Deep Discrete Latent Variable Models

ProbAi – 2021

Wilker Aziz
w.aziz@uva.nl

University of Amsterdam
Institute for Logic, Language and Computation

Outline

1 Overview

2 Modelling Random Experiments
- Modelling Observed Data
- Modelling Unobserved Data

3 Approximate Inference

4 Learning
- Score Function Estimator
- Reparameterised Gradients
- Sparse Gradients

5 Diagnostics

Hello

I am an assistant professor at the University of Amsterdam, you can check some of my work here https://probabll.github.io

Stuff I typically work on include

- machine learning
  VAEs, normalising flows, gradient estimation for discrete variables
- probabilistic models
  Bayesian models, deep latent variable models
- natural language processing
  translation, parsing, text classification, question answering, interpretability and transparency

Probabilistic models

A probabilistic model *predicts* possible outcomes of a random experiment.

This ‘prediction’ often requires mapping from complex data types (e.g., images, text, graphs) to probability distributions (or energy functions, or random outcomes) in a space of complex data types (e.g., images, text, graphs).

NNs are excellent at learning complex mappings, thus it is only natural that we seek using NNs to help specify probabilistic models.

ProbAi

At the school, you've seen that *unobserved variables* make probabilistic models all the more interesting.

Unfortunately, they also lead to some key challenges.

This class will deal with difficulties arising from discrete unobserved random variables.

Probabilistic Modelling and Reasoning

Probabilistic modelling concerns the specification of a joint distribution over random variables of interest.

Probabilistic reasoning concerns fixing a subset of these random variables to some observations and inferring marginal and conditional distributions by application of probability calculus.

The latter is also know as *probabilistic inference*.

If you would like to learn *all* about PGMs, check the excellent book by Koller and Friedman (2009). I'd recommend Part I (on representation of distributions) to *anyone*.

Learning

Oftentimes, we begin specifying a probability distribution by specifying a class of distributions.

Parameter estimation (e.g., via MLE) singles out a member of the class.

For many practical models, the computations needed for parameter estimation are tractable, but for most latent variable models this is not true (e.g., marginal inferences are intractable). Models like that call for *approximate inference*.

For example, a neural language model is an autoregressive factorisation of the probability of a sequence, i.e.,

$$p(\langle x_1, \ldots, x_I \rangle | \theta) = \prod_{i=1}^{I} p(x_i | x_{<i}, \theta)$$

where each conditional is a Categorical distribution predicted by an NN:

$$X_i | x_{<i}, \theta \sim \mathrm{Cat}(\mathbf{f}(x_{<i}; \theta))$$

Typical algorithms for parameter estimation require assessing

$$p(x_i = c | x_{<i}, \theta) = f_c(x_{<i}; \theta)$$

and its gradient, which take only a forward and a backward pass through a tractable computation graph.

What are the benefits of probabilistic models?

Probabilistic models allows to incorporate assumptions through

- the choice of distribution
- dependencies among random variables
- the way that distributions uses side information
- stipulate unobserved data and their properties

They return a distribution over outcomes which can be used to

- generate data
- account for unobserved data
- provide explanation and suggest improvements
- inform decision makers

Outline

1 Overview

2 Modelling Random Experiments
- Modelling Observed Data
- Modelling Unobserved Data

3 Approximate Inference

4 Learning
- Score Function Estimator
- Reparameterised Gradients
- Sparse Gradients

5 Diagnostics

Modelling random experiments

We treat data, the main subject of statistical interest, as outcomes of experiments involving random variables.

A *model* of the data may

- shed light onto properties of data
  (or our own understanding of such properties);
- be used to support decisions about existing and future data.

Modelling data does not imply solving a predictive task.

For example, a generative classifier is a joint distribution $p(y)p(x|y,\theta)$ over labels $y \in \mathcal{Y}$ and inputs $x \in \mathcal{X}$. Making a specific prediction for a novel input x_* is a decision problem, oftentimes handled independently of model specification and learning. A common decision rule for classification is $y_* = \arg\max_y p(y|x_*,\theta)$.

Modelling random experiments

We treat data, the main subject of statistical interest, as outcomes of experiments involving random variables.

A *model* of the data may

- shed light onto properties of data
  (or our own understanding of such properties);
- be used to support decisions about existing and future data.

Goals for this section

- be able to specify joint distributions of observed and unobserved data
- be able to estimate (some of) these distributions
- be able to recognise applications to modelling with latent data

Modelling data does not imply solving a predictive task.

For example, a generative classifier is a joint distribution $p(y)p(x|y,\theta)$ over labels $y \in \mathcal{Y}$ and inputs $x \in \mathcal{X}$. Making a specific prediction for a novel input x_* is a decision problem, oftentimes handled independently of model specification and learning. A common decision rule for classification is $y_* = \arg\max_y p(y|x_*,\theta)$.

Outline

1 Overview

2 Modelling Random Experiments
- Modelling Observed Data
- Modelling Unobserved Data

3 Approximate Inference

4 Learning
- Score Function Estimator
- Reparameterised Gradients
- Sparse Gradients

5 Diagnostics

Modelling observed random variables

Our goal is to learn a distribution over a set of **observed** random variables.

Observed random variables are the result of random experiments that have already happened: e.g., sentences in a collection of news articles, number of stars in a product review.

Modelling observed random variables

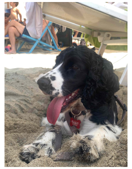
Our goal is to learn a distribution over a set of **observed** random variables.

Observed random variables are the result of random experiments that have already happened: e.g., sentences in a collection of news articles, number of stars in a product review.

Typical use in ML: *conditional models.*
▷ *We are given some variables (inputs) and we are interested in making predictions about other variables (outputs)*

- such inputs are also called *predictors* (or *covariates*)
- with some probability, *predicted by the model*, an output takes on a certain *outcome* in a sample space

Modelling conditionally - Examples

Predictor	Outcome	Sample space
Why did they bother recording this???	$\star$	$\{\star, \star\star, \star\star\star, \star\star\star\star, \star\star\star\star\star\}$
Source: geen standaard MT: no standard	0.5	$[0, 1]$
proposed a famous solution to an inverse probability problem in the 18th century	https://en.wikipedia.org/wiki/Thomas_Bayes	$\mathcal{W}_{\text{en}}$
	*Pepper loves the beach!*	Σ^*_{en}
That's not possible!	*Dat is niet mogelijk!*	Σ^*_{nl}

1. text classification
2. machine translation quality estimation
3. question answering
4. image captioning
5. machine translation

Shallow statistical models

We have data $x^{(1)}, \dots, x^{(N)}$
generated by some **unknown** procedure which we assume can be captured by a probabilistic model

- with **known** probability (mass/density) function e.g.

$$X \sim \text{Cat}(\pi_1, \dots, \pi_K) \qquad \text{or} \qquad X \sim \mathcal{N}(\mu, \sigma^2)$$

and estimate parameters via MLE

In this class we take a Frequentist approach and rely on point estimates via MLE.

In a Bayesian treatment we would acknowledge that parameters are unobservable random variables and give them the random treatment they deserve.

Parameterisation by NNs

Let h be all side information available
e.g. *inputs/features/predictors/covariates*

Have neural networks predict parameters of our probabilistic model

$$X|h \sim \mathrm{Cat}(\pi(h;\theta)) \qquad \text{or} \qquad X|h \sim \mathcal{N}(\mu(h;\theta), \sigma(h;\theta)^2)$$

and proceed to estimate parameters θ of the NNs

NNs compute/predict the parameters of the statistical model from available predictors.

Maximum likelihood estimation

We have a probability model of a random variable X, this model uses a set of deterministic parameters θ to assign probability $p(x|\theta)$ to an observation.

Given a dataset $\mathcal{D} = \{x^{(1)}, \ldots, x^{(N)}\}$ of i.i.d. observations, the log-likelihood function gives us a criterion for parameter estimation

$$\mathcal{L}_{\mathcal{D}}(\theta) = \log \prod_{s=1}^{N} p(x^{(s)}|\theta) = \sum_{n=1}^{N} \log p(x^{(n)}|\theta)$$

MLE via gradient-based optimisation

If the log-likelihood is **differentiable** and **tractable**
then backpropagation gives us the gradient

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \nabla_\theta \sum_{n=1}^{N} \log p(x^{(n)}|\theta) \quad = \sum_{n=1}^{N} \nabla_\theta \log p(x^{(n)}|\theta)$$

and we can update θ in the direction

$$\gamma \nabla_\theta \mathcal{L}_\mathcal{D}(\theta)$$

to attain a local maximum of the likelihood function

Stochastic optimisation

For large N, we can use a gradient estimate

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N\nabla_\theta \log p(x^{(S)}|\theta)\right]}_{\text{expected gradient :)}}$$

The theory of stochastic optimisation (Robbins and Monro, 1951) tells us that we will converge to a local optimum of the objective as long as we take small steps whose directions are correct *on average*. This means we can use (stochastic) gradient estimates, for as long as they are unbiased estimates of the exact gradient.

If you want to read more, but need something more accessible than the 1951 paper, check (Bottou, 2010).

Stochastic optimisation

For large N, we can use a gradient estimate

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N\nabla_\theta \log p(x^{(S)}|\theta)\right]}_{\text{expected gradient :)}}$$

$$\overset{\text{MC}}{\approx} \frac{1}{M}\sum_{m=1}^{M} N\nabla_\theta \log p(x^{(s_m)}|\theta)$$

$$S_m \sim \mathcal{U}(1/N)$$

The theory of stochastic optimisation (Robbins and Monro, 1951) tells us that we will converge to a local optimum of the objective as long as we take small steps whose directions are correct *on average*. This means we can use (stochastic) gradient estimates, for as long as they are unbiased estimates of the exact gradient.

If you want to read more, but need something more accessible than the 1951 paper, check (Bottou, 2010).

Stochastic optimisation

For large N, we can use a gradient estimate

$$\nabla_\theta \mathcal{L}_\mathcal{D}(\theta) = \underbrace{\mathbb{E}_{S\sim\mathcal{U}(1/N)}\left[N\nabla_\theta \log p(x^{(S)}|\theta)\right]}_{\text{expected gradient :)}}$$

$$\overset{\text{MC}}{\approx} \frac{1}{M}\sum_{m=1}^{M} N\nabla_\theta \log p(x^{(s_m)}|\theta)$$

$$S_m \sim \mathcal{U}(1/N)$$

and take a step in the direction

$$\gamma\frac{N}{M}\underbrace{\nabla_\theta \mathcal{L}_\mathcal{B}(\theta)}_{\text{stochastic gradient}}$$

where $\mathcal{B} = \{x^{(s_1)}, \ldots, x^{(s_M)}\}$ is a random mini-batch

The theory of stochastic optimisation (Robbins and Monro, 1951) tells us that we will converge to a local optimum of the objective as long as we take small steps whose directions are correct *on average*. This means we can use (stochastic) gradient estimates, for as long as they are unbiased estimates of the exact gradient.

If you want to read more, but need something more accessible than the 1951 paper, check (Bottou, 2010).

Recipe for Learning with Fully Observed Data

NNs

- flexibly parameterise complex joint distributions

Maximum likelihood estimation

- tells you which loss to optimise

Automatic differentiation

- tractable loss
- intermediate representations must be continuous
- activations must be differentiable

Recipe for Learning with Fully Observed Data

NNs

- flexibly parameterise complex joint distributions

Maximum likelihood estimation

- tells you which loss to optimise

Automatic differentiation

- tractable loss
- intermediate representations must be continuous
- activations must be differentiable

Modelling with unobserved variables (generally) violates tractability and/or differentiability constraints.

Outline

1. Overview
2. Modelling Random Experiments
   - Modelling Observed Data
   - Modelling Unobserved Data
3. Approximate Inference
4. Learning
   - Score Function Estimator
   - Reparameterised Gradients
   - Sparse Gradients
5. Diagnostics

Modelling unobserved random variables

Unobserved random variables are variables that are

- observable in principle, but not available for observation (e.g., the topic of a piece of text, a semantic graph)
- unobservable (e.g., a 100-dimensional sentence embedding)

they help us prescribe and even estimate our models.

Our goal is to learn a distribution over *observed and unobserved rvs*

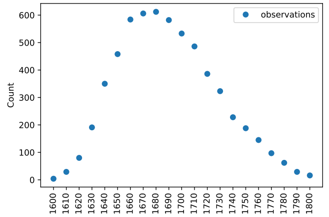
- make explicit assumptions about statistical dependence
- discover hidden structure
- mimic intuitions/knowledge about the data generating process
- deal with missing data
- estimate uncertainty about predictions

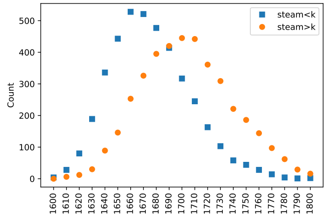
Unobserved random variables are also called latent variables.

For those interested in Bayesian statistics, note that the presence of unobserved random variables does not imply Bayesian modelling. Bayesian principles are a collection of ideas organised in what is called the Bayesian Theory (or Bayesian Decision Theory) for rational decision making under uncertainty (Bernardo and Smith, 2009). These ideas may cross paths with many aspects of our ML solutions.

Latent Structure and Over-Dispersion

We found some old manuscripts in an excavation site, historians began labelling them for publication date.





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

Left: publication date of documents in the labelled collection.

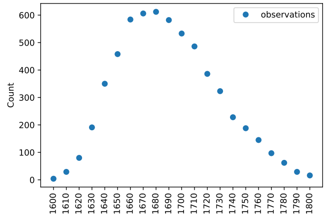
Our historians claim that above some threshold *k* a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society).

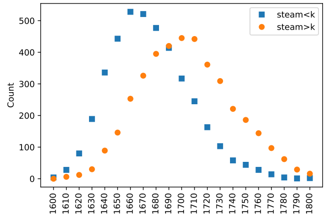
Right: data as a function of the frequency of the word *steam*.

Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Latent Structure and Over-Dispersion

We found some old manuscripts in an excavation site, historians began labelling them for publication date.





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

Left: publication date of documents in the labelled collection.

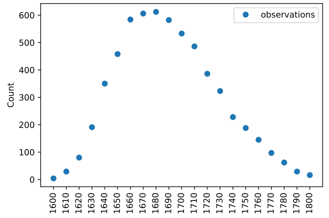
Our historians claim that above some threshold *k* a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society).

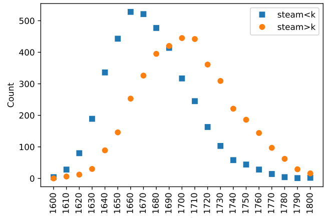
Right: data as a function of the frequency of the word *steam*.

Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Latent Structure and Over-Dispersion

We found some old manuscripts in an excavation site, historians began labelling them for publication date.





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

Left: publication date of documents in the labelled collection.

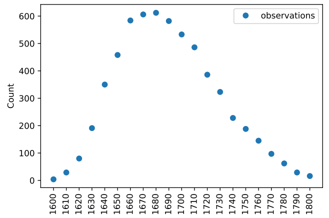
Our historians claim that above some threshold k a document was likely written after 1699 (when Thomas Savery demonstrated his first steam engine to the British Royal society).

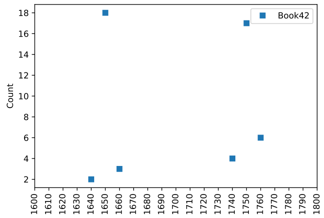
Right: data as a function of the frequency of the word *steam*.

Plotting two streams of data under such criterion reveals what could have been 2 different Poisson distributions.

Latent Structure and Multimodality

For a few manuscripts, we obtained labels from multiple experts.





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

a unimodal conditional $Y|\mathrm{doc}$ seems inappropriate.

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 $Y|\mathrm{doc}$. If we group our data into bins, where bin membership depends on matching a specific value of doc, 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 $Y|\mathrm{doc}$ 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 doc and draw Y multiple times (rightmost plot on the slide).

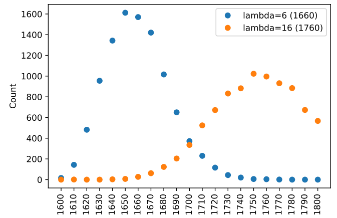
Can we combine simple distributions?

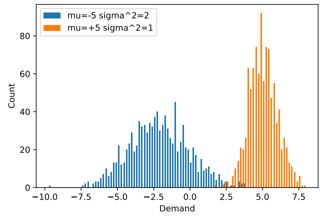
We can however *mix* K members of each family to get a good fit:

For example, with $K = 2$

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

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





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

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

to joint outcomes in the sample space $\mathcal{Z} \times \mathcal{X}$. 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 x 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:

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

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

Prescribing Flexible Distributions

The **marginal** distribution of the mixture model is potentially multimodal and exhibits richer covariance structure. It assigns probability

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

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

Marginal inference for the MM scales linearly in the number of *components*. This is a key type of computation, for example, indispensable for parameter estimation / learning via maximum likelihood.

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

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

Learning an LVM via 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.

Learning an LVM via 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.

Learning an LVM via 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.

Learning an LVM via 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 $\nabla_\theta \log p(x|\theta)$

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

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.

Learning an LVM via 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 $f' = f(\log f)'$

Learning an LVM via 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.

Learning an LVM via 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 $\mathcal{Z}$.

Learning an LVM via 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! Dependence on $Z|x,\theta$ makes the gradient $G_x(Z) = \nabla_\theta \log p(Z, X = x|\theta)$ a random variable.

Learning an LVM via 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 **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! Dependence on $Z|x, \theta$ makes the gradient $G_x(Z) = \nabla_\theta \log p(Z, X = x|\theta)$ a random variable.
- The gradient of the log-marginal $g_x = \nabla_\theta \log p(x|\theta)$ is deterministic, it is the expected value $\mathbb{E}_{Z|X=x,\theta}[G_x(Z)]$.

Learning an LVM via 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 **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! Dependence on $Z|x, \theta$ makes the gradient $G_x(Z) = \nabla_\theta \log p(Z, X = x|\theta)$ a random variable.
- The gradient of the log-marginal $g_x = \nabla_\theta \log p(x|\theta)$ is deterministic, it is the expected value $\mathbb{E}_{Z|X=x,\theta}[G_x(Z)]$.

A Remarkable Result

We have just seen that

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

This reveals two ways to learn LVMs

1. We can evaluate the marginal and let autodiff evaluate the expectation
2. We can sample a latent assignment z from the posterior and let autodiff evaluate $\nabla_\theta \log p(z,x|\theta)$ instead.

Posterior Inference

We have now met another object that requires probabilistic inference: the posterior distribution.

Assessing posterior probability mass/density requires marginal inference.

These inferences are useful beyond parameter estimation. For a trained model, they are just as useful (if not more!).

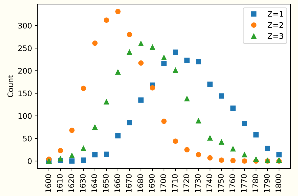
The posterior, for example, can be regarded as an interface through which we can appreciate how the model (the joint distribution) maps from latent space to data space and back.

Posterior Component Assignment in MMs

Given an observation x, and an estimated MM, we can infer a distribution over component assignments via Bayes rule

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

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.

Semi-supervised learning

Suppose some documents are annotated and others are not, and say we model generatively.

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

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

and the marginal probability of an unlabelled document x is

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

For a discrete and finite set $\mathcal{Y}$, this is a mixture model!

This is a very special mixture model for its components are *labelled* with self-evident information (e.g., decades).

A generative model of this kind can be thought of as a classifier, 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. Our observations are indeed a collection of documents, where a few documents are labelled for decade. When we model conditionally we call the labelled instances *training data* and ignore all unlabelled instances (the vast majority of our observations).

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 $i = 1, \ldots, |x|$.

Tractable Marginal Inference

Models with tractable marginals (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, not on how that model's probability distributions are parameterised.

Marginalisation algorithms are generally harder to parallelise on GPUs.

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, marginal inference is computable in cubic-time (wrt sequence length) via the inside algorithm. Similarly, there is an exponential number of state sequences in an HMM, but its marginal is computable in linear-time (wrt sequence length) via the forward algorithm.

Kim et al. (2017) present some efficient solutions. See also `https://github.com/harvardnlp/pytorch-struct`

Warnings

In the presence of latent variables, crucial **computations become challenging** and often require approximations.

Discrete latent variables pose further challenges.

- marginal inference requires enumeration or dynamic programming
- sometimes no tractable algorithm is known, as we shall see next

Competition or cooperation?

In a mixture model the components *compete* to generate a data point. This means they each account for some of the 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), especially so when we have some amount of supervision.

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 document model

Let us consider a latent factor model for document modelling:

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

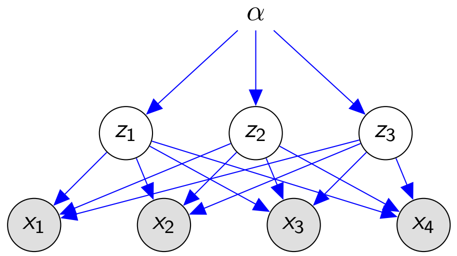
$$\begin{aligned} Z_j &\sim \text{Bernoulli}(\alpha) & (1 \leq d \leq D) \\ X_i|z &\sim \text{Categorical}\left(f(z;\theta)\right) & (1 \leq i \leq I) \end{aligned}$$

Here $f(\cdot;\theta)$ is an NN and θ its parameters.

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

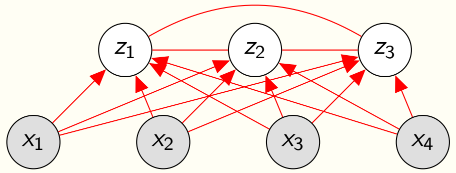
Joint distribution: independent latent variables



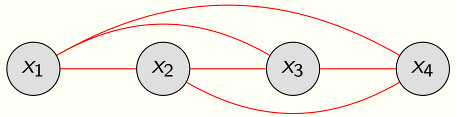
I omit θ from the graphical model, but every $X_i|z$ depends on it.

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

Posterior



Marginal



I'm omitting θ and α from the graphical models.

Intractable Marginals

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

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

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

But we know that

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

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

Perhaps we can estimate the gradient?

Combinatorial Latent Structure

The posterior of the latent factor model reveals attributes that are relevant to an observation.

Sampling from it can help discover discrete factors of variation (e.g. morphological attributes of a word).

Unfortunately, the posterior in this case is a Gibbs distribution whose parameter is intractable to compute

- its natural parameter has length 2^D
- its log-normaliser requires a summation over $z \in \{0, 1\}^D$

Applications

Alignment Learn to match two data structures (e.g., word alignment, phrase alignment, visual question answering).

Data: $\langle x_1, \ldots, x_I \rangle$ and $\langle y_1, \ldots, y_J \rangle$

Generate each part of y using a subset of the parts of x.

- this can be a mixture model
- or a latent factor model
- and there can be constraints on the parts (e.g., disjoint)

(Rios et al., 2018; Deng et al., 2018; Kawakami et al., 2019)

Applications

Latent attribution What parts of the input (or of a computation graph) affect predictions.

Data: $\langle x_1, \ldots, x_I \rangle$ and y

$$Z_i \sim \text{Bern}(\alpha)$$
$$Y|x, z \sim \text{Cat}(f(x \odot z; \theta))$$

x could also be every hidden state internal to a given NN, and y could be that NN's output $y = g(x; \phi)$

(Lei et al., 2016; Bastings et al., 2019; Cao et al., 2020)

Applications

Compositionality Learn a computation graph.

Sample a structure z from a prior or conditional distribution $Z|x$ and let this structure determine a composition function to parameterise a distribution $Y|x, z$. This can be used for semi-supervised learning of syntactic/semantic representations, for learning to solve arithmetic expressions, interpretable text classifiers, etc.

(Yogatama et al., 2017; Corro and Titov, 2018; Niculae et al., 2018b; Havrylov et al., 2019)

Applications

Controllable generation Learn to affect a conditional generator by controlling a latent prompt.

For example, translation models parameterise a conditional distribution $Y|x, \theta$ over translations of a given input x.

The source may contain a certain word (e.g., doctor), and the target language gender-marks nouns. The source sentence *does not* contain enough information to resolve the ambiguity, wouldn't it be nice to have a mechanism, other than requiring the user to produce a less ambiguous x, to control inflections?

(Hu et al., 2017; Zhou and Neubig, 2017; Ataman et al., 2020)

Applications

(Hu et al., 2017; Zhou and Neubig, 2017; Ataman et al., 2020)

Summary

Be able to specify joint distributions of observed and unobserved data

- have NNs predict the parameters of our conditional distributions

Be able to estimate (some of) these distributions

- MLE for observed data can be automated
- for LVMs we need tractable marginal and/or posterior inference

Be able to recognise applications to modelling with (discrete) latent data

- increased flexibility (e.g., multimodality, over-dispersion)
- unsupervised learning (e.g., word alignments, LDA, IBP)
- semi-supervised learning (e.g., generative classifiers, disentanglement)
- transparency (e.g., latent rationale for predictions)

Many discrete LVMs admit tractable marginalisation. Yet, many do not!
What happens when we cannot solve $\sum_{z \in \mathcal{Z}} p(x, z|\theta)$?

Outline

1. Overview
2. Modelling Random Experiments
   - Modelling Observed Data
   - Modelling Unobserved Data
3. Approximate Inference
4. Learning
   - Score Function Estimator
   - Reparameterised Gradients
   - Sparse Gradients
5. Diagnostics

When Inference is Intractable

When marginal (or posterior) inferences are intractable, we resort to approximate inference techniques.

Markov chain Monte Carlo [another day]

- Simulate a Markov chain that has $p(z|x, \theta)$ as stationary distribution.

Variational methods [today]

- Fit an independent approximation $q(z|x, \lambda)$ without requiring assessments of $p(x|\theta)$.

Goal for this section Circumvent the need for marginal inference.

Strategy

Variational Inference

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

Variational Inference

Approximate Posterior Inference

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

For a fixed joint distribution $p(z,x|\theta)$, that is, a fixed choice of θ, the minimiser of the VI objective is

$$\arg\max_{\lambda}\ \mathrm{ELBO}(\lambda;\theta)$$

which is simpler in that it dispenses with marginal inference.

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 class. 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)

ELBO

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

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

ELBO highlights

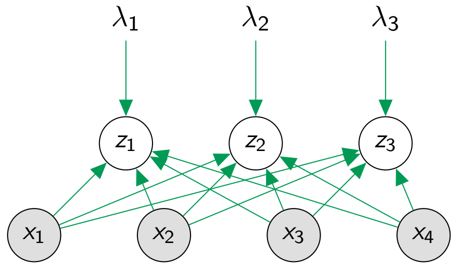
- we get to design $q(z|x,\lambda)$, so for example, while the true posterior of a latent factor model depends on an intractable marginalisation, the approximate posterior $q(z|x,\lambda)$ might simply combine D independent Bernoulli distributions (one per latent factor);
- as we get to pick $q(z|x,\lambda)$, we choose a family that's convenient, for example, one for which we can obtain independent samples;
- tractable sampling from $q(z|x,\lambda)$ means that we can obtain MC estimates of the ELBO (no need for MCMC);
- given z, an ELBO estimate only requires assessing the joint probability $\log p(x,z|\theta)$ and $\log q(z|x,\lambda)$, both tractable by design;
- ideally, we would choose a family for which entropy (or relative entropy) is also tractable.

Designing a tractable approximation

Mean field approximation:

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

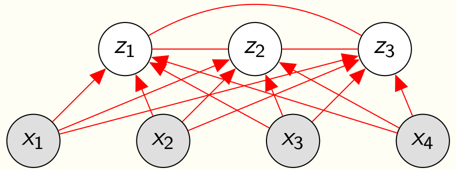
Mean Field Latent Factor Model Inference



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

Instead of inferring the true posterior $Z|X = x, \theta$,

Posterior



which 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.

Amortised variational inference

Let a shared NN predict variational factors for different observations

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

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.

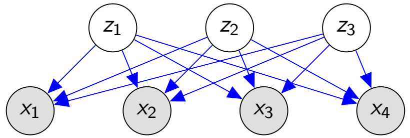
With NNs, we can condition on complex data efficiently, thus it seems like an interesting idea to jointly parameterise the independent factors of the posterior approximation $q(z|x, \lambda)$.

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.

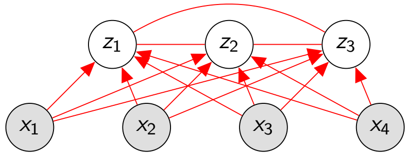
Recall our notion of model: a mechanism to predict the outcomes of a random experiment. So far, we've been attempting to model outcomes of some joint distribution $p(x, z|\theta)$. In variational inference, we introduce a rather unusual model, i.e., $q(z|x, \lambda)$, it predicts another model's posterior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

Overview

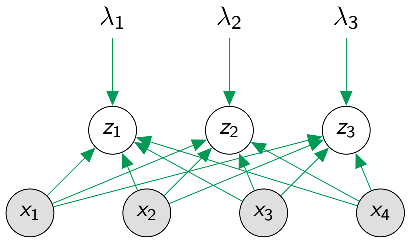
Joint distribution



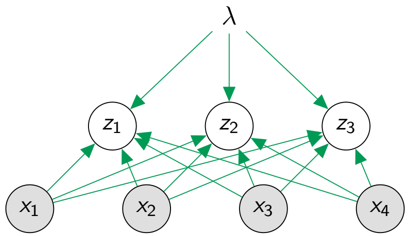
Posterior



Mean field



Amortised VI



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

Posterior: latent variables are potentially dependent. 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

Intractable posterior and marginal inferences

- prevent learning: gradient wrt θ is an expectation under the posterior
- make it difficult to reason using a trained model

We now have a mechanism (other than classic MCMC) to approximate posterior inferences

- optimise an approximation in an objective that dispenses with assessing the marginal

Inferences can be approximated using this new object

- Reason using $q(z|x, \lambda)$ instead of $p(z|x, \theta)$
- Use importance sampling to approximate marginalisation

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

Recap this section's goal: circumvent the need for marginal inference.

Outline

1 Overview

2 Modelling Random Experiments
- Modelling Observed Data
- Modelling Unobserved Data

3 Approximate Inference

4 Learning
- Score Function Estimator
- Reparameterised Gradients
- Sparse Gradients

5 Diagnostics

Model Learning

We first considered a mechanism for approximate posterior inference (i.e., for a fixed joint distribution).

We now consider the case where we also need to learn the joint distribution (i.e., estimate θ).

For that, we will simply use the ELBO, and fit both models in tandem.

Goal for this section Gradient-based estimation of θ and λ.

Variational EM

Approximate Posterior Inference

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

Model Learning

$$\begin{aligned}&\min_{\theta}\ \mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(q(z|x,\lambda)\ ||\ p(z|x,\theta)\right)\right]\\ =&\min_{\theta}\ \underbrace{\mathbb{E}_{x\sim\mathcal{D}}\left[\mathrm{KL}\left(q(z|x,\lambda)\ ||\ p(z,x|\theta)\right)\right]}_{-\,\mathrm{ELBO}(\lambda;\theta)\leq -\log p(x|\theta)} - \underbrace{\log p(x|\theta)}_{\text{not constant wrt }\theta}\end{aligned}$$

Note that in the second step we optimise a bound on the likelihood function (not the actual likelihood function).

This can lead to undesirable effects: if your joint distribution does not exploit the latent variable at all, i.e., $I(X;Z)=0$, then the first step has a trivial minimum, i.e., $q(z|x,\lambda)=p(z)$ for every $x\in\mathcal{X}$.

A special case of this algorithm is EM (see appendix).

A related algorithm that predates variational EM is wake-sleep (WS; Hinton et al., 1995). In addition to updating θ on a bound (same bound, the ELBO), it estimates λ on a different variational objective, one that only holds under the assumption that $p(x|\theta)$ perfectly matches the data generating process (see appendix).

Variational Inference Learning (NVIL)

Gradient-based training of a probabilistic model with NN likelihood and an approximation to its posterior via amortised variational inference.

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

As in variational EM, we use the ELBO to estimate both the probabilistic model and the posterior approximation, but do so jointly.

NVIL's original paper (Mnih and Gregor, 2014).

Generative model

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

- a document $x = (x_1, \ldots, x_I)$ consists of I 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.

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

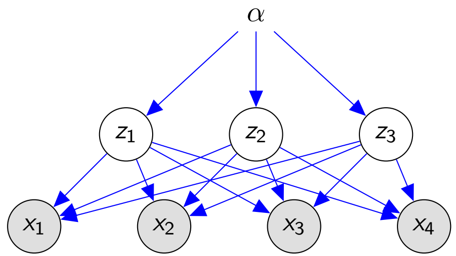
Here $0 < \alpha < 1$ specifies a Bernoulli prior (assume fixed) and $f(\cdot;\theta)$ is a function computed by an NN

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

To keep the slide simple, we use a very shallow NN.

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

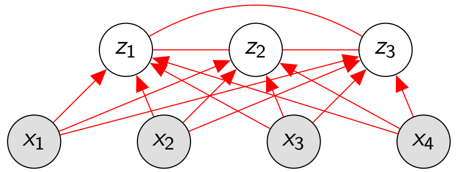
Example Model



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, as well as on α.

Example Model

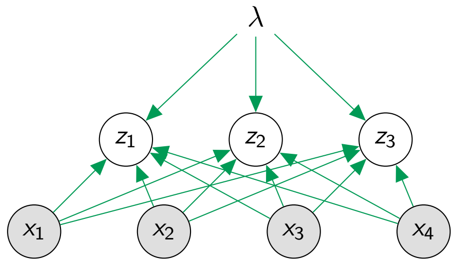


Posterior: latent variables are marginally dependent.

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, as well as on α.

Mean Field Amortised 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)

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

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

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

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

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.

Our **goal** for the rest of this section is to compute $\boldsymbol{\nabla}_\theta$ and $\boldsymbol{\nabla}_\lambda$, or at least unbiased estimates thereof, as we need those for optimisation.

KL Term

KL between D independent Bernoulli distributions is tractable

$$\begin{aligned}\mathrm{KL}\left(q(z|x,\lambda) \mid\mid p(z|\alpha)\right) &= \sum_{d=1}^{D} \mathrm{KL}\left(q(z_d|x,\lambda) \mid\mid p(z_d|\alpha)\right) \\ &= \sum_{d=1}^{D} \mathrm{KL}\left(\mathrm{Bernoulli}\left(b_d\right)\right) \mid\mid \mathrm{Bernoulli}\left(\alpha\right)) \\ &= \sum_{d=1}^{D} b_d \log \frac{b_d}{\alpha} + (1-b_d) \log \frac{1-b_d}{1-\alpha}\end{aligned}$$

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 $\mathrm{KL}\left(Z_d|X=x,\lambda \mid\mid Z_d|\alpha\right)$ 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 depended 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 depended 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 depended 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 depended 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.

MC is not Differentiable

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

Unfortunately, we cannot turn to MC either, as we can only MC estimate expected values.

Outline

1. Overview
2. Modelling Random Experiments
   - Modelling Observed Data
   - Modelling Unobserved Data
3. Approximate Inference
4. Learning
   - Score Function Estimator
   - Reparameterised Gradients
   - Sparse Gradients
5. Diagnostics

Score Function Estimator

We can again use the log identity for derivatives

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

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

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

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

Score Function Estimator

We can now build an MC estimator

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

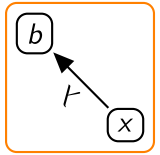
$$= \mathbb{E}_{q(z|x,\lambda)}\left[\log p(x|z,\theta)\frac{\partial}{\partial \lambda}\log q(z|x,\lambda)\right]$$

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

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

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

Computation Graph

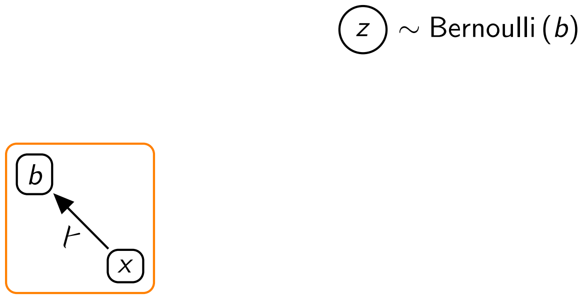


inference network

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

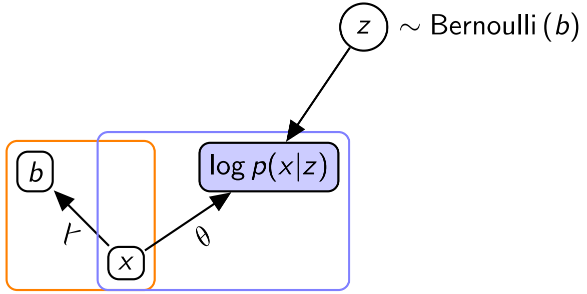


inference network

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

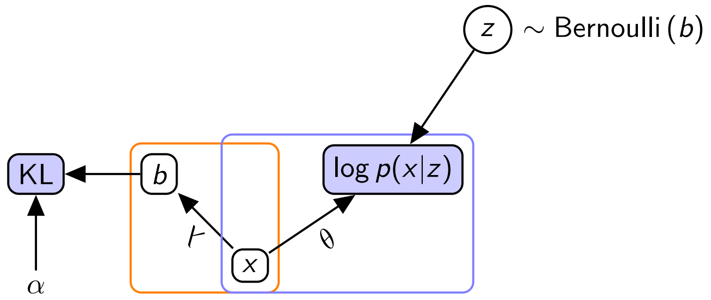


inference network generative network

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;

Computation Graph



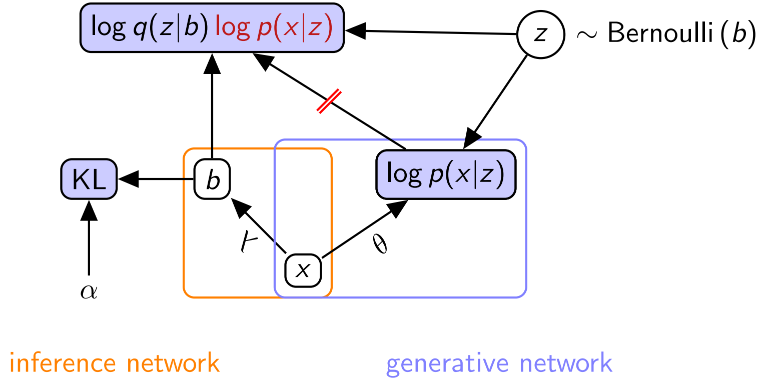
inference network

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 $\text{KL}\left(q(z|x, \lambda) \mid\mid p(z|\alpha)\right)$, whose gradient we need in order to update the inference model;

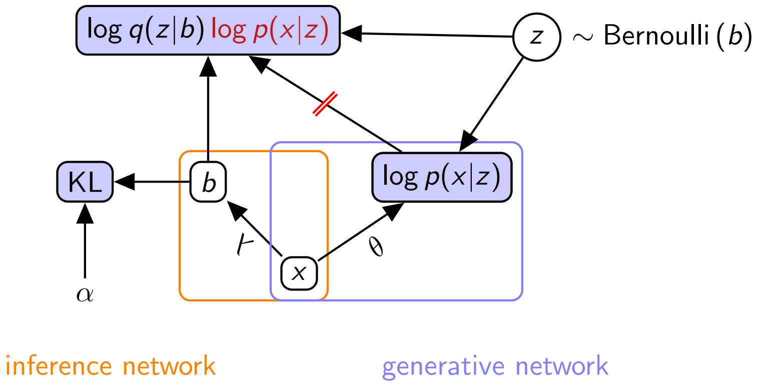
Computation Graph



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;
- finally, to update the inference model we also need the score function estimator, which is $\log p(x|z,\theta)\boldsymbol{\nabla}_\lambda \log q(z|x,\lambda)$; to obtain that gradient using autodiff we need to get a gradient for $\log q(z|x,\lambda)$ and scale it by $\log p(x|z,\theta)$.

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;
- finally, to update the inference model we also need the score function estimator, which is $\log p(x|z,\theta)\boldsymbol{\nabla}_\lambda \log q(z|x,\lambda)$; to obtain that gradient using autodiff we need to get a gradient for $\log q(z|x,\lambda)$ and scale it by $\log p(x|z,\theta)$.

Stochastic Surrogate Objectives

A computation node whose gradient estimates the gradient we want:

$$\log p(x|z,\theta) - \mathrm{KL}\left(q(z|x,\lambda) \mid\mid p(z|\alpha)\right) + \underbrace{\log p(x|z,\cancel{\theta})}_{\text{'detached'}} \log q(z|x,\lambda)$$

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 backpropagation). For brevity, we will denote this by crossing the parameter out (e.g., $\cancel{\theta}$).

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

- Estimate mean and variance of reward based on previous observations: $\hat{\mu}$ and $\hat{\sigma^2}$.
- $\hat{r}(z) = \frac{\log p(x|z,\theta) - \hat{\mu}}{\hat{\sigma^2}}$

It can be shown that

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

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.

Every function c independent of the sample z is a baseline for $\nabla_\lambda \log q(z|x,\lambda)$. You can find a proof in Appendix.

Can you think of quantities other than a moving average? Guiding principle: you want baselines to 'correlate with' or 'be predictive of' the learning signal / reward.

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.

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.

Outline

1 Overview

2 Modelling Random Experiments
- Modelling Observed Data
- Modelling Unobserved Data

3 Approximate Inference

4 Learning
- Score Function Estimator
- Reparameterised Gradients
- Sparse Gradients

5 Diagnostics

Gradient Estimators

Basic problem: we want to differentiate an expected value wrt λ

$$\frac{\partial}{\partial \lambda}\mathbb{E}_{f_{Z|\lambda}}[\psi(z)] \qquad \text{e.g., } \psi(z) := \log p(x|z,\theta)$$
$$f_{Z|\lambda}(z) := q(z|x,\lambda)$$

but the distribution of Z depends on λ.

We have met the SFE and the reparameterised gradient estimator:

$$\mathbb{E}_{f_\lambda(z)}\Bigg[\underbrace{\psi(z)\frac{\partial}{\partial\lambda}\log f_{Z|\lambda}(z)}_{\hat{g}_{\text{sfe}}}\Bigg] = \mathbb{E}_{s(\epsilon)}\Bigg[\underbrace{\frac{\partial}{\partial z}\psi(z)\frac{\partial}{\partial\lambda}t(\epsilon,\lambda)}_{\hat{g}_{\text{rep}}}\Bigg]$$

- $\hat{g}_{\text{sfe}}$ is typically cursed with variance

From VAEs, you know the *reparameterised gradient estimator*

$$\frac{\partial}{\partial\lambda}\mathbb{E}_{f_{Z|\lambda}(z)}[\psi(z)] = \mathbb{E}_{s(\epsilon)}\left[\frac{\partial}{\partial\lambda}\psi(t(\epsilon,\lambda))\right]$$
$$= \mathbb{E}_{s(\epsilon)}\left[\frac{\partial}{\partial z}\psi(z)\frac{\partial}{\partial\lambda}t(\epsilon,\lambda)\right]$$

it takes an invertible and differentiable transformation t such that

$$t(\epsilon,\lambda) \sim f_{Z|\lambda}$$
$$t^{-1}(z,\lambda) \sim s(\epsilon)$$

Goals Understand why there can't be a $\hat{g}_{\text{rep}}$ for discrete rvs. Meet alternatives to SFE.

A general reparameterisation

For univariate Z, what transformation will always absorb the parameters of the density $f_{Z|\lambda}(z)$?

The argument extends to a vector of independent univariate rvs.

A general reparameterisation

For univariate Z, what transformation will always absorb the parameters of the density $f_{Z|\lambda}(z)$?

$$\underbrace{F_{Z|\lambda}(z)}_{\text{cdf}} \sim \mathcal{U}(\underbrace{0,1}_{\text{fixed}})$$

The argument extends to a vector of independent univariate rvs.

A general reparameterisation

For univariate Z, what transformation will always absorb the parameters of the density $f_{Z|\lambda}(z)$?

$$\underbrace{F_{Z|\lambda}(z)}_{\text{cdf}} \sim \mathcal{U}(\underbrace{0,1}_{\text{fixed}})$$

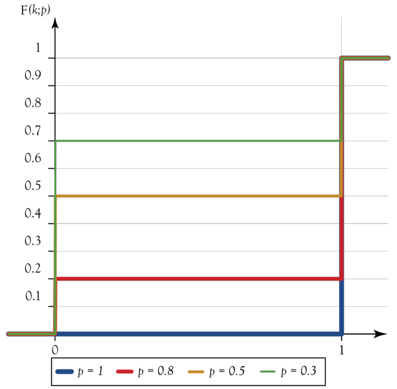
So, if I know the inverse cdf,

$$\epsilon \sim \mathcal{U}(0,1)$$
$$F^{-1}_{Z|\lambda}(\epsilon) \sim Z|\lambda$$

I have access to $\hat{g}_{\text{rep}}$.

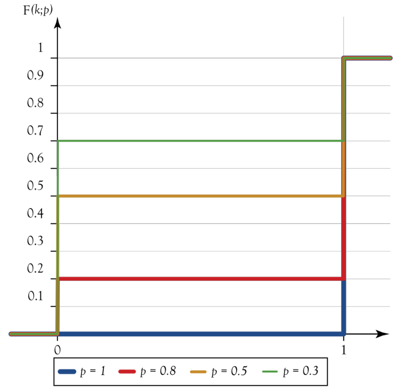
The argument extends to a vector of independent univariate rvs.

Let's reparameterise a Bernoulli



$$Z \sim \text{Bernoulli}(p)$$

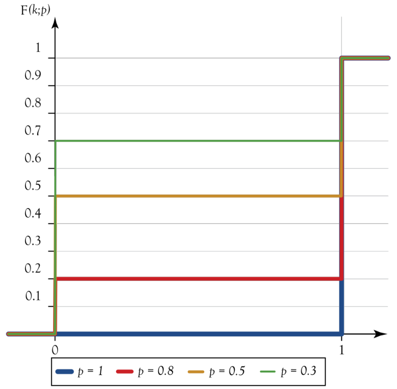
Let's reparameterise a Bernoulli



$$Z \sim \text{Bernoulli}(p)$$

$$\begin{aligned} F^{-1}_{Z|p}(\epsilon) &= \begin{cases} 1 & \text{if } \epsilon < p \\ 0 & \text{otherwise} \end{cases} \\ &= \mathbb{1}_{(0,p)}(\epsilon) \end{aligned}$$

Let’s reparameterise a Bernoulli



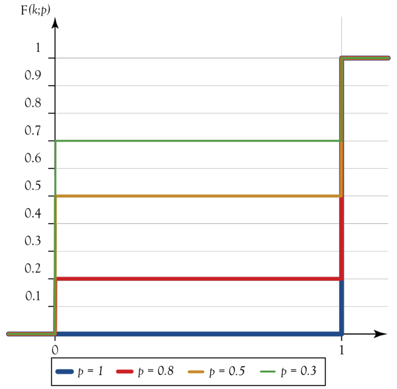
$$Z \sim \text{Bernoulli}(p)$$

$$F^{-1}_{Z|p}(\epsilon) = \begin{cases} 1 & \text{if } \epsilon < p \\ 0 & \text{otherwise} \end{cases}$$

$$= \mathbb{1}_{(0,p)}(\epsilon)$$

How about $\frac{\partial}{\partial p} F^{-1}_{Z|p}(\epsilon)$?

Let's reparameterise a Bernoulli



$$Z \sim \text{Bernoulli}(p)$$

$$F^{-1}_{Z|p}(\epsilon) = \begin{cases} 1 & \text{if } \epsilon < p \\ 0 & \text{otherwise} \end{cases}$$
$$= \mathbb{1}_{(0,p)}(\epsilon)$$

How about $\frac{\partial}{\partial p} F^{-1}_{Z|p}(\epsilon)$? Mostly 0, sometimes undefined!

Discrete case

Discrete variables do not admit a differentiable reparameterisation. The derivatives of the inverse cdf are either 0 or undefined :/

STE's original paper (Bengio et al., 2013).

There are other pseudo-gradients in the literature, for example for relaxed combinatorial random variables (Peng et al., 2018; Mihaylova et al., 2021).

Discrete case

Discrete variables do not admit a differentiable reparameterisation. The derivatives of the inverse cdf are either 0 or undefined :/

The score function estimator is fully general, but very noisy.

STE's original paper (Bengio et al., 2013).

There are other pseudo-gradients in the literature, for example for relaxed combinatorial random variables (Peng et al., 2018; Mihaylova et al., 2021).

Discrete case

Discrete variables do not admit a differentiable reparameterisation. The derivatives of the inverse cdf are either 0 or undefined :/

The score function estimator is fully general, but very noisy.

How about we fake a Jacobian and call it a pseudo-gradient?

$$J_t(\epsilon, \lambda) = \text{diag}(\mathbf{1})$$

This is the ingredient behind the straight-through estimator (STE).

STE's original paper (Bengio et al., 2013).

There are other pseudo-gradients in the literature, for example for relaxed combinatorial random variables (Peng et al., 2018; Mihaylova et al., 2021).

Bernoulli-STE

Consider a VAE where $q(z|x) = \text{Bern}(z|g(x;\lambda))$.

Bernoulli-STE

Consider a VAE where $q(z|x) = \mathrm{Bern}(z|g(x;\lambda))$.
We sample z via a reparameterisation that absorbs λ:

$$\epsilon \sim \mathcal{U}(0,1) \qquad p = g(x;\lambda) \qquad z = \underbrace{\mathbb{1}_{(0,p)}(\epsilon)}_{t(\epsilon,\lambda)}$$

Bernoulli-STE

Consider a VAE where $q(z|x) = \text{Bern}(z|g(x;\lambda))$.
We sample z via a reparameterisation that absorbs λ:

$$\epsilon \sim \mathcal{U}(0,1) \qquad p = g(x;\lambda) \qquad z = \underbrace{\mathbb{1}_{(0,p)}(\epsilon)}_{t(\epsilon,\lambda)}$$

Optimising the ELBO via reparameterised samples requires

$$\hat{g}_{\text{rep}} = \frac{\partial}{\partial\lambda}\log p(x|z=t(\epsilon,\lambda)) = \frac{\partial}{\partial z}\log p(x|z)\frac{\partial}{\partial\lambda}t(\epsilon,\lambda)$$

Bernoulli-STE

Consider a VAE where $q(z|x) = \text{Bern}(z|g(x;\lambda))$.
We sample z via a reparameterisation that absorbs λ:

$$\epsilon \sim \mathcal{U}(0,1) \qquad p = g(x;\lambda) \qquad z = \underbrace{\mathbb{1}_{(0,p)}(\epsilon)}_{t(\epsilon,\lambda)}$$

Optimising the ELBO via reparameterised samples requires

$$\hat{g}_{\text{rep}} = \frac{\partial}{\partial\lambda}\log p(x|z=t(\epsilon,\lambda)) = \frac{\partial}{\partial z}\log p(x|z)\frac{\partial}{\partial\lambda}t(\epsilon,\lambda)$$

Let's use our *pseudo gradient*

$$\hat{g}_{\text{ste}} := \frac{\partial}{\partial\lambda}t(\epsilon,\lambda) = \frac{\partial}{\partial\lambda}g(x;\lambda)\overset{:=1}{\cancel{\frac{\partial}{\partial p}\mathbb{1}_{(0,p)}(\epsilon)}}$$

Concrete Distribution

We can sample from a Categorical distribution via

$$\begin{aligned} \epsilon_k &\sim \text{Gumbel}(0,1) \\ \underbrace{\arg\max_k \; \{\lambda_k + \epsilon_k\}_{k=1}^K}_{z=t(\epsilon,\lambda)} &\sim \text{Cat}(\text{softmax}(\lambda)) \end{aligned}$$

Concrete distribution (Maddison et al., 2017), Gumbel-Softmax distribution (Jang et al., 2017).

Concrete Distribution

We can sample from a Categorical distribution via

$$\begin{aligned} \epsilon_k &\sim \text{Gumbel}(0,1) \\ \underbrace{\arg\max_k \; \{\lambda_k + \epsilon_k\}_{k=1}^K}_{z=t(\epsilon,\lambda)} &\sim \text{Cat}(\text{softmax}(\lambda)) \end{aligned}$$

The problem is that $t(\epsilon, \lambda)$ is not differentiable, but note

$$\text{softmax}\left(\frac{\lambda+\epsilon}{\tau}\right) \to \text{onehot}(z) \qquad \text{as } \tau \to 0$$

and now the transformation is differentiable, but the outcome is dense. For sparsity, use (biased) STE.

Concrete distribution (Maddison et al., 2017), Gumbel-Softmax distribution (Jang et al., 2017).

Mixed Binary Variables

An alternative to biased gradients is to change the model to employ continuous random variables that take on sparse outcomes with non-zero probability mass.

Example: sample $\zeta \sim \mathcal{N}(0,1)$ and rectify the sample via hardsigmoid $z = \min(1, \max(0, z))$.

Then $\Pr(Z \in \{0\}) =$

Spike-and-slab (Rolfe, 2017); HardConcrete (Louizos et al., 2018); Hard-Kumaraswamy (Bastings et al., 2019).

Applications to interpretability (Voita et al., 2019; Cao et al., 2020; Ataman et al., 2020)

Extensions to multivariate draws are not easy (but we have an awesome pre-print coming up, stay tuned!).

Mixed Binary Variables

An alternative to biased gradients is to change the model to employ continuous random variables that take on sparse outcomes with non-zero probability mass.

Example: sample $\zeta \sim \mathcal{N}(0,1)$ and rectify the sample via hardsigmoid $z = \min(1, \max(0, z))$.

Then $\Pr(Z \in \{0\}) = \Phi(0)$, similarly $\Pr(Z \in \{1\}) =$

Spike-and-slab (Rolfe, 2017); HardConcrete (Louizos et al., 2018); Hard-Kumaraswamy (Bastings et al., 2019).

Applications to interpretability (Voita et al., 2019; Cao et al., 2020; Ataman et al., 2020)

Extensions to multivariate draws are not easy (but we have an awesome pre-print coming up, stay tuned!).

Mixed Binary Variables

An alternative to biased gradients is to change the model to employ continuous random variables that take on sparse outcomes with non-zero probability mass.

Example: sample $\zeta \sim \mathcal{N}(0, 1)$ and rectify the sample via hardsigmoid $z = \min(1, \max(0, z))$.

Then $\Pr(Z \in \{0\}) = \Phi(0)$, similarly $\Pr(Z \in \{1\}) = 1 - \Phi(1)$

When $\zeta < 0$ or $\zeta > 1$ the derivative of hardsigmoid is 0, when $0 < \zeta < 1$ the derivative is 1. Hardsigmoid has undefined derivatives for $\zeta = 0$ and $\zeta = 1$, but we will never sample those.

If we had a parameterised Gaussian, we could sample with a reparameterisation and learn the Gaussian parameters.

Spike-and-slab (Rolfe, 2017); HardConcrete (Louizos et al., 2018); Hard-Kumaraswamy (Bastings et al., 2019).

Applications to interpretability (Voita et al., 2019; Cao et al., 2020; Ataman et al., 2020)

Extensions to multivariate draws are not easy (but we have an awesome pre-print coming up, stay tuned!).

Outline

1 Overview

2 Modelling Random Experiments
- Modelling Observed Data
- Modelling Unobserved Data

3 Approximate Inference

4 Learning
- Score Function Estimator
- Reparameterised Gradients
- Sparse Gradients

5 Diagnostics

Latent Computation Graphs

Estimators built on reparameterisation require

- z to be of some fixed finite dimensionality
- the decoder's computation graph must be independent of z.

Some composition functions are parameterised by their inputs (e.g., a tree-LSTM), they are dynamic computation graphs controlled by the discrete latent.

STE is not an option, so we are back to SFE. Or are we?

$\hat{g}_{\text{rep}}$ differentiates the decoder wrt z

$$\hat{g}_{\text{rep}} = \frac{\partial}{\partial z}\psi(z) \times \frac{\partial}{\partial \lambda}t(\epsilon, \lambda)$$

This cannot work when the computation graph of ψ depends on z. For example, a tree-LSTM updates its states following a depth-first traversal of an input tree.

Parameterise for Tractability

We can use sparse projections to the probability simplex to parameterise discrete distributions that assign 0 probability mass to most of the outcomes in their supports.

$$
\begin{aligned}
&\nabla_\lambda \mathbb{E}_{q(z|x,\lambda)}[\log p(x, z|\theta) - \log q(z|x, \lambda)] \\
=&\nabla_\lambda q(c_1|x, \lambda)(\log p(x, c_1|\theta) - \log q(c_1|x, \lambda)) \\
&+ \dots \\
&+ \nabla_\lambda q(c_k|x, \lambda)(\log p(x, c_k|\theta) - \log q(c_k|x, \lambda)) \\
&+ \nabla_\lambda \sum_{z=c_{k+1}}^{c_K} \underbrace{q(z|x, \lambda)}_{=0}(\log p(x, z|\theta) - \log q(z|x, \lambda))
\end{aligned}
$$

Effectively, we have an inference network that parameterises a model whose support is small enough for *enumeration*.

Sparse projections: (Martins and Astudillo, 2016; Niculae et al., 2018a)

Latent dynamic computation graphs: (Niculae et al., 2018b)

Sparse marginals: (Correia et al., 2020)

Summary

Learning discrete LVMs poses challenges for gradient estimation, in particular, gradients of the inference network are challenging.

SFE is the most general, it requires tractable pmf and sampling, nothing else. It is too noisy to be useful without variance reduction techniques.

STE requires a relaxation of the decoder and introduces biases, violating the requirements for stochastic optimisation.

We can mix a pmf and a pdf to obtain reparameterised and unbiased gradients for a sparse rv. This is difficult to extend beyond variables distributed on $[0, 1]$ and, like STE, requires relaxing the decoder.

Sparse parameterisation of the inference model leads to sparse gradients with many terms evaluating trivially to zero. Enumeration dispenses with relaxations and works for combinatorial variables.

Outline

1. Overview
2. Modelling Random Experiments
   - Modelling Observed Data
   - Modelling Unobserved Data
3. Approximate Inference
4. Learning
   - Score Function Estimator
   - Reparameterised Gradients
   - Sparse Gradients
5. Diagnostics

I trained my first discrete LVM, but does it work?

Well, you gotta know where to look :-)

- Track validation ELBO, distortion (D), and rate (R).
- $I(X; Z) \leq R$, if rate is low the latent is not informative.
- For a generative model, $E_{x\sim\mathcal{D}}[p(z|x)]$ should match the prior. Thus for an approximate posterior it is worth tracking K, which should be small. A visual check often suffices (plot a histogram of prior samples against posterior samples for the entire dev set).
- Track importance sampling estimates of the model's log-likelihood (after all, this is what MLE would have optimised).
- Good NLL does not say much: certain likelihood functions do not need any additional flexibility to model the data well.

We are looking for models that have large ELBO, small NLL, large R, and very small K.

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

Hoffman and Johnson (2016); Alemi et al. (2018); Poole et al. (2019)

Look for Failure Modes

Decoding prior samples should cover enough of the data space.

Decoding posterior samples should preserve some appreciable aspect of the seed data.

Conversely, $\mathrm{KL}\left(q(z|x,\lambda) \,||\, q(z|x',\lambda)\right)$ should be smaller the more related the two data samples are.

You can use nearest neighbour retrieval to help automate some of this analysis.

Final Remarks

- Probabilistic models are extremely flexible tools.
- They are interesting precisely because we can make choices about unobserved aspects of the data.
- Discrete latent variables are oftentimes key to revealing interpretable structure, or to imposing some interpretable structure on a joint distribution.
- Learning discrete LVMs is challenging, but recent years have seen amazing progress.
- Join the party! Apply these models, extend them, discover problems with their estimation/evaluation, investigate solutions.
- Avoid approaching LVMs wondering whether they will beat some non-LVM NN. If such NN exists, then you are probably looking at an aspect of the problem that does not require latent variables.

What Next?

- Check the Appendix.
- For more material, check
  https://vitutorial.github.io/classes/
- We also have some coding exercises
  https://github.com/vitutorial/exercises
- Check this great tutorial by our friends from DeepSPIN
  https://deep-spin.github.io/tutorial/

See you around!

References I

Alexander Alemi, Ben Poole, Ian Fischer, Joshua Dillon, Rif A. Saurous, and Kevin Murphy. Fixing a Broken ELBO. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 159–168, Stockholmsmässan, Stockholm Sweden, July 2018. PMLR. URL http://proceedings.mlr.press/v80/alemi18a.html.

Duygu Ataman, Wilker Aziz, and Alexandra Birch. A Latent Morphology Model for Open-Vocabulary Neural Machine Translation. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJxSI1SKDH.

References II

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.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

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.

References III

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.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Nicola De Cao, Michael Schlichtkrull, Wilker Aziz, and Ivan Titov. How do Decisions Emerge across Layers in Neural Models? Interpretation with Differentiable Masking. In *EMNLP*, 2020.

Gonçalo M. Correia, Vlad Niculae, Wilker Aziz, and André F. T. Martins. Efficient Marginalization of Discrete and Structured Latent Variables via Sparsity. In *NeurIPS*, 2020.

References IV

Caio Corro and Ivan Titov. Differentiable Perturb-and-Parse: Semi-Supervised Parsing with a Structured Variational Autoencoder. In *ICLR*, September 2018. URL https://openreview.net/forum?id=BJlgNh0qKQ.

Yuntian Deng, Yoon Kim, Justin Chiu, Demi Guo, and Alexander Rush. Latent Alignment and Variational Attention. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9712–9724. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8179-latent-alignment-and-variational-attention.pdf.

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.

References V

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.

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.

References VI

Serhii Havrylov, Germán Kruszewski, and Armand Joulin. Cooperative learning of disjoint syntax and semantics. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1118–1128, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1115. URL https://www.aclweb.org/anthology/N19-1115.

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.

Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1, page 2, 2016.

References VII

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. Toward controlled generation of text. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 06–11 Aug 2017. URL http://proceedings.mlr.press/v70/hu17e.html.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

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.

References VIII

Kazuya Kawakami, Chris Dyer, and Phil Blunsom. Learning to discover, ground and use words with segmental neural language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6429–6441, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1645. URL https://www.aclweb.org/anthology/P19-1645.

Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. In *ICLR*, 2017.

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 IX

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models*. MIT Press, 2009.

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 X

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning Sparse Neural Networks through L_0 Regularization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1Y8hhg0b.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*, 2017.

Andre Martins and Ramon Astudillo. From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1614–1623, New York, New York, USA, June 2016. PMLR. URL http://proceedings.mlr.press/v48/martins16.html.

References XI

Tsvetomila Mihaylova, Vlad Niculae, and André FT Martins. Understanding the mechanics of spigot: Surrogate gradients for latent structure learning. In *EMNLP*, 2021.

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.

References XII

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.

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.

References XIII

Vlad Niculae, Andre Martins, Mathieu Blondel, and Claire Cardie. SparseMAP: Differentiable Sparse Structured Inference. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3799–3808, Stockholmsmässan, Stockholm Sweden, July 2018a. PMLR. URL http://proceedings.mlr.press/v80/niculae18a.html.

Vlad Niculae, André F. T. Martins, and Claire Cardie. Towards Dynamic Computation Graphs via Sparse Latent Structure. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 905–911, Brussels, Belgium, October 2018b. Association for Computational Linguistics. doi: 10.18653/v1/D18-1108. URL https://www.aclweb.org/anthology/D18-1108.

References XIV

Hao Peng, Sam Thomson, and Noah A. Smith. Backpropagating through structured argmax using a SPIGOT. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1863–1873, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1173. URL https://www.aclweb.org/anthology/P18-1173.

Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180. PMLR, 2019.

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.

References XV

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.

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.

References XVI

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. Publisher: JSTOR.

Jason Tyler Rolfe. Discrete variational autoencoders. In *ICLR*, 2017.

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.

Akash Srivastava and Charles Sutton. Autoencoding variational inference for topic models. In *ICLR*, 2017.

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.

References XVII

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.

Elena Voita, Rico Sennrich, and Ivan Titov. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4396–4406, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1448. URL https://www.aclweb.org/anthology/D19-1448.

References XVIII

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.

Dani Yogatama, Phil Blunsom, Chris Dyer, Edward Grefenstette, and Wang Ling. Learning to compose words into sentences with reinforcement learning. In *ICLR*, 2017.

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.

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

$$\begin{tikzcd}
 & & z_3 \arrow[dl, "\theta"'] \arrow[dr, "\theta"] & & \\
 & z_1 \arrow[dl, "\theta"'] \arrow[dr, "\theta"] & & z_2 \arrow[dl, "\theta"'] \arrow[dr, "\theta"] & \\
x_1 & & x_2 & & x_3
\end{tikzcd}$$

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.

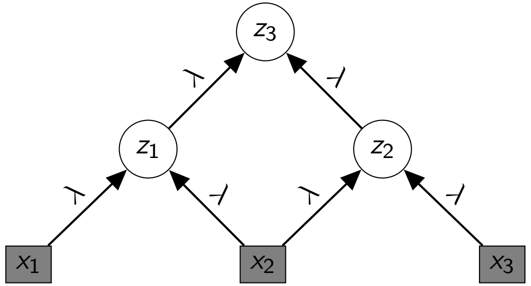
For example, this NN has 3 layers:

- 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 probability of data and latents $p(x, z|\theta)$

Wake-sleep Training

Wake Phase

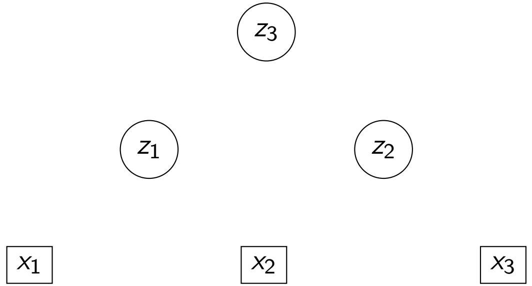
- Use inference network to sample hidden unit setting z from $q(z|x, \lambda)$
- Update generation parameters θ to maximize joint log probability of data and latents $p(x, z|\theta)$

Sleep Phase

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

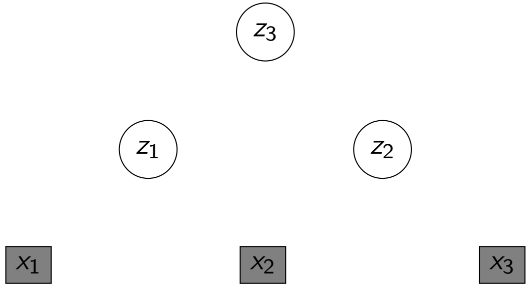
Wake Phase Sampling

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



Wake Phase Sampling

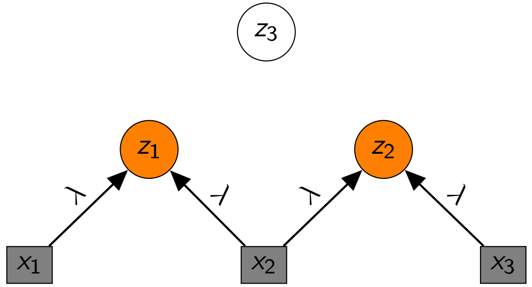
Sampling $z \sim q(z|x, \lambda)$



- Observe x

Wake Phase Sampling

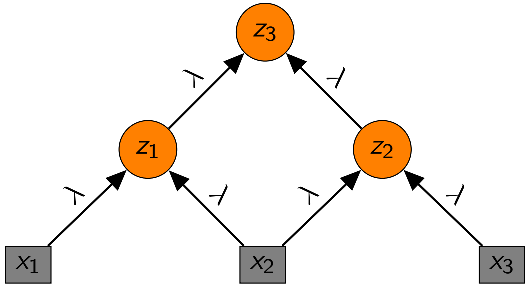
Sampling $z \sim q(z|x, \lambda)$



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

Wake Phase Sampling

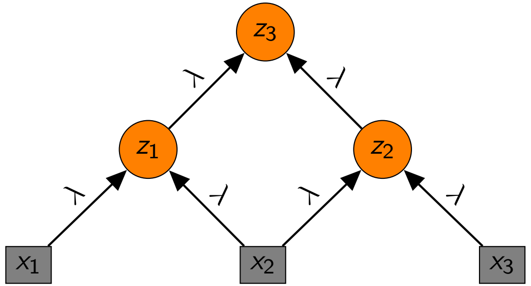
Sampling $z \sim q(z|x, \lambda)$



- 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

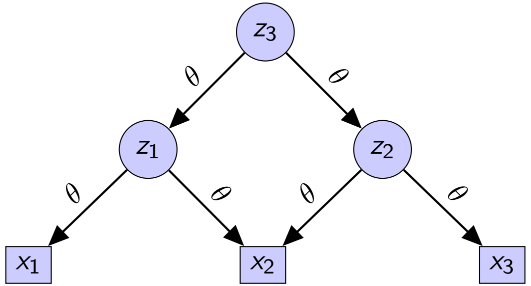
Sampling $z \sim q(z|x, \lambda)$



- 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

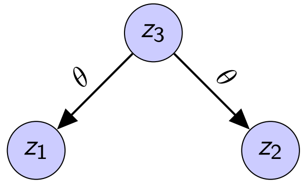
Sampling $(z, \tilde{x}) \sim p(x, z|\theta)$

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

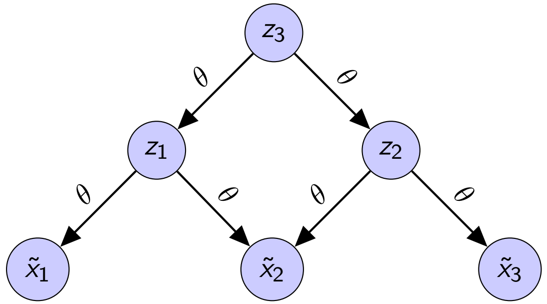


We do a stochastic forward pass through the generative model sampling our random variables.

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

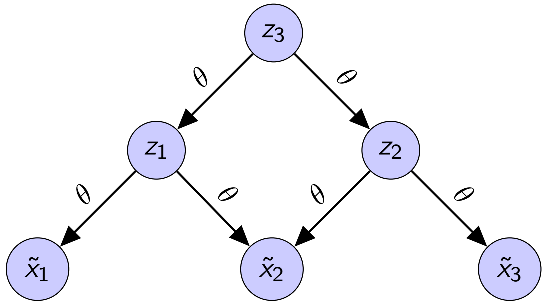


We do a stochastic forward pass through the generative model sampling our random variables.

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

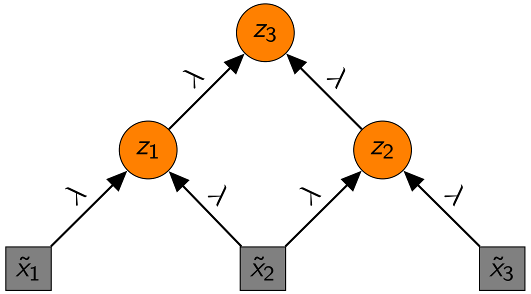


We do a stochastic forward pass through the generative model sampling our random variables.

- 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

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

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

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

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

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

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

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

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 $\mathrm{KL}\left(q(z|x,\lambda)\ ||\ p(z|x,\theta)\right)$.

Sleep Phase (Convenient) Objective

Flip the direction of the KL

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

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

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

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

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

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

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

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

$$\begin{aligned} z &\sim p(z|\theta) \\ \tilde{x} &\sim p(x|z, \theta) \end{aligned}$$

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

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

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

Frequentist VI also optimises the model!

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

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

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

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

Recap: EM Algorithm

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

$= p(z|x)$

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

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 $p(z|x)$.

$$q(z) = p(z|x)$$
$$\mathrm{KL}(q(z) \mid\mid p(z|x)) = 0$$

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.

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

Expected score

The Expectation of the score function is 0.

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

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 reward 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!