\text{ELBO}_x(\theta,\lambda)\right]$$
$$= \arg\max_{\theta} \; \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{q(z|x,\lambda)}\left[\log p(z,x|\theta)\right] + \mathbb{H}[q(z|x,\lambda)]\right]$$

Gradient wrt θ for $x \sim \mathcal{D}$ (an observation)

$$\nabla_\theta \mathbb{E}_{q(z|x,\lambda)}\left[\log p(z,x|\theta)\right] + \nabla_\theta \mathbb{H}[q(z|x,\lambda)]$$
$$= \mathbb{E}_{q(z|x,\lambda)}\left[\nabla_\theta \log p(z,x|\theta)\right]$$
$$\overset{\text{MC}}{\approx} \nabla_\theta \log p(z,x|\theta) \quad \text{where } z \sim q(z|x,\lambda)$$

The gradient of the entropy term is 0 and the first term corresponds to the expected value of a stochastic gradient, thus MC gives us the unbiased estimate we need for optimisation of the generative model.

In this phase z if fixed to a random draw from $q(z|x,\lambda)$, from the point of view of the generative model it is as if z had been observed, so we can maximise $\log p(z,x|\theta)$.

This is simply supervised learning with imputed latent data!

Sleep Phase Objective

Objective

$$\arg\max_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\text{ELBO}_x(\theta,\lambda)\right]$$
$$= \arg\max_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{q(z|x,\lambda)}\left[\log p(z,x|\theta)\right] + \mathbb{H}[q(z|x,\lambda)]\right]$$

Gradient wrt λ for $x \sim \mathcal{D}$ (an observation)

$$\nabla_\lambda \mathbb{E}_{q(z|x,\lambda)}\left[\log p(z,x|\theta)\right] + \nabla_\lambda \mathbb{H}[q(z|x,\lambda)]$$

Let's change the objective!

When we turn to the gradient of the recognition model, as expected, things are not as easy.

Of course we know that we can re-express both gradients (recall that the entropy term is also an expected value) as expected gradients via the score function method. That's not how WS goes about this problem. Instead, WS changes the objective of optimisation.

This means that for the sleep phase, where we are supposed to learn the recognition model, we are not going to do VI. This is indeed a pity, since maximising the ELBO w.r.t. our choice of λ indeed minimises $\text{KL}\left(q(z|x,\lambda) \,||\, p(z|x,\theta)\right)$.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

$$\arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}} \left[\mathrm{KL}\left(p(z|x,\theta) \;||\; q(z|x,\lambda)\right)\right]$$

The strategy for the sleep phase is to flip the KL around, that is, to assess the KL divergence of $p(z|x,\theta)$ from $q(z|x,\lambda)$.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

$$\begin{aligned}&\arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(p(z|x,\theta) \,||\, q(z|x,\lambda)\right)\right] \\ &= \arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{p(z|x,\theta)}\left[\log p(z|x,\theta) - \log q(z|x,\lambda)\right]\end{aligned}$$

The strategy for the sleep phase is to flip the KL around, that is, to assess the KL divergence of $p(z|x,\theta)$ from $q(z|x,\lambda)$.

- See that this change is in some sense convenient. Assume we are able to sample from the true posterior, then we can get gradient estimates w.r.t. λ. Clearly this is only superficially simple, as we have no means to sample from the true posterior.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

$$\begin{aligned}
&\arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(p(z|x,\theta) \;||\; q(z|x,\lambda)\right)\right] \\
&= \arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{p(z|x,\theta)}\left[\log p(z|x,\theta) - \log q(z|x,\lambda)\right] \\
&\overset{\text{asm}}{=} \arg\max_{\lambda} \mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] - \underbrace{\mathbb{E}_{p(x,z|\theta)}\left[\log p(z|x,\theta)\right]}_{\text{constant}}
\end{aligned}$$

The strategy for the sleep phase is to flip the KL around, that is, to assess the KL divergence of $p(z|x,\theta)$ from $q(z|x,\lambda)$.

- See that this change is in some sense convenient. Assume we are able to sample from the true posterior, then we can get gradient estimates w.r.t. λ. Clearly this is only superficially simple, as we have no means to sample from the true posterior.
- Here is where WS makes a big assumption, it assumes that sampling from the data $x \sim \mathcal{D}$ is equivalent to sampling from the marginal of the model $x \sim p(x|\theta)$, this can only be true if our model perfectly reproduces the data generating process. This is very unlikely in general, since the data generating process is unknown to us, and it's particularly unlikely at the beginning of training.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

$$\begin{aligned}
&\arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(p(z|x,\theta) \,||\, q(z|x,\lambda)\right)\right] \\
&= \arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{p(z|x,\theta)}\left[\log p(z|x,\theta) - \log q(z|x,\lambda)\right] \\
&\overset{\text{asm}}{=} \arg\max_{\lambda} \mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] - \underbrace{\mathbb{E}_{p(x,z|\theta)}\left[\log p(z|x,\theta)\right]}_{\text{constant}}
\end{aligned}$$

Gradient wrt λ

$$\begin{aligned}
&\nabla_{\lambda}\mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] \\
&= \mathbb{E}_{p(x,z|\theta)}\left[\nabla_{\lambda}\log q(z|x,\lambda)\right]
\end{aligned}$$

The strategy for the sleep phase is to flip the KL around, that is, to assess the KL divergence of $p(z|x,\theta)$ from $q(z|x,\lambda)$.

- See that this change is in some sense convenient. Assume we are able to sample from the true posterior, then we can get gradient estimates w.r.t. λ. Clearly this is only superficially simple, as we have no means to sample from the true posterior.
- Here is where WS makes a big assumption, it assumes that sampling from the data $x \sim \mathcal{D}$ is equivalent to sampling from the marginal of the model $x \sim p(x|\theta)$, this can only be true if our model perfectly reproduces the data generating process. This is very unlikely in general, since the data generating process is unknown to us, and it's particularly unlikely at the beginning of training.
- With this assumption in place, it's easy to express the gradient as an expected gradient.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

$$\begin{aligned}
&\arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(p(z|x,\theta) \,||\, q(z|x,\lambda)\right)\right] \\
&= \arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{p(z|x,\theta)}\left[\log p(z|x,\theta) - \log q(z|x,\lambda)\right] \\
&\overset{\text{asm}}{=} \arg\max_{\lambda} \mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] - \underbrace{\mathbb{E}_{p(x,z|\theta)}\left[\log p(z|x,\theta)\right]}_{\text{constant}}
\end{aligned}$$

Gradient wrt λ

$$\begin{aligned}
&\nabla_{\lambda}\mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] \\
&= \mathbb{E}_{p(x,z|\theta)}\left[\nabla_{\lambda}\log q(z|x,\lambda)\right] \\
&\overset{\text{MC}}{\approx} \nabla_{\lambda}\log q(z|\tilde{x},\lambda) \quad \text{where } z\sim p(z|\theta) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \tilde{x}\sim p(x|z,\theta)
\end{aligned}$$

The strategy for the sleep phase is to flip the KL around, that is, to assess the KL divergence of $p(z|x,\theta)$ from $q(z|x,\lambda)$.

- See that this change is in some sense convenient. Assume we are able to sample from the true posterior, then we can get gradient estimates w.r.t. λ. Clearly this is only superficially simple, as we have no means to sample from the true posterior.
- Here is where WS makes a big assumption, it assumes that sampling from the data $x\sim\mathcal{D}$ is equivalent to sampling from the marginal of the model $x\sim p(x|\theta)$, this can only be true if our model perfectly reproduces the data generating process. This is very unlikely in general, since the data generating process is unknown to us, and it's particularly unlikely at the beginning of training.
- With this assumption in place, it's easy to express the gradient as an expected gradient.
- An MC estimation is possible by ancestral sampling from $p(x,z|\theta)$. This gives us a *dream* (model-generated) observation.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

$$\begin{aligned}
&\arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(p(z|x,\theta) \,||\, q(z|x,\lambda)\right)\right] \\
&= \arg\min_{\lambda} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{p(z|x,\theta)}\left[\log p(z|x,\theta) - \log q(z|x,\lambda)\right] \\
&\overset{\text{asm}}{=} \arg\max_{\lambda} \mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] - \underbrace{\mathbb{E}_{p(x,z|\theta)}\left[\log p(z|x,\theta)\right]}_{\text{constant}}
\end{aligned}$$

Gradient wrt λ

$$\begin{aligned}
&\nabla_{\lambda}\mathbb{E}_{p(x,z|\theta)}\left[\log q(z|x,\lambda)\right] \\
&= \mathbb{E}_{p(x,z|\theta)}\left[\nabla_{\lambda}\log q(z|x,\lambda)\right] \\
&\overset{\text{MC}}{\approx} \nabla_{\lambda}\log q(z|\tilde{x},\lambda) \quad \text{where } z\sim p(z|\theta) \\
&\qquad\qquad\qquad\qquad\qquad\qquad \tilde{x}\sim p(x|z,\theta)
\end{aligned}$$

The strategy for the sleep phase is to flip the KL around, that is, to assess the KL divergence of $p(z|x,\theta)$ from $q(z|x,\lambda)$.

- See that this change is in some sense convenient. Assume we are able to sample from the true posterior, then we can get gradient estimates w.r.t. λ. Clearly this is only superficially simple, as we have no means to sample from the true posterior.
- Here is where WS makes a big assumption, it assumes that sampling from the data $x\sim\mathcal{D}$ is equivalent to sampling from the marginal of the model $x\sim p(x|\theta)$, this can only be true if our model perfectly reproduces the data generating process. This is very unlikely in general, since the data generating process is unknown to us, and it's particularly unlikely at the beginning of training.
- With this assumption in place, it's easy to express the gradient as an expected gradient.
- An MC estimation is possible by ancestral sampling from $p(x,z|\theta)$. This gives us a *dream* (model-generated) observation.

Sleep Phase (Convenient) Objective

Assumes fake data $\tilde{x}$ and latent variables z to be fixed random draws from $p(x, z|\theta)$ via

$$z \sim p(z|\theta)$$
$$\tilde{x} \sim p(x|z, \theta)$$

and maximises $\log q(z|\tilde{x}, \lambda)$.

This is maximum likelihood estimation for the recognition model as if $z, \tilde{x}$ were observed.

Wake-sleep Algorithm

Advantages

- Simple layer-wise updates
- Amortised inference: all latent variables are inferred from the same weights λ

Drawbacks

- Inference and generative models are trained on different objectives
- Inference weights λ are updated on fake data $\tilde{x}$
- Generative weights are bad initially, giving wrong signal to the updates of λ

Though there are some instances of WS even in modern literature, its drawbacks are generally quite serious.

Frequentist VI

Variational Objective

$$\arg\max_{q(z)} \mathbb{E}_{q(z)}\left[\log p(x,z)\right] + \mathbb{H}\left(q(z)\right)$$

This finds us the best posterior approximation for a **given model**.

Frequentist VI also optimises the model!

$$\arg\max_{q(z),p(x,z)} \mathbb{E}_{q(z)}\left[\log p(x,z)\right] + \mathbb{H}\left(q(z)\right)$$

VI comes from the literature of Bayesian modelling, where it is known as Variational Bayes (VB). VB is concerned with the variational objective, i.e., ELBO maximisation w.r.t. a choice of posterior approximation $q(z)$.

In Frequentism, we make point estimates of model parameters. Whereas we can use the ELBO for that it should be noted that we are not optimising log-likelihood, as customary in MLE, rather we are optimising a lowerbound on it. There's no guarantee that an improvement in the lowerbound correlates with an improvement in log-evidence.

Coordinate Ascent Variational Inference

Frequentist VI can be performed via coordinate ascent. This can be done as a 2-step procedure.

1. Maximise (regularised) expected log-density.

$$\arg\max_{q(z)} \mathbb{E}_{q(z)} \left[\log p(x, z)\right] + \mathbb{H}\left(q(z)\right)$$

2. Optimise generative model.

$$\arg\max_{p(x,z)} \mathbb{E}_{q(z)} \left[\log p(x, z)\right] + \underbrace{\mathbb{H}\left(q(z)\right)}_{\text{constant}}$$

Think of our choice of approximation $q(z)$ and our choice of model $p(x, z)$ as coordinates.

We can keep one fixed an update the other. This is coordinate ascent VI.

Unconstrained (exact) optimisation

What's the solution to the following?

$$\arg\max_{q(z)\in\mathcal{Q}} \mathbb{E}_{q(z)}\left[\log p(x,z)\right] + \mathbb{H}\left(q(z)\right)$$

(assume $\mathcal{Q}$ is large enough a family)

Unconstrained (exact) optimisation

What's the solution to the following?

$$\arg\max_{q(z)\in\mathcal{Q}} \mathbb{E}_{q(z)}\left[\log p(x,z)\right] + \mathbb{H}\left(q(z)\right)$$

(assume $\mathcal{Q}$ is large enough a family)

The true posterior $p(z|x)$! Exactly because

$$\arg\max_{q(z)\in\mathcal{Q}} \text{ELBO} = \arg\min_{q(z)\in\mathcal{Q}} \text{KL}\left(q(z) \,||\, p(z|x)\right)$$

and KL is never negative and 0 iff $q(z) = p(z|x)$.

Recap: EM Algorithm

$$\text{E-step} \quad \arg\max_{q(z)} \mathbb{E}_{q(z)}\left[\log p(x,z)\right] + \mathbb{H}\left(p(z|x)\right)$$
$$= p(z|x)$$
$$\text{M-step} \quad \arg\max_{p(x,z)} \mathbb{E}_{p(z|x)}\left[\log p(x,z)\right] + \underbrace{\mathbb{H}\left(p(z|x)\right)}_{\text{constant}}$$

Expectation Maximisation (EM) is Frequentist variational inference where we solve ELBO maximisation w.r.t. $q(z)$ exactly, that is, we use the true posterior $p(z|x)$.

$$q(z) = p(z|x)$$
$$\mathrm{KL}\left(q(z) \,||\, p(z|x)\right) = 0$$

The implication is that we can only do EM for models whose marginals are already tractable (and thus do not require approximate inference).

When we train a discrete LVM with exact marginals via gradient-based MLE, we solve the marginal exactly (sidestepping the E-step), and the M-step approximately, via iterative gradient-based ascent.

Score Function Estimator: Variance

$$\frac{\partial}{\partial\lambda}\mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\right] = \mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right]$$

Empirically this estimator often exhibits high variance.

- the magnitude of $\log p(x|z,\theta)$ varies widely
- the model likelihood does not contribute to direction of gradient

The simplest way to reduce variance of an MC estimator is to sample more times. But it's not very efficient.

Control variates

Intuition

To estimate $\mathbb{E}[f(z)]$ via Monte Carlo we compute the empirical average of $\hat{f}(z)$ where $\hat{f}(z)$ is chosen so that $\mathbb{E}[\hat{f}(z)] = \mathbb{E}[f(z)]$ and $\text{Var}(f) > \text{Var}(\hat{f})$.

Equivalent expectations

Let $\bar{f} = \mathbb{E}[f(z)]$ be an expectation of interest

Equivalent expectations

Let $\bar{f} = \mathbb{E}[f(z)]$ be an expectation of interest

- say we know $\bar{c} = \mathbb{E}[c(z)]$

Equivalent expectations

Let $\bar{f} = \mathbb{E}[f(z)]$ be an expectation of interest

- say we know $\bar{c} = \mathbb{E}[c(z)]$
- then for $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$

Equivalent expectations

Let $\bar{f} = \mathbb{E}[f(z)]$ be an expectation of interest

- say we know $\bar{c} = \mathbb{E}[c(z)]$
- then for $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$ it holds that $\mathbb{E}[\hat{f}(z)] = \mathbb{E}[f(z)]$

Equivalent expectations

Let $\bar{f} = \mathbb{E}[f(z)]$ be an expectation of interest

- say we know $\bar{c} = \mathbb{E}[c(z)]$
- then for $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$
  it holds that $\mathbb{E}[\hat{f}(z)] = \mathbb{E}[f(z)]$
- and $\mathrm{Var}(\hat{f}) = \mathrm{Var}(f) - 2b\,\mathrm{Cov}(f, c) + b^2\,\mathrm{Var}(c)$

Choosing the control variate

1. $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$
2. $\mathrm{Var}(\hat{f}) = \mathrm{Var}(f) - 2b\,\mathrm{Cov}(f, c) + b^2\,\mathrm{Var}(c)$

How do we choose b and $c(z)$?

Choosing the control variate

1. $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$
2. $\mathrm{Var}(\hat{f}) = \mathrm{Var}(f) - 2b\,\mathrm{Cov}(f, c) + b^2\,\mathrm{Var}(c)$

How do we choose b and $c(z)$?

- If $f(z)$ and $c(z)$ are positively correlated, then we may reduce variance

Choosing the control variate

1. $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$
2. $\mathrm{Var}(\hat{f}) = \mathrm{Var}(f) - 2b\,\mathrm{Cov}(f, c) + b^2\,\mathrm{Var}(c)$

How do we choose b and $c(z)$?

- If $f(z)$ and $c(z)$ are positively correlated, then we may reduce variance
- solving $\frac{\partial}{\partial b}\,\mathrm{Var}(\hat{f}) = 0$

Choosing the control variate

1. $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$
2. $\mathrm{Var}(\hat{f}) = \mathrm{Var}(f) - 2b\,\mathrm{Cov}(f, c) + b^2\,\mathrm{Var}(c)$

How do we choose b and $c(z)$?

- If $f(z)$ and $c(z)$ are positively correlated, then we may reduce variance
- solving $\frac{\partial}{\partial b}\mathrm{Var}(\hat{f}) = 0$ yields $b^\star = {}^{\mathrm{Cov}(f,c)}/_{\mathrm{Var}(c)}$

Choosing the control variate

1. $\hat{f}(z) \triangleq f(z) - b(c(z) - \mathbb{E}[c(z)])$
2. $\mathrm{Var}(\hat{f}) = \mathrm{Var}(f) - 2b\,\mathrm{Cov}(f, c) + b^2\,\mathrm{Var}(c)$

How do we choose b and $c(z)$?

- If $f(z)$ and $c(z)$ are positively correlated, then we may reduce variance
- solving $\frac{\partial}{\partial b}\mathrm{Var}(\hat{f}) = 0$ yields $b^\star = {}^{\mathrm{Cov}(f,c)}/_{\mathrm{Var}(c)}$

Of course, $\mathbb{E}[c(z)]$ must be known!

MC

We then use the estimate

$$\bar{f} \overset{\text{MC}}{\approx} \frac{1}{S}\left(\sum_{s=1}^{S} f(z^{(s)}) - bc(z^{(s)})\right) + b\bar{c}$$

MC

We then use the estimate

$$\bar{f} \overset{\text{MC}}{\approx} \frac{1}{S}\left(\sum_{s=1}^{S} f(z^{(s)}) - bc(z^{(s)})\right) + b\bar{c}$$

And recall that for us

$$f(z) = \log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

and $z^{(s)} \sim q(z|x,\lambda)$

Expected score

The Expectation of the score function is 0.

$$\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right]$$

Expected score

The Expectation of the score function is 0.

$$\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right]$$
$$= \int q(z|x,\lambda)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\mathrm{d}z$$

Expected score

The Expectation of the score function is 0.

$$\begin{aligned}
&\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right] \\
&= \int q(z|x,\lambda)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\mathrm{d}z \\
&= \int \frac{\partial}{\partial\lambda} q(z|x,\lambda)\mathrm{d}z
\end{aligned}$$

Expected score

The Expectation of the score function is 0.

$$
\begin{aligned}
&\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right] \\
&= \int q(z|x,\lambda)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\mathrm{d}z \\
&= \int \frac{\partial}{\partial\lambda} q(z|x,\lambda)\mathrm{d}z \\
&= \frac{\partial}{\partial\lambda}\int q(z|x,\lambda)\mathrm{d}z
\end{aligned}
$$

Expected score

The Expectation of the score function is 0.

$$
\begin{aligned}
&\mathbb{E}_{q(z|x,\lambda)}\left[\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\right] \\
&= \int q(z|x,\lambda)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)\mathrm{d}z \\
&= \int \frac{\partial}{\partial\lambda} q(z|x,\lambda)\mathrm{d}z \\
&= \frac{\partial}{\partial\lambda}\int q(z|x,\lambda)\mathrm{d}z \\
&= \frac{\partial}{\partial\lambda} 1 = 0
\end{aligned}
$$

Baselines

With

$$f(z) = \log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

and

$$c(z) = \frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

we have

$$\hat{f}(z) =$$

Baselines

With

$$f(z) = \log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

and

$$c(z) = \frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

we have

$$\hat{f}(z) = (\log p(x|z,\theta) - b)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

Baselines

With

$$f(z) = \log p(x|z,\theta)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

and

$$c(z) = \frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

we have

$$\hat{f}(z) = (\log p(x|z,\theta) - b)\frac{\partial}{\partial\lambda}\log q(z|x,\lambda)$$

b is known as *baseline* in RL literature.

Examples of baselines

- Moving average of $\log p(x|z,\theta)$ based on previous batches

Examples of baselines

- Moving average of $\log p(x|z,\theta)$ based on previous batches
- A trainable constant b

Examples of baselines

- Moving average of $\log p(x|z,\theta)$ based on previous batches
- A trainable constant b
- A neural network prediction based on x e.g. $b(x;\omega)$

Examples of baselines

- Moving average of $\log p(x|z,\theta)$ based on previous batches
- A trainable constant b
- A neural network prediction based on x e.g. $b(x;\omega)$
- The likelihood assessed at a deterministic point, e.g. $b(x) = \log p(x|z^\star,\theta)$ where $z^\star = \arg\max_z q(z|x,\lambda)$

Trainable baselines

Baselines are predicted by a regression model (e.g. a neural net).

The model is trained using an L_2-loss.

$$\min_{\omega} \left(b(x;\omega) - \log p(x|z,\theta)\right)^2$$

Summary

- In practice the score function estimator leads to high variance gradient estimates.
- We can design control variates that reduce estimator variance, yet do not bias the estimator!

References I

Jasmijn Bastings, Wilker Aziz, and Ivan Titov. Interpretable neural predictions with differentiable binary variables. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 2963–2977, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1284. URL https://www.aclweb.org/anthology/P19-1284.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent Dirichlet Allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944937. Publisher: JMLR.org.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. Publisher: Taylor & Francis.

References II

Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, and Robert L. Mercer. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311, 1993. URL https://www.aclweb.org/anthology/J93-2003.

Zoubin Ghahramani and Thomas L. Griffiths. Infinite latent feature models and the Indian buffet process. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 475–482. MIT Press, 2006.

Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the Void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyzKd1bCW.

References III

Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research*, 5(Nov):1471–1530, 2004. ISSN ISSN 1533-7928. URL https://www.jmlr.org/papers/v5/greensmith04a.html.

Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. MuProp: Unbiased backpropagation for stochastic neural networks. In *ICLR (poster)*, 2016. URL http://arxiv.org/abs/1511.05176. tex.cdate: 1451606400000 tex.crossref: conf/iclr/2016.

G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The Wake-Sleep Algorithm for Unsupervised Neural Networks. *Science*, 268:1158–1161, 1995.

References IV

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, November 1999. ISSN 1573-0565. doi: 10.1023/A:1007665907178. URL https://doi.org/10.1023/A:1007665907178.

Durk P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised Learning with Deep Generative Models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3581–3589. Curran Associates, Inc., 2014.

References V

Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing Neural Predictions. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 107–117, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1011. URL https://www.aclweb.org/anthology/D16-1011.

Runjing Liu, Jeffrey Regier, Nilesh Tripuraneni, Michael Jordan, and Jon Mcauliffe. Rao-blackwellized stochastic gradients for discrete distributions. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *ICML*, volume 97 of *Proceedings of machine learning research*, pages 4023–4031, Long Beach, California, USA, June 2019. PMLR. URL http://proceedings.mlr.press/v97/liu19c.html. tex.pdf: http://proceedings.mlr.press/v97/liu19c/liu19c.pdf.

References VI

Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *Proceedings of the seventeenth conference on uncertainty in artificial intelligence*, UAI'01, pages 362–369, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-800-1. Number of pages: 8 Place: Seattle, Washington.

Andriy Mnih and Karol Gregor. Neural Variational Inference and Learning in Belief Networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1791–II–1799. JMLR.org, 2014. event-place: Beijing, China.

Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *ICML*, volume 48 of *Proceedings of machine learning research*, pages 2188–2196, New York, New York, USA, June 2016. PMLR. URL http://proceedings.mlr.press/v48/mnihb16.html. tex.pdf: http://proceedings.mlr.press/v48/mnihb16.pdf.

References VII

Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo Gradient Estimation in Machine Learning. *CoRR*, abs/1906.10652, 2019. URL http://arxiv.org/abs/1906.10652.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In Samuel Kaski and Jukka Corander, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*, pages 814–822, Reykjavik, Iceland, April 2014. PMLR. URL http://proceedings.mlr.press/v33/ranganath14.html.

Steven J. Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-Critical Sequence Training for Image Captioning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 1179–1195. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.131. URL https://doi.org/10.1109/CVPR.2017.131.

References VIII

Miguel Rios, Wilker Aziz, and Khalil Sima'an. Deep Generative Model for Joint Alignment and Word Representation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1011–1023, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1092. URL https://www.aclweb.org/anthology/N18-1092.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in neural information processing systems*, pages 3528–3536, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

References IX

George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2627–2636. Curran Associates, Inc., 2017.

Aki Vehtari, Andrew Gelman, Tuomas Sivula, Pasi Jylänki, Dustin Tran, Swupnil Sahai, Paul Blomstedt, John P Cunningham, David Schiminovich, and Christian P Robert. Expectation propagation as a way of life: A framework for bayesian inference on partitioned data. *Journal of Machine Learning Research*, 21(17):1–53, 2020.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. HMM-Based word alignment in statistical translation. In *COLING 1996 volume 2: The 16th international conference on computational linguistics*, 1996. URL https://www.aclweb.org/anthology/C96-2141.

References X

Weiyue Wang, Derui Zhu, Tamer Alkhouli, Zixuan Gan, and Hermann Ney. Neural Hidden Markov Model for Machine Translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 377–382, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2060. URL https://www.aclweb.org/anthology/P18-2060.

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8(3-4): 229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

References XI

Chunting Zhou and Graham Neubig. Multi-space variational encoder-decoders for semi-supervised labeled sequence transduction. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 310–320, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1029. URL https://www.aclweb.org/anthology/P17-1029.