Deep Latent-Variable Models of Natural Language

Yoon Kim, Sam Wiseman, Alexander Rush

harvardnlp

Tutorial 2018

https://github.com/harvardnlp/DeepLatentNLP

Goal of Latent-Variable Modeling

Probabilistic models provide a declarative language for specifying prior knowledge and structural relationships in the context of unknown variables.

Makes it easy to specify:

- Known interactions in the data
- Uncertainty about unknown factors
- Constraints on model properties

Goal of Latent-Variable Modeling

Probabilistic models provide a declarative language for specifying prior knowledge and structural relationships in the context of unknown variables.

Makes it easy to specify:

- Known interactions in the data
- Uncertainty about unknown factors
- Constraints on model properties

Latent-Variable Modeling in NLP

Long and rich history of latent-variable models of natural language.

Major successes include, among many others:

- Statistical alignment for translation
- Document clustering and topic modeling
- Unsupervised part-of-speech tagging and parsing

Goals of Deep Learning

Toolbox of methods for learning rich, non-linear data representations through numerical optimization.

Makes it easy to fit:

- Highly-flexible predictive models
- Transferable feature representations
- Structurally-aligned network architectures

Goals of Deep Learning

Toolbox of methods for learning rich, non-linear data representations through numerical optimization.

Makes it easy to fit:

- Highly-flexible predictive models
- Transferable feature representations
- Structurally-aligned network architectures

Deep Learning in NLP

Current dominant paradigm for NLP.

Major successes include, among many others:

- Text classification
- Neural machine translation
- NLU Tasks (QA, NLI, etc)

Tutorial: Deep Latent-Variable Models for NLP

- How should a contemporary ML/NLP researcher reason about latent-variables?
- What unique challenges come from modeling text with latent variables?
- What techniques have been explored and shown to be effective in recent papers?

We explore these through the lens of *variational inference*.

Tutorial: Deep Latent-Variable Models for NLP

- How should a contemporary ML/NLP researcher reason about latent-variables?
- What unique challenges come from modeling text with latent variables?
- What techniques have been explored and shown to be effective in recent papers?

We explore these through the lens of *variational inference*.

Tutorial Take-Aways

1. A collection of deep latent-variable models for NLP
2. An understanding of a variational objective
3. A toolkit of algorithms for optimization
4. A formal guide to advanced techniques
5. A survey of example applications
6. Code samples and techniques for practical use

Tutorial Non-Objectives

Not covered (for time, not relevance):

- Many classical latent-variable approaches.
- Undirected graphical models such as MRFs
- Non-likelihood based models such as GANs
- Sampling-based inference such as MCMC.
- Details of deep learning architectures.

What are deep networks?

Deep networks are parameterized non-linear functions; They transform input $\mathbf{z}$ into features $\boldsymbol{h}$ using parameters π.

Important examples: The multilayer perceptron,

$$\boldsymbol{h} = \mathrm{MLP}(\mathbf{z}; \pi) = \boldsymbol{V}\sigma(\boldsymbol{W}\mathbf{z} + \boldsymbol{b}) + \boldsymbol{a} \qquad \pi = \{\boldsymbol{V}, \boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}\},$$

The recurrent neural network, which maps a sequence of inputs $\mathbf{z}_{1:T}$ into a sequence of features $\boldsymbol{h}_{1:T}$,

$$\boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{h}_{t-1}, \mathbf{z}_t; \pi) = \sigma(\boldsymbol{U}\mathbf{z}_t + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}) \qquad \pi = \{\boldsymbol{V}, \boldsymbol{U}, \boldsymbol{b}\}$$

.

What are deep networks?

Deep networks are parameterized non-linear functions; They transform input $\mathbf{z}$ into features $\boldsymbol{h}$ using parameters π.

Important examples: The multilayer perceptron,

$$\boldsymbol{h} = \mathrm{MLP}(\mathbf{z}; \pi) = \boldsymbol{V}\sigma(\boldsymbol{W}\mathbf{z} + \boldsymbol{b}) + \boldsymbol{a} \qquad \pi = \{\boldsymbol{V}, \boldsymbol{W}, \boldsymbol{a}, \boldsymbol{b}\},$$

The recurrent neural network, which maps a sequence of inputs $\mathbf{z}_{1:T}$ into a sequence of features $\boldsymbol{h}_{1:T}$,

$$\boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{h}_{t-1}, \mathbf{z}_t; \pi) = \sigma(\boldsymbol{U}\mathbf{z}_t + \boldsymbol{V}\boldsymbol{h}_{t-1} + \boldsymbol{b}) \qquad \pi = \{\boldsymbol{V}, \boldsymbol{U}, \boldsymbol{b}\}$$

.

What are latent variable models?

Latent variable models give us a joint distribution

$$p(x, z;\, \theta).$$

- x is our observed data
- z is a collection of latent variables
- θ are the deterministic parameters of the model, such as the neural network parameters

- Data consists of N i.i.d samples,

$$p(x^{(1:N)}, z^{(1:N)};\, \theta) = \prod_{n=1}^{N} p(x^{(n)} \mid z^{(n)}; \theta) p(z^{(n)}; \theta).$$

What are latent variable models?

Latent variable models give us a joint distribution

$$p(x, z;\, \theta).$$

- x is our observed data
- z is a collection of latent variables
- θ are the deterministic parameters of the model, such as the neural network parameters
- Data consists of N i.i.d samples,

$$p(x^{(1:N)}, z^{(1:N)};\, \theta) = \prod_{n=1}^{N} p(x^{(n)} \mid z^{(n)}; \theta) p(z^{(n)}; \theta).$$

What are latent variable models?

Latent variable models give us a joint distribution

$$p(x, z;\, \theta).$$

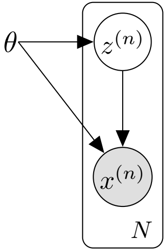
- x is our observed data
- z is a collection of latent variables
- θ are the deterministic parameters of the model, such as the neural network parameters

- Data consists of N i.i.d samples,

$$p(x^{(1:N)}, z^{(1:N)};\, \theta) = \prod_{n=1}^{N} p(x^{(n)} \mid z^{(n)}; \theta) p(z^{(n)}; \theta).$$

Probabilistic Graphical Models

- A directed PGM shows the conditional independence structure.
- By chain rule, latent variable model over observations can be represented as,



$$p(x^{(1:N)}, z^{(1:N)};\, \theta) = \prod_{n=1}^{N} p(x^{(n)} \mid z^{(n)}; \theta) p(z^{(n)}; \theta)$$

- Specific models may factor further.

Posterior Inference

For models $p(x, z; \theta)$, we'll be interested in the *posterior* over latent variables z:

$$p(z \mid x; \theta) = \frac{p(x, z; \theta)}{p(x; \theta)}.$$

Why?

- z will often represent interesting information about our data (e.g., the cluster $x^{(n)}$ lives in, how similar $x^{(n)}$ and $x^{(n+1)}$ are).
- Learning the parameters θ of the model often requires calculating posteriors as a subroutine.
- Intuition: if I know likely $z^{(n)}$ for $x^{(n)}$, I can learn by maximizing $p(x^{(n)} \mid z^{(n)}; \theta)$.

Posterior Inference

For models $p(x, z;\, \theta)$, we'll be interested in the *posterior* over latent variables z:

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

Why?

- z will often represent interesting information about our data (e.g., the cluster $x^{(n)}$ lives in, how similar $x^{(n)}$ and $x^{(n+1)}$ are).
- Learning the parameters θ of the model often requires calculating posteriors as a subroutine.
- Intuition: if I know likely $z^{(n)}$ for $x^{(n)}$, I can learn by maximizing $p(x^{(n)} \,|\, z^{(n)};\, \theta)$.

Problem Statement: Two Views

Deep Models & LV Models are naturally complementary:

- Rich function approximators with modular parts.
- Declarative methods for specifying model constraints.

Deep Models & LV Models are frustratingly incompatible:

- Deep networks make posterior inference intractable.
- Latent variable objectives complicate backpropagation.

Problem Statement: Two Views

Deep Models & LV Models are naturally complementary:

- Rich function approximators with modular parts.
- Declarative methods for specifying model constraints.

Deep Models & LV Models are frustratingly incompatible:

- Deep networks make posterior inference intractable.
- Latent variable objectives complicate backpropagation.

A Language Model

Our goal is to model a sentence, $x_1 \ldots x_T$.

Context: RNN language models are remarkable at this task,

$$x_{1:T} \sim \mathrm{RNNLM}(x_{1:T}; \theta).$$

Defined as,

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \,|\, x_{<t}) = \prod_{t=1}^{T} \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_t)_{x_t}$$

$$\text{where} \quad \boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{h}_{t-1}, \mathbf{x}_{t-1}; \theta)$$

A Language Model

Our goal is to model a sentence, $x_1 \dots x_T$.

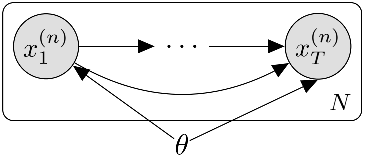
Context: RNN language models are remarkable at this task,

$$x_{1:T} \sim \mathrm{RNNLM}(x_{1:T}; \theta).$$

Defined as,

$$p(x_{1:T}) = \prod_{t=1}^{T} p(x_t \,|\, x_{<t}) = \prod_{t=1}^{T} \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_t)_{x_t}$$

where $\quad \boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{h}_{t-1}, \mathbf{x}_{t-1}; \theta)$



A Collection of Model Archetypes

Focus: semi-supervised or unsupervised learning, i.e. don't just learn the probabilities, but the process. Range of choices in selecting z

1. Discrete LVs z (*Clustering*)
2. Continuous LVs z (*Dimensionality reduction*)
3. Structured LVs z (*Structured learning*)

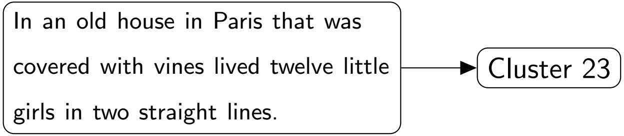
A Collection of Model Archetypes

Focus: semi-supervised or unsupervised learning, i.e. don't just learn the probabilities, but the process. Range of choices in selecting z

1. Discrete LVs z (*Clustering*)
2. Continuous LVs z (*Dimensionality reduction*)
3. Structured LVs z (*Structured learning*)

Model 1: Discrete Clustering

Inference Process:



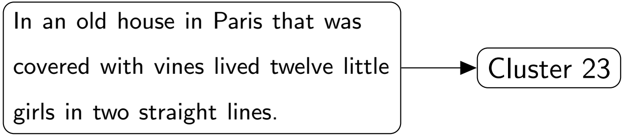
Discrete latent variable models induce a clustering over sentences $x^{(n)}$.

Example uses:

- Document/sentence clustering [Willett 1988; Aggarwal and Zhai 2012].
- Mixture of expert text generation models [Jacobs et al. 1991; Garmash and Monz 2016; Lee et al. 2016]

Model 1: Discrete Clustering

Inference Process:



Discrete latent variable models induce a clustering over sentences $x^{(n)}$.

Example uses:

- Document/sentence clustering [Willett 1988; Aggarwal and Zhai 2012].
- Mixture of expert text generation models [Jacobs et al. 1991; Garmash and Monz 2016; Lee et al. 2016]

Model 1: Discrete - Mixture of Categoricals

Generative process:

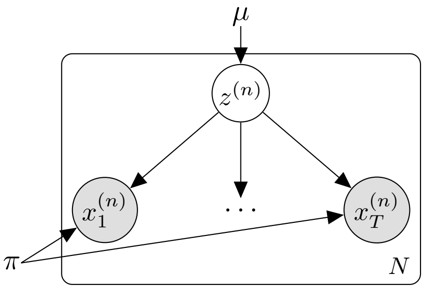
1. Draw cluster $z \in \{1, \dots, K\}$ from a categorical with param μ.
2. Draw word T words x_t from a categorical with word distribution π_z.

Parameters: $\theta = \{\mu \in \Delta^{K-1}, K \times V$ stochastic matrix $\pi\}$

Gives rise to the "Naive Bayes" distribution:

$$p(x, z; \theta) = p(z; \mu) \times p(x \mid z; \pi) = \mu_z \times \prod_{t=1}^{T} \text{Cat}(x_t; \pi)$$
$$= \mu_z \times \prod_{t=1}^{T} \pi_{z, x_t}$$

Model 1: Graphical Model View



$$\prod_{n=1}^{N} p(x^{(n)}, z^{(n)};\, \mu, \pi) = \prod_{n=1}^{N} p(z^{(n)};\, \mu) \times p(x^{(n)} \mid z^{(n)};\, \pi)$$

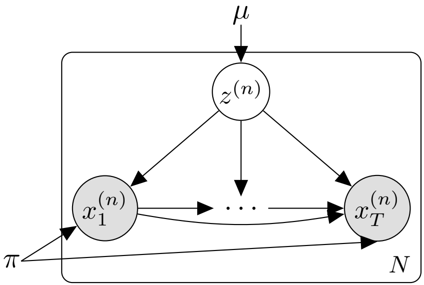
$$= \prod_{n=1}^{N} \mu_{z^{(n)}} \times \prod_{t=1}^{T} \pi_{z^{(n)}, x_t^{(n)}}$$

Deep Model 1: Discrete - Mixture of RNNs

Generative process:

1. Draw cluster $z \in \{1, \dots, K\}$ from a categorical.
2. Draw words $x_{1:T}$ from RNNLM with parameters π_z.

$$p(x, z; \theta) = \mu_z \times \mathrm{RNNLM}(x_{1:T}; \pi_z)$$



Difference Between Models

- Dependence structure:
  - Mixture of Categoricals: x_t independent of other x_j given z.
  - Mixture of RNNs: x_t fully dependent.

  Interesting question: how will this affect the learned latent space?

- Number of parameters:
  - Mixture of Categoricals: $K \times V$.
  - Mixture of RNNs: $K \times d^2 + V \times d$ with RNN with d hidden dims.

Difference Between Models

- Dependence structure:
  - Mixture of Categoricals: x_t independent of other x_j given z.
  - Mixture of RNNs: x_t fully dependent.

  Interesting question: how will this affect the learned latent space?

- Number of parameters:
  - Mixture of Categoricals: $K \times V$.
  - Mixture of RNNs: $K \times d^2 + V \times d$ with RNN with d hidden dims.

Posterior Inference

For both discrete models, can apply Bayes' rule:

$$p(z \mid x; \theta) = \frac{p(z) \times p(x \mid z)}{p(x)}$$

$$= \frac{p(z) \times p(x \mid z)}{\sum_{k=1}^{K} p(z{=}k) \times p(x \mid z{=}k)}$$

- For mixture of categoricals, posterior uses word counts under each π_k.
- For mixture of RNNs, posterior requires running RNN over x for each k.

Posterior Inference

For both discrete models, can apply Bayes' rule:

$$p(z \mid x; \theta) = \frac{p(z) \times p(x \mid z)}{p(x)} = \frac{p(z) \times p(x \mid z)}{\sum_{k=1}^{K} p(z{=}k) \times p(x \mid z{=}k)}$$

- For mixture of categoricals, posterior uses word counts under each π_k.
- For mixture of RNNs, posterior requires running RNN over x for each k.

Posterior Inference

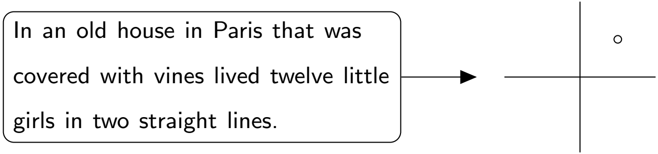
For both discrete models, can apply Bayes' rule:

$$p(z \mid x;\, \theta) = \frac{p(z) \times p(x \mid z)}{p(x)}$$
$$= \frac{p(z) \times p(x \mid z)}{\displaystyle\sum_{k=1}^{K} p(z{=}k) \times p(x \mid z{=}k)}$$

- For mixture of categoricals, posterior uses word counts under each π_k.
- For mixture of RNNs, posterior requires running RNN over x for each k.

Model 2: Continuous / Dimensionality Reduction

Inference Process:



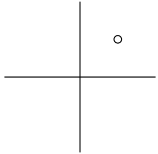
Find a lower-dimensional, well-behaved continuous representation of a sentence. Latent variables in $\mathbb{R}^d$ make distance/similarity easy. Examples:

- Recent work in text generation assumes a latent vector per sentence [Bowman et al. 2016; Yang et al. 2017; Hu et al. 2017].
- Certain sentence embeddings (e.g., Skip-Thought vectors [Kiros et al. 2015]) can be interpreted in this way.

Model 2: Continuous / Dimensionality Reduction

Inference Process:

In an old house in Paris that was covered with vines lived twelve little girls in two straight lines. →



Find a lower-dimensional, well-behaved continuous representation of a sentence.

Latent variables in $\mathbb{R}^d$ make distance/similarity easy. Examples:

- Recent work in text generation assumes a latent vector per sentence [Bowman et al. 2016; Yang et al. 2017; Hu et al. 2017].
- Certain sentence embeddings (e.g., Skip-Thought vectors [Kiros et al. 2015]) can be interpreted in this way.

Model 2: Continuous "Mixture"

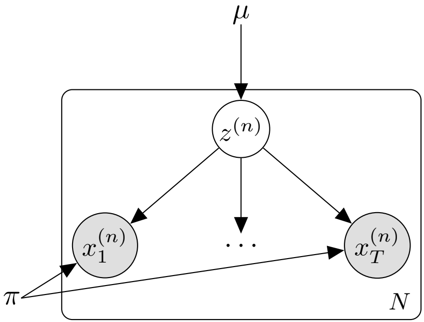
Generative Process:

1. Draw continuous latent variable $\mathbf{z}$ from Normal with param μ.
2. For each t, draw word x_t from categorical with param $\mathrm{softmax}(\boldsymbol{W}\mathbf{z})$.

Parameters: $\theta = \{\mu \in \mathbb{R}^d, \pi\}, \pi = \{\boldsymbol{W} \in \mathbb{R}^{V \times d}\}$

Intuition: μ is a global distribution, $\mathbf{z}$ captures local word distribution of the sentence.

Graphical Model View



Gives rise to the joint distribution:

$$\prod_{n=1}^{N} p(x^{(n)}, z^{(n)};\, \theta) = \prod_{n=1}^{N} p(z^{(n)};\, \mu) \times p(x^{(n)} \mid z^{(n)};\, \pi)$$

Deep Model 2: Continuous "Mixture" of RNNs

Generative Process:

1. Draw latent variable $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I})$.
2. Draw each token x_t from a conditional RNNLM.

RNN is also conditioned on latent $\mathbf{z}$,

$$p(x, \mathbf{z}; \pi, \boldsymbol{\mu}, \boldsymbol{I}) = p(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{I}) \times p(x \mid \mathbf{z}; \pi)$$
$$= \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{I}) \times \text{CRNNLM}(x_{1:T}; \pi, \mathbf{z})$$

where

$$\text{CRNNLM}(x_{1:T}; \pi, \mathbf{z}) = \prod_{t=1}^{T} \text{softmax}(\boldsymbol{W}\boldsymbol{h}_t)_{x_t}$$

$$\boldsymbol{h}_t = \text{RNN}(\boldsymbol{h}_{t-1}, [\mathbf{x}_{t-1}; \mathbf{z}]; \pi)$$

Deep Model 2: Continuous "Mixture" of RNNs

Generative Process:

1. Draw latent variable $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I})$.
2. Draw each token x_t from a conditional RNNLM.

RNN is also conditioned on latent $\mathbf{z}$,

$$\begin{aligned} p(x, \mathbf{z}; \pi, \boldsymbol{\mu}, \boldsymbol{I}) &= p(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{I}) \times p(x \mid \mathbf{z}; \pi) \\ &= \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{I}) \times \mathrm{CRNNLM}(x_{1:T}; \pi, \mathbf{z}) \end{aligned}$$

where

$$\mathrm{CRNNLM}(x_{1:T}; \pi, \mathbf{z}) = \prod_{t=1}^{T} \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_t)_{x_t}$$

$$\boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{h}_{t-1}, [\mathbf{x}_{t-1}; \mathbf{z}]; \pi)$$

Deep Model 2: Continuous "Mixture" of RNNs

Generative Process:

1. Draw latent variable $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{I})$.
2. Draw each token x_t from a conditional RNNLM.

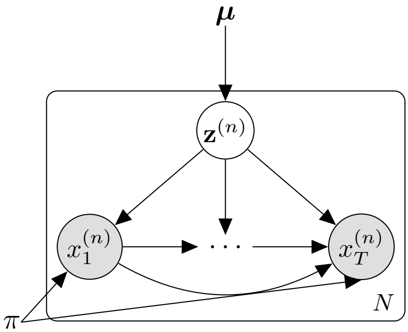
RNN is also conditioned on latent $\mathbf{z}$,

$$p(x, \mathbf{z};\, \pi, \boldsymbol{\mu}, \boldsymbol{I}) = p(\mathbf{z};\, \boldsymbol{\mu}, \boldsymbol{I}) \times p(x \,|\, \mathbf{z};\, \pi)$$
$$= \mathcal{N}(\mathbf{z};\, \boldsymbol{\mu}, \boldsymbol{I}) \times \mathrm{CRNNLM}(x_{1:T};\, \pi, \mathbf{z})$$

where

$$\mathrm{CRNNLM}(x_{1:T};\, \pi, \mathbf{z}) = \prod_{t=1}^{T} \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_t)_{x_t}$$
$$\boldsymbol{h}_t = \mathrm{RNN}(\boldsymbol{h}_{t-1}, [\mathbf{x}_{t-1}; \mathbf{z}]; \pi)$$

Graphical Model View



Posterior Inference

For continuous models, Bayes' rule is harder to compute,

$$p(z \mid x; \theta) = \frac{p(z;\, \mu) \times p(x \mid z;\, \pi)}{\int_z p(z;\, \mu) \times p(x \mid z;\, \pi)\, dz}$$

- Shallow and deep Model 2 variants mirror Model 1 variants exactly, but with continuous z.
- Integral intractable (in general) for both shallow and deep variants.

Posterior Inference

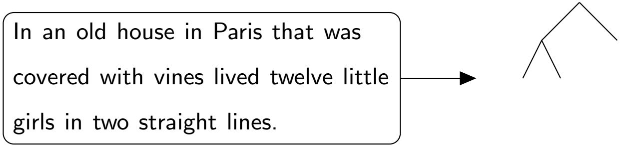
For continuous models, Bayes' rule is harder to compute,

$$p(z \,|\, x; \theta) = \frac{p(z;\, \mu) \times p(x \,|\, z;\, \pi)}{\displaystyle\int_z p(z;\, \mu) \times p(x \,|\, z;\, \pi)\, dz}$$

- Shallow and deep Model 2 variants mirror Model 1 variants exactly, but with continuous z.
- Integral intractable (in general) for both shallow and deep variants.

Model 3: Structure Learning

Inference Process:



Structured latent variable models are used to infer unannotated structure:

- Unsupervised POS tagging [Brown et al. 1992; Merialdo 1994; Smith and Eisner 2005]
- Unsupervised dependency parsing [Klein and Manning 2004; Headden III et al. 2009]

Or when structure is useful for *interpreting* our data:

- Segmentation of documents into topical passages [Hearst 1997]
- Alignment [Vogel et al. 1996]

Model 3: Structured - Hidden Markov Model

Generative Process:

1. For each t, draw $z_t \in \{1, \dots, K\}$ from a categorical with param $\mu_{z_{t-1}}$.
2. Draw observed token x_t from categorical with param π_{z_t}.

Parameters: $\theta = \{K \times K$ stochastic matrix $\mu, K \times V$ stochastic matrix $\pi\}$

Gives rise to the joint distribution:

$$p(x, z;\ \theta) = \prod_{t=1}^{T} p(z_t \,|\, z_{t-1};\ \mu_{z_{t-1}}) \times \prod_{t=1}^{T} p(x_t \,|\, z_t;\ \pi_{z_t})$$
$$= \prod_{t=1}^{T} \mu_{z_{t-1}, z_t} \times \prod_{t=1}^{T} \pi_{z_t, x_t}$$

Model 3: Structured - Hidden Markov Model

Generative Process:

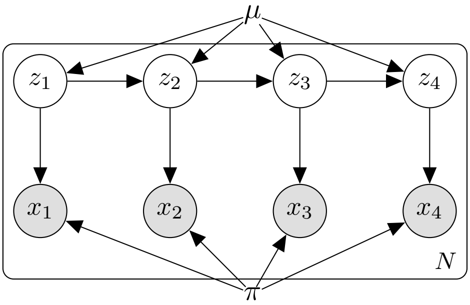
1. For each t, draw $z_t \in \{1, \dots, K\}$ from a categorical with param $\mu_{z_{t-1}}$.
2. Draw observed token x_t from categorical with param π_{z_t}.

Parameters: $\theta = \{K \times K$ stochastic matrix $\mu, K \times V$ stochastic matrix $\pi\}$

Gives rise to the joint distribution:

$$\begin{aligned} p(x, z;\ \theta) &= \prod_{t=1}^{T} p(z_t \mid z_{t-1};\ \mu_{z_{t-1}}) \times \prod_{t=1}^{T} p(x_t \mid z_t;\ \pi_{z_t}) \\ &= \prod_{t=1}^{T} \mu_{z_{t-1}, z_t} \times \prod_{t=1}^{T} \pi_{z_t, x_t} \end{aligned}$$

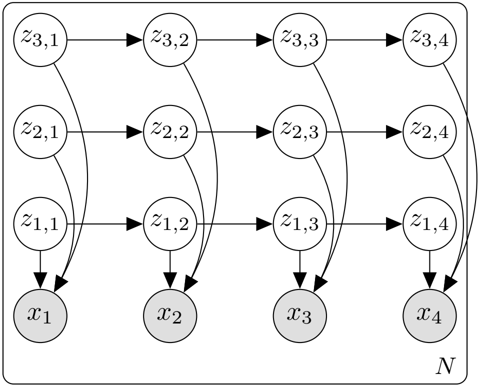
Graphical Model View



$$p(x, z;\ \theta) = \prod_{t=1}^{T} p(z_t \mid z_{t-1};\ \mu_{z_{t-1}}) \times \prod_{t=1}^{T} p(x_t \mid z_t;\ \pi_{z_t})$$

$$= \prod_{t=1}^{T} \mu_{z_{t-1}, z_t} \times \prod_{t=1}^{T} \pi_{z_t, x_t}$$

Further Extension: Factorial HMM



$$p(x, z;\ \theta) = \prod_{l=1}^{L} \prod_{t=1}^{T} p(z_{l,t} \mid z_{l,t-1}) \times \prod_{t=1}^{T} p(x_t \mid z_{1:L,t})$$

Deep Model 3: Deep HMM

Parameterize transition and emission distributions with neural networks (c.f., Tran et al. [2016])

- Model transition distribution as

$$p(z_t \mid z_{t-1}) = \text{softmax}(\text{MLP}(z_{t-1};\, \mu))$$

- Model emission distribution as

$$p(x_t \mid z_t) = \text{softmax}(\text{MLP}(z_t;\, \pi))$$

Note: $K \times K$ transition parameters for standard HMM vs. $O(K \times d + d^2)$ for deep version.

Deep Model 3: Deep HMM

Parameterize transition and emission distributions with neural networks (c.f., Tran et al. [2016])

- Model transition distribution as

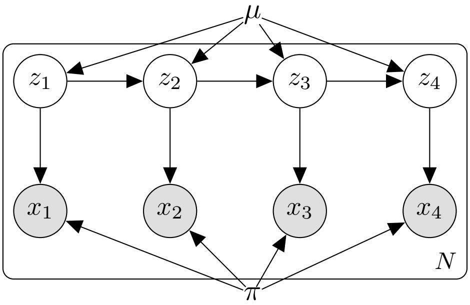
$$p(z_t \mid z_{t-1}) = \mathrm{softmax}(\mathrm{MLP}(z_{t-1};\, \mu))$$

- Model emission distribution as

$$p(x_t \mid z_t) = \mathrm{softmax}(\mathrm{MLP}(z_t;\, \pi))$$

Note: $K \times K$ transition parameters for standard HMM vs. $O(K \times d + d^2)$ for deep version.

Graphical Model View



$$p(x, z;\ \theta) = \prod_{t=1}^{T} p(z_t \mid z_{t-1};\ \mu_{z_{t-1}}) \times \prod_{t=1}^{T} p(x_t \mid z_t;\ \pi_{z_t})$$

$$= \prod_{t=1}^{T} \mu_{z_{t-1}, z_t} \times \prod_{t=1}^{T} \pi_{z_t, x_t}$$

Posterior Inference

For structured models, Bayes' rule may tractable,

$$p(z \mid x;\, \theta) = \frac{p(z;\, \mu) \times p(x \mid z;\, \pi)}{\sum_{z'} p(z';\, \mu) \times p(x \mid z';\, \pi)}$$

- Unlike previous models, z contains interdependent "parts."
- For *both* shallow and deep Model 3 variants, it's possible to calculate $p(x;\, \theta)$ exactly, with a dynamic program.
- For some structured models, like Factorial HMM, the dynamic program may still be intractable.

Posterior Inference

For structured models, Bayes' rule may tractable,

$$p(z \mid x;\, \theta) = \frac{p(z;\, \mu) \times p(x \mid z;\, \pi)}{\sum_{z'} p(z';\, \mu) \times p(x \mid z';\, \pi)}$$

- Unlike previous models, z contains interdependent "parts."
- For *both* shallow and deep Model 3 variants, it's possible to calculate $p(x;\, \theta)$ exactly, with a dynamic program.
- For some structured models, like Factorial HMM, the dynamic program may still be intractable.

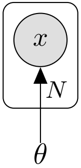
Learning with Maximum Likelihood

Objective: Find model parameters θ that maximize the likelihood of the data,

$$\theta^* = \arg\max_{\theta} \sum_{n=1}^{N} \log p(x^{(n)};\, \theta)$$

Learning Deep Models

$$L(\theta) = \sum_{n=1}^{N} \log p(x^{(n)};\, \theta)$$



- Dominant framework is gradient-based optimization:

$$\theta^{(i)} = \theta^{(i-1)} + \eta \nabla_\theta L(\theta)$$

- $\nabla_\theta L(\theta)$ calculated with backpropagation.
- Tactics: mini-batch based training, adaptive learning rates [Duchi et al. 2011; Kingma and Ba 2015].

Learning Deep Latent-Variable Models: Marginalization

Likelihood requires summing out the latent variables,

$$p(x;\theta) = \sum_{z \in \mathcal{Z}} p(x, z; \theta) \quad (= \int p(x, z; \theta) dz \text{ if continuous } z)$$

In general, **hard to optimize** log-likelihood for the training set,

$$L(\theta) = \sum_{n=1}^{N} \log \sum_{z \in \mathcal{Z}} p(x^{(n)}, z; \theta)$$

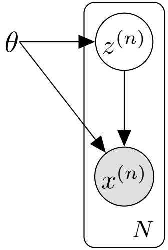
Learning Deep Latent-Variable Models: Marginalization

Likelihood requires summing out the latent variables,

$$p(x;\theta) = \sum_{z \in \mathcal{Z}} p(x, z; \theta) \quad \left(= \int p(x, z; \theta) dz \text{ if continuous } z\right)$$

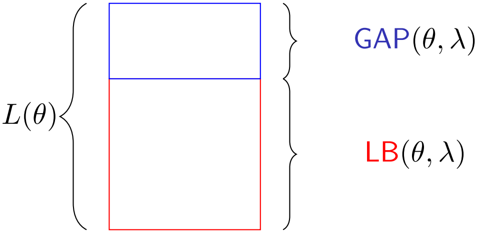
In general, **hard to optimize** log-likelihood for the training set,

$$L(\theta) = \sum_{n=1}^{N} \log \sum_{z \in \mathcal{Z}} p(x^{(n)}, z; \theta)$$



Variational Inference

High-level: decompose objective into lower-bound and gap.



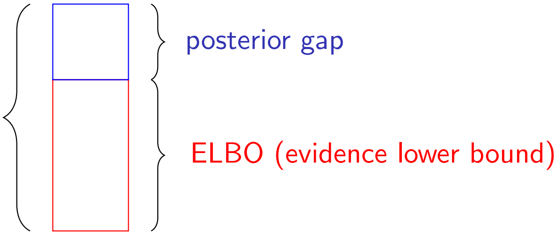
$$L(\theta) = \text{LB}(\theta, \lambda) + \text{GAP}(\theta, \lambda) \text{ for some } \lambda$$

Provides framework for deriving a rich set of optimization algorithms.

Marginal Likelihood: Variational Decomposition

For any[1] distribution $q(z \mid x; \lambda)$ over z,

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



Since KL is always non-negative, $L(\theta) \geq \mathrm{ELBO}(\theta, \lambda)$.

[1] Technical condition: $\text{supp}(q(z)) \subset \text{supp}(p(z \mid x;\, \theta))$

Evidence Lower Bound: Proof

$$\log p(x;\,\theta) = \mathbb{E}_q \log p(x) \quad \textit{(Expectation over } z\textit{)}$$

$$= \mathbb{E}_q \log \frac{p(x,z)}{p(z \mid x)} \quad \textit{(Mult/div by } p(z|x)\textit{, combine numerator)}$$

$$= \mathbb{E}_q \log \left(\frac{p(x,z)}{q(z \mid x)} \frac{q(z \mid x)}{p(z \mid x)} \right) \quad \textit{(Mult/div by } q(z|x)\textit{)}$$

$$= \mathbb{E}_q \log \frac{p(x,z)}{q(z \mid x)} + \mathbb{E}_q \log \frac{q(z \mid x)}{p(z \mid x)} \quad \textit{(Split Log)}$$

$$= \mathbb{E}_q \log \frac{p(x,z;\,\theta)}{q(z \mid x;\lambda)} + \mathrm{KL}[q(z \mid x;\lambda) \,\|\, p(z \mid x;\,\theta)]$$

Evidence Lower Bound: Proof

$$\begin{aligned}
\log p(x;\, \theta) &= \mathbb{E}_q \log p(x) \quad \textit{(Expectation over } z\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x, z)}{p(z \mid x)} \quad \textit{(Mult/div by } p(z|x)\textit{, combine numerator)} \\
&= \mathbb{E}_q \log \left(\frac{p(x, z)}{q(z \mid x)} \frac{q(z \mid x)}{p(z \mid x)} \right) \quad \textit{(Mult/div by } q(z|x)\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x, z)}{q(z \mid x)} + \mathbb{E}_q \log \frac{q(z \mid x)}{p(z \mid x)} \quad \textit{(Split Log)} \\
&= \mathbb{E}_q \log \frac{p(x, z;\, \theta)}{q(z \mid x;\, \lambda)} + \mathrm{KL}[q(z \mid x;\, \lambda) \,\|\, p(z \mid x;\, \theta)]
\end{aligned}$$

Evidence Lower Bound: Proof

$$\log p(x;\,\theta) = \mathbb{E}_q \log p(x) \quad \text{(Expectation over } z\text{)}$$

$$= \mathbb{E}_q \log \frac{p(x, z)}{p(z \,|\, x)} \quad \text{(Mult/div by } p(z|x)\text{, combine numerator)}$$

$$= \mathbb{E}_q \log \left(\frac{p(x, z)}{q(z \,|\, x)} \frac{q(z \,|\, x)}{p(z \,|\, x)} \right) \quad \text{(Mult/div by } q(z|x)\text{)}$$

$$= \mathbb{E}_q \log \frac{p(x, z)}{q(z \,|\, x)} + \mathbb{E}_q \log \frac{q(z \,|\, x)}{p(z \,|\, x)} \quad \text{(Split Log)}$$

$$= \mathbb{E}_q \log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} + \mathrm{KL}[q(z \,|\, x;\, \lambda) \,\|\, p(z \,|\, x;\, \theta)]$$

Evidence Lower Bound: Proof

$$
\begin{aligned}
\log p(x;\,\theta) &= \mathbb{E}_q \log p(x) \quad \textit{(Expectation over } z\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x,z)}{p(z \mid x)} \quad \textit{(Mult/div by } p(z|x)\textit{, combine numerator)} \\
&= \mathbb{E}_q \log \left(\frac{p(x,z)}{q(z \mid x)} \frac{q(z \mid x)}{p(z \mid x)} \right) \quad \textit{(Mult/div by } q(z|x)\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x,z)}{q(z \mid x)} + \mathbb{E}_q \log \frac{q(z \mid x)}{p(z \mid x)} \quad \textit{(Split Log)} \\
&= \mathbb{E}_q \log \frac{p(x,z;\,\theta)}{q(z \mid x;\lambda)} + \mathrm{KL}[q(z \mid x;\lambda) \,\|\, p(z \mid x;\,\theta)]
\end{aligned}
$$

Evidence Lower Bound: Proof

$$
\begin{aligned}
\log p(x;\,\theta) &= \mathbb{E}_q \log p(x) \quad \textit{(Expectation over } z\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x,z)}{p(z\,|\,x)} \quad \textit{(Mult/div by } p(z|x)\textit{, combine numerator)} \\
&= \mathbb{E}_q \log \left(\frac{p(x,z)}{q(z\,|\,x)} \frac{q(z\,|\,x)}{p(z\,|\,x)} \right) \quad \textit{(Mult/div by } q(z|x)\textit{)} \\
&= \mathbb{E}_q \log \frac{p(x,z)}{q(z\,|\,x)} + \mathbb{E}_q \log \frac{q(z\,|\,x)}{p(z\,|\,x)} \quad \textit{(Split Log)} \\
&= \mathbb{E}_q \log \frac{p(x,z;\,\theta)}{q(z\,|\,x;\lambda)} + \mathrm{KL}[q(z\,|\,x;\lambda) \,\|\, p(z\,|\,x;\,\theta)]
\end{aligned}
$$

Evidence Lower Bound over Observations

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

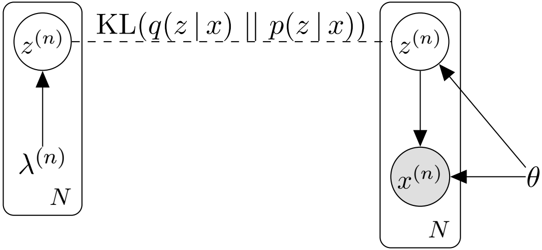
- ELBO is a function of the generative model parameters, θ, and the variational parameters, λ.

$$\begin{aligned}\sum_{n=1}^{N} \log p(x^{(n)};\, \theta) &\geq \sum_{n=1}^{N} \text{ELBO}(\theta, \lambda;\, x^{(n)}) \\ &= \sum_{n=1}^{N} \mathbb{E}_{q(z \,|\, x^{(n)};\, \lambda)}\left[\log \frac{p(x^{(n)}, z;\, \theta)}{q(z \,|\, x^{(n)};\, \lambda)}\right] \\ &= \text{ELBO}(\theta, \lambda;\, x^{(1:N)}) = \text{ELBO}(\theta, \lambda)\end{aligned}$$

Setup: Selecting Variational Family

- Just as with p and θ, we can select any form of q and λ that satisfies ELBO conditions.
- Different choices of q will lead to different algorithms.
- We will explore several forms of q:
  - Posterior
  - Point Estimate / MAP
  - Amortized
  - Mean Field (later)

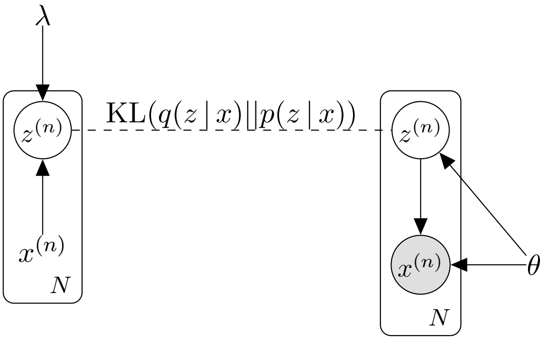
Example Family : Full Posterior Form



$\lambda = [\lambda^{(1)}, \dots, \lambda^{(N)}]$ is a concatenation of local variational parameters $\lambda^{(n)}$, e.g.

$$q(z^{(n)} \mid x^{(n)};\, \lambda) = q(z^{(n)} \mid x^{(n)};\, \lambda^{(n)}) = \mathcal{N}(\lambda^{(n)}, 1)$$

Example Family: Amortized Parameterization [Kingma and Welling 2014]



λ parameterizes a global network (encoder/inference network) that is run over $x^{(n)}$ to produce the local variational distribution, e.g.

$$q(z^{(n)} \mid x^{(n)}; \lambda) = \mathcal{N}(\mu^{(n)}, 1), \qquad \mu^{(n)} = \mathrm{enc}(x^{(n)}; \lambda)$$

Maximizing the Evidence Lower Bound

Central quantity of interest: almost all methods are maximizing the ELBO

$$\arg\max_{\theta,\lambda} \text{ELBO}(\theta, \lambda)$$

Aggregate ELBO objective,

$$\arg\max_{\theta,\lambda} \text{ELBO}(\theta, \lambda) = \arg\max_{\theta,\lambda} \sum_{n=1}^{N} \text{ELBO}(\theta, \lambda; x^{(n)})$$

$$= \arg\max_{\theta,\lambda} \sum_{n=1}^{N} \mathbb{E}_q \left[\log \frac{p(x^{(n)}, z^{(n)}; \theta)}{q(z^{(n)} \mid x^{(n)}; \lambda)} \right]$$

Maximizing the Evidence Lower Bound

Central quantity of interest: almost all methods are maximizing the ELBO

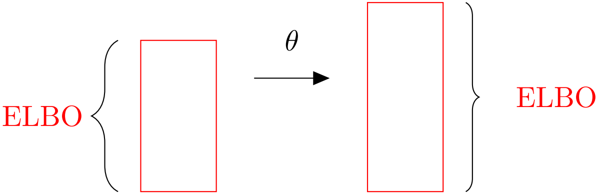
$$\arg\max_{\theta,\lambda} \mathrm{ELBO}(\theta, \lambda)$$

Aggregate ELBO objective,

$$\begin{aligned}
\arg\max_{\theta,\lambda} \mathrm{ELBO}(\theta, \lambda) &= \arg\max_{\theta,\lambda} \sum_{n=1}^{N} \mathrm{ELBO}(\theta, \lambda;\, x^{(n)}) \\
&= \arg\max_{\theta,\lambda} \sum_{n=1}^{N} \mathbb{E}_q \Big[\log \frac{p(x^{(n)}, z^{(n)};\, \theta)}{q(z^{(n)} \mid x^{(n)};\, \lambda)} \Big]
\end{aligned}$$

Maximizing ELBO: Model Parameters

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



Intuition: Maximum likelihood problem under variables drawn from $q(z \,|\, x; \lambda)$.

Model Estimation: Gradient Ascent on Model Parameters

Easy: Gradient respect to θ

$$\begin{aligned} \nabla_\theta \operatorname{ELBO}(\theta, \lambda; x) &= \nabla_\theta \mathbb{E}_q \Big[\log p(x, z; \theta) \Big] \\ &= \mathbb{E}_q \Big[\nabla_\theta \log p(x, z; \theta) \Big] \end{aligned}$$

- Since q not dependent on θ, ∇ moves inside expectation.
- Estimate with samples from q. Term $\log p(x, z; \theta)$ is easy to evaluate. (In practice single sample is often sufficient).
- In special cases, can exactly evaluate expectation.

Model Estimation: Gradient Ascent on Model Parameters

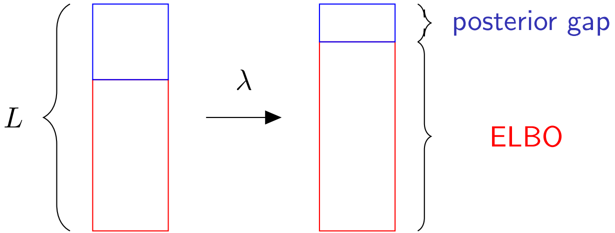
Easy: Gradient respect to θ

$$\begin{aligned}\nabla_\theta \, \text{ELBO}(\theta, \lambda; x) &= \nabla_\theta \mathbb{E}_q \Big[\log p(x, z; \theta) \Big] \\ &= \mathbb{E}_q \Big[\nabla_\theta \log p(x, z; \theta) \Big]\end{aligned}$$

- Since q not dependent on θ, ∇ moves inside expectation.
- Estimate with samples from q. Term $\log p(x, z; \theta)$ is easy to evaluate. (In practice single sample is often sufficient).
- In special cases, can exactly evaluate expectation.

Maximizing ELBO: Variational Distribution

$$\arg\max_{\lambda} \text{ELBO}(\theta, \lambda)$$
$$= \arg\max_{\lambda} \log p(x;\, \theta) - \text{KL}[q(z \,|\, x; \lambda) \,\|\, p(z \,|\, x;\, \theta)]$$
$$= \arg\min_{\lambda} \text{KL}[q(z \,|\, x; \lambda) \,\|\, p(z \,|\, x;\, \theta)]$$



Intuition: q should approximate the posterior $p(z|x)$. However, may be difficult if q or p is a deep model.

Model Inference: Gradient Ascent on λ?

Hard: Gradient respect to λ

$$\nabla_\lambda \operatorname{ELBO}(\theta, \lambda; x) = \nabla_\lambda \mathbb{E}_q\Big[\log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)}\Big]$$
$$\neq \mathbb{E}_q\Big[\nabla_\lambda \log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)}\Big]$$

- Cannot naively move ∇ inside the expectation, since q depends on λ.
- This section: Inference in practice:
  1. Exact gradient
  2. Sampling: score function, reparameterization
  3. Conjugacy: closed-form, coordinate ascent

Model Inference: Gradient Ascent on λ?

Hard: Gradient respect to λ

$$\nabla_\lambda \mathrm{ELBO}(\theta, \lambda; x) = \nabla_\lambda \mathbb{E}_q\left[\log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)}\right]$$
$$\neq \mathbb{E}_q\left[\nabla_\lambda \log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)}\right]$$

- Cannot naively move ∇ inside the expectation, since q depends on λ.
- This section: Inference in practice:
  1. Exact gradient
  2. Sampling: score function, reparameterization
  3. Conjugacy: closed-form, coordinate ascent

Strategy 1: Exact Gradient

$$\nabla_\lambda \operatorname{ELBO}(\theta, \lambda; x) = \nabla_\lambda \mathbb{E}_{q(z \,|\, x; \lambda)}\left[\log \frac{p(x, z; \theta)}{q(z \,|\, x; \lambda)}\right]$$
$$= \nabla_\lambda \left(\sum_{z \in \mathcal{Z}} q(z \,|\, x; \lambda) \log \frac{p(x, z; \theta)}{q(z \,|\, x; \lambda)} \right)$$

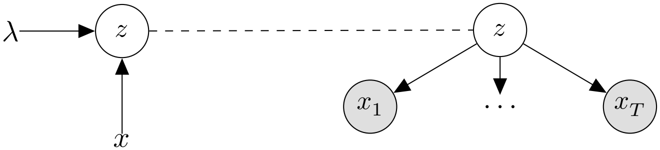
- Naive enumeration: Linear in $|\mathcal{Z}|$.
- Depending on structure of q and p, potentially faster with dynamic programming.
- Applicable mainly to Model 1 and 3 (Discrete and Structured), or Model 2 with point estimate.

Strategy 1: Exact Gradient

$$\begin{aligned}\nabla_\lambda \operatorname{ELBO}(\theta, \lambda; x) &= \nabla_\lambda \mathbb{E}_{q(z \mid x; \lambda)}\Big[\log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)}\Big] \\ &= \nabla_\lambda \Big(\sum_{z \in \mathcal{Z}} q(z \mid x; \lambda) \log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)} \Big)\end{aligned}$$

- Naive enumeration: Linear in $|\mathcal{Z}|$.
- Depending on structure of q and p, potentially faster with dynamic programming.
- Applicable mainly to Model 1 and 3 (Discrete and Structured), or Model 2 with point estimate.

Example: Model 1 - Naive Bayes



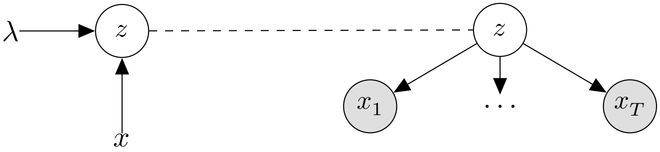
Let $q(z \,|\, x;\, \lambda) = \text{Cat}(\nu)$ where $\nu = \text{enc}(x; \lambda)$

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

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

$$= \nabla_\lambda \Big(\sum_{z \in \mathcal{Z}} \nu_z \log \frac{p(x, z;\, \theta)}{\nu_z} \Big)$$

Example: Model 1 - Naive Bayes



Let $q(z \mid x;\ \lambda) = \mathrm{Cat}(\nu)$ where $\nu = \mathrm{enc}(x; \lambda)$

$$\begin{aligned}
\nabla_\lambda \,\mathrm{ELBO}(\theta, \lambda;\ x) &= \nabla_\lambda \mathbb{E}_{q(z \mid x;\, \lambda)}\Big[\log \frac{p(x, z;\ \theta)}{q(z \mid x;\ \lambda)} \Big] \\
&= \nabla_\lambda \Big(\sum_{z \in \mathcal{Z}} q(z \mid x;\ \lambda) \log \frac{p(x, z;\ \theta)}{q(z \mid x;\ \lambda)} \Big) \\
&= \nabla_\lambda \Big(\sum_{z \in \mathcal{Z}} \nu_z \log \frac{p(x, z;\ \theta)}{\nu_z} \Big)
\end{aligned}$$

Strategy 2: Sampling

$$\nabla_\lambda \operatorname{ELBO}(\theta, \lambda; x) = \nabla_\lambda \mathbb{E}_q \left[\log \frac{\log p(x, z;\, \theta)}{\log q(z \,|\, x;\, \lambda)} \right]$$

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

- How can we approximate this gradient with sampling? Naive algorithm fails to provide non-zero gradient.

$$z^{(1)}, \dots, z^{(J)} \sim q(z \,|\, x;\, \lambda)$$

$$\nabla_\lambda \frac{1}{J} \sum_{j=1}^{J} \left[\log p(x, z^{(j)};\, \theta) \right] = 0$$

- Manipulate expression so we can move ∇_λ inside $\mathbb{E}_q$ before sampling.

Strategy 2: Sampling

$$\nabla_\lambda \operatorname{ELBO}(\theta, \lambda; x) = \nabla_\lambda \mathbb{E}_q \Big[\log \frac{\log p(x, z;\, \theta)}{\log q(z \,|\, x;\, \lambda)} \Big]$$

$$= \nabla_\lambda \mathbb{E}_q \Big[\log p(x, z;\, \theta) \Big] - \nabla_\lambda \mathbb{E}_q \Big[\log q(z \,|\, x;\, \theta) \Big]$$

- How can we approximate this gradient with sampling? Naive algorithm fails to provide non-zero gradient.

$$z^{(1)}, \dots, z^{(J)} \sim q(z \,|\, x;\, \lambda)$$

$$\nabla_\lambda \frac{1}{J} \sum_{j=1}^{J} \Big[\log p(x, z^{(j)};\, \theta) \Big] = 0$$

- Manipulate expression so we can move ∇_λ inside $\mathbb{E}_q$ before sampling.

Strategy 2a: Sampling — Score Function Gradient Estimator

First term. Use basic identity:

$$\nabla \log q = \frac{\nabla q}{q} \Rightarrow \nabla q = q \nabla \log q$$

Policy-gradient style training [Williams 1992]

$$\nabla_\lambda \mathbb{E}_q \Big[\log p(x, z; \theta) \Big] = \sum_z \nabla_\lambda q(z \,|\, x; \lambda) \log p(x, z; \theta)$$

Strategy 2a: Sampling — Score Function Gradient Estimator

First term. Use basic identity:

$$\nabla \log q = \frac{\nabla q}{q} \Rightarrow \nabla q = q \nabla \log q$$

Policy-gradient style training [Williams 1992]

$$\nabla_\lambda \mathbb{E}_q \Big[\log p(x, z; \theta) \Big] = \sum_z \underbrace{\nabla_\lambda q(z \mid x; \lambda)}_{q \nabla \log q} \log p(x, z; \theta)$$

Strategy 2a: Sampling — Score Function Gradient Estimator

First term. Use basic identity:

$$\nabla \log q = \frac{\nabla q}{q} \Rightarrow \nabla q = q \nabla \log q$$

Policy-gradient style training [Williams 1992]

$$\nabla_\lambda \mathbb{E}_q \Big[\log p(x, z; \theta) \Big] = \sum_z \nabla_\lambda q(z \mid x; \lambda) \log p(x, z; \theta)$$
$$= \sum_z q(z \mid x; \lambda) \nabla_\lambda \log q(z \mid x; \lambda) \log p(x, z; \theta)$$

Strategy 2a: Sampling — Score Function Gradient Estimator

First term. Use basic identity:

$$\nabla \log q = \frac{\nabla q}{q} \Rightarrow \nabla q = q \nabla \log q$$

Policy-gradient style training [Williams 1992]

$$\begin{aligned}\nabla_\lambda \mathbb{E}_q\Big[\log p(x, z;\, \theta)\Big] &= \sum_z \nabla_\lambda q(z \,|\, x;\, \lambda) \log p(x, z;\, \theta) \\ &= \sum_z q(z \,|\, x;\, \lambda) \nabla_\lambda \log q(z \,|\, x;\, \lambda) \log p(x, z;\, \theta) \\ &= \mathbb{E}_q\Big[\log p(x, z;\, \theta) \nabla_\lambda \log q(z \,|\, x;\, \lambda)\Big]\end{aligned}$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Second term. Need additional identity:

$$\sum \nabla q = \nabla \sum q = \nabla 1 = 0$$

$$\nabla_\lambda \mathbb{E}_q \Big[\log q(z \,|\, x;\, \lambda) \Big] = \sum_z \nabla_\lambda \Big(q(z \,|\, x;\, \lambda) \log q(z \,|\, x;\, \lambda) \Big)$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Second term. Need additional identity:

$$\sum \nabla q = \nabla \sum q = \nabla 1 = 0$$

$$\nabla_\lambda \mathbb{E}_q \Big[\log q(z \,|\, x;\, \lambda) \Big] = \sum_z \nabla_\lambda \Big(q(z \,|\, x;\, \lambda) \log q(z \,|\, x;\, \lambda) \Big)$$

$$= \sum_z \Big(\underbrace{\nabla_\lambda q(z \,|\, x;\, \lambda)}_{q \nabla \log q} \Big) \log q(z \,|\, x;\, \lambda) + q(z \,|\, x;\, \lambda) \Big(\underbrace{\nabla_\lambda \log q(z \,|\, x;\, \lambda)}_{\frac{\nabla q}{q}} \Big)$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Second term. Need additional identity:

$$\sum \nabla q = \nabla \sum q = \nabla 1 = 0$$

$$\nabla_\lambda \mathbb{E}_q \Big[\log q(z \,|\, x;\, \lambda) \Big] = \sum_z \nabla_\lambda \Big(q(z \,|\, x;\, \lambda) \log q(z \,|\, x;\, \lambda) \Big)$$

$$= \sum_z \log q(z \,|\, x;\, \lambda) q(z \,|\, x;\, \lambda) \nabla_\lambda \log q(z \,|\, x;\, \lambda) + \sum_z \nabla_\lambda q(z \,|\, x;\, \lambda)$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Second term. Need additional identity:

$$\sum \nabla q = \nabla \sum q = \nabla 1 = 0$$

$$\nabla_\lambda \mathbb{E}_q \left[\log q(z \,|\, x;\, \lambda) \right] = \sum_z \nabla_\lambda \Big(q(z \,|\, x;\, \lambda) \log q(z \,|\, x;\, \lambda) \Big)$$

$$= \sum_z \log q(z \,|\, x;\, \lambda) q(z \,|\, x;\, \lambda) \nabla_\lambda \log q(z \,|\, x;\, \lambda) + \underbrace{\sum_z \nabla_\lambda q(z \,|\, x;\, \lambda)}_{= \nabla \sum q = \nabla 1 = 0}$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Second term. Need additional identity:

$$\sum \nabla q = \nabla \sum q = \nabla 1 = 0$$

$$\nabla_\lambda \mathbb{E}_q\left[\log q(z \,|\, x;\, \lambda)\right] = \sum_z \nabla_\lambda \Big(q(z \,|\, x;\, \lambda) \log q(z \,|\, x;\, \lambda)\Big)$$

$$= \sum_z \log q(z \,|\, x;\, \lambda) q(z \,|\, x;\, \lambda) \nabla_\lambda \log q(z \,|\, x;\, \lambda) + \sum_z \nabla_\lambda q(z \,|\, x;\, \lambda)$$

$$= \mathbb{E}_q[\log q(z \,|\, x;\, \lambda) \nabla_\lambda q(z \,|\, x;\, \lambda)]$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Putting these together,

$$
\begin{aligned}
\nabla_\lambda \operatorname{ELBO}(\theta, \lambda; x) &= \nabla_\lambda \mathbb{E}_q \Big[\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} \Big] \\
&= \mathbb{E}_q \Big[\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} \nabla_\lambda \log q(z \,|\, x;\, \lambda) \Big] \\
&= \mathbb{E}_q \Big[R_{\theta,\lambda}(z) \nabla_\lambda \log q(z \,|\, x;\, \lambda) \Big]
\end{aligned}
$$

Strategy 2a: Sampling — Score Function Gradient Estimator

Estimate with samples,

$$z^{(1)}, \dots, z^{(J)} \sim q(z \,|\, x;\, \lambda)$$

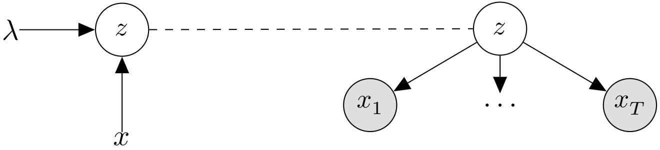
$$\begin{aligned}\mathbb{E}_q \Big[R_{\theta,\lambda}(z) \nabla_\lambda \log q(z \,|\, x;\, \lambda) \Big] \\ \approx \frac{1}{J} \sum_{j=1}^{J} R_{\theta,\lambda}(z^{(j)}) \nabla_\lambda \log q(z^{(j)} \,|\, x;\, \lambda)\end{aligned}$$

Intuition: if a sample $z^{(j)}$ is has high reward $R_{\theta,\lambda}(z^{(j)})$, increase the probability of $z^{(j)}$ by moving along the gradient $\nabla_\lambda \log q(z^{(j)} \,|\, x;\, \lambda)$.

Strategy 2a: Sampling — Score Function Gradient Estimator

- Essentially reinforcement learning with reward $R_{\theta,\lambda}(z)$
- Score function gradient is generally applicable regardless of what distribution q takes (only need to evaluate $\nabla_{\lambda} \log q$).
- This generality comes at a cost, since the reward is "black-box": unbiased estimator, but high variance.
- In practice, need variance-reducing **control variate** B. (More on this later).

Example: Model 1 - Naive Bayes



Let $q(z \mid x;\, \lambda) = \text{Cat}(\nu)$ where $\nu = \text{enc}(x; \lambda)$

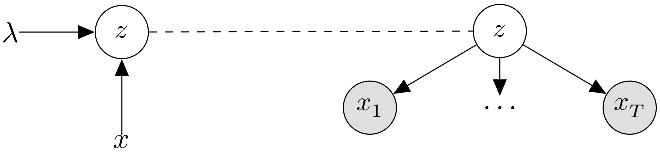
Sample $z^{(1)}, \dots, z^{(J)} \sim q(z \mid x; \lambda)$

$$\nabla_\lambda \, \text{ELBO}(\theta, \lambda;\, x) = \mathbb{E}_q\left[\log \frac{p(x, z;\, \theta)}{q(z \mid x;\, \lambda)} \nabla_\lambda \log q(z \mid x;\, \lambda)\right]$$

$$\approx \frac{1}{J} \sum_{j=1}^{J} \nu_{z^{(j)}} \log \frac{p(x, z^{(j)};\, \theta)}{\nu_{z^{(j)}}} \nabla_\lambda \log \nu_{z^{(j)}}$$

Computational complexity: $O(J)$ vs $O(|\mathcal{Z}|)$

Example: Model 1 - Naive Bayes



Let $q(z \,|\, x;\, \lambda) = \text{Cat}(\nu)$ where $\nu = \text{enc}(x; \lambda)$

Sample $z^{(1)}, \dots, z^{(J)} \sim q(z \,|\, x; \lambda)$

$$\nabla_\lambda \text{ELBO}(\theta, \lambda;\, x) = \mathbb{E}_q \left[\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} \nabla_\lambda \log q(z \,|\, x;\, \lambda) \right]$$

$$\approx \frac{1}{J} \sum_{j=1}^{J} \nu_{z^{(j)}} \log \frac{p(x, z^{(j)};\, \theta)}{\nu_{z^{(j)}}} \nabla_\lambda \log \nu_{z^{(j)}}$$

Computational complexity: $O(J)$ vs $O(|\mathcal{Z}|)$

Strategy 2b: Sampling — Reparameterization

Suppose we can sample from q by applying a deterministic, differentiable transformation g to a base noise density,

$$\epsilon \sim \mathcal{U} \qquad z = g(\epsilon, \lambda)$$

Gradient calculation (first term):

$$\begin{aligned}\nabla_\lambda \mathbb{E}_{z \sim q(z \mid x; \lambda)}\Big[\log p(x, z; \theta)\Big] &= \nabla_\lambda \mathbb{E}_{\epsilon \sim \mathcal{U}}\Big[\log p(x, g(\epsilon, \lambda); \theta)\Big] \\ &= \mathbb{E}_{\epsilon \sim \mathcal{U}}\Big[\nabla_\lambda \log p(x, g(\epsilon, \lambda); \theta)\Big] \\ &\approx \frac{1}{J}\sum_{j=1}^{J} \nabla_\lambda \log p(x, g(\epsilon^{(j)}, \lambda); \theta)\end{aligned}$$

where

$$\epsilon^{(1)}, \dots, \epsilon^{(J)} \sim \mathcal{U}$$

Strategy 2b: Sampling — Reparameterization

Suppose we can sample from q by applying a deterministic, differentiable transformation g to a base noise density,

$$\epsilon \sim \mathcal{U} \qquad z = g(\epsilon, \lambda)$$

Gradient calculation (first term):

$$\begin{aligned}
\nabla_\lambda \mathbb{E}_{z \sim q(z \,|\, x;\, \lambda)}\Big[\log p(x, z;\, \theta)\Big] &= \nabla_\lambda \mathbb{E}_{\epsilon \sim \mathcal{U}}\Big[\log p(x, g(\epsilon, \lambda);\, \theta)\Big] \\
&= \mathbb{E}_{\epsilon \sim \mathcal{U}}\Big[\nabla_\lambda \log p(x, g(\epsilon, \lambda);\, \theta)\Big] \\
&\approx \frac{1}{J}\sum_{j=1}^{J} \nabla_\lambda \log p(x, g(\epsilon^{(j)}, \lambda);\, \theta)
\end{aligned}$$

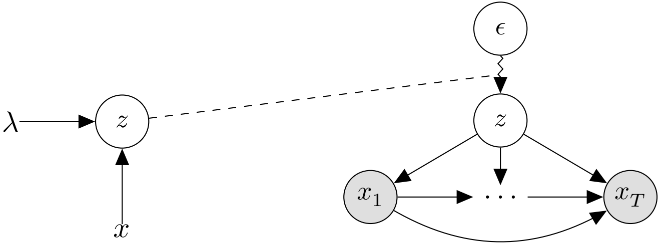
where

$$\epsilon^{(1)}, \dots, \epsilon^{(J)} \sim \mathcal{U}$$

Strategy 2b: Sampling — Reparameterization

- Unbiased, like the score function gradient estimator, but empirically lower variance.
- In practice, single sample is often sufficient.
- Cannot be used out-of-the-box for discrete z.

Strategy 2: Continuous Latent Variable RNN



Choose variational family to be an amortized diagonal Gaussian

$$q(z \,|\, x;\, \lambda) = \mathcal{N}(\mu, \sigma^2)$$

$$\mu, \sigma^2 = \mathrm{enc}(x;\, \lambda)$$

Then we can sample from $q(z \,|\, x;\, \lambda)$ by

$$\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \qquad z = \mu + \sigma\epsilon$$

Strategy 2b: Sampling — Reparameterization

(Recall $R_{\theta,\lambda}(z) = \log \frac{p(x,z;\theta)}{q(z \mid x;\lambda)}$)

- Score function:

$$\nabla_\lambda \, \text{ELBO}(\theta, \lambda; x) = \mathbb{E}_{z\sim q}[R_{\theta,\lambda}(z)\nabla_\lambda \log q(z \mid x; \lambda)]$$

- Reparameterization:

$$\nabla_\lambda \, \text{ELBO}(\theta, \lambda; x) = \mathbb{E}_{\epsilon\sim\mathcal{N}(\mathbf{0},\mathbf{I})}[\nabla_\lambda R_{\theta,\lambda}(g(\epsilon, \lambda; x))]$$

where $g(\epsilon, \lambda; x) = \mu + \sigma\epsilon$.

Informally, reparameterization gradients differentiate through $R_{\theta,\lambda}(\cdot)$ and thus has "more knowledge" about the structure of the objective function.

Strategy 3: Conjugacy

For certain choices for p and q, we can compute parts of

$$\arg\max_{\lambda} \text{ELBO}(\theta, \lambda; x)$$

exactly in closed-form.

Recall that

$$\arg\max_{\lambda} \text{ELBO}(\theta, \lambda; x) = \arg\min_{\lambda} \text{KL}[q(z \mid x;\, \lambda) \| p(z \mid x;\, \theta)]$$

Strategy 3: Conjugacy

For certain choices for p and q, we can compute parts of

$$\arg\max_{\lambda} \text{ELBO}(\theta, \lambda; x)$$

exactly in closed-form.

Recall that

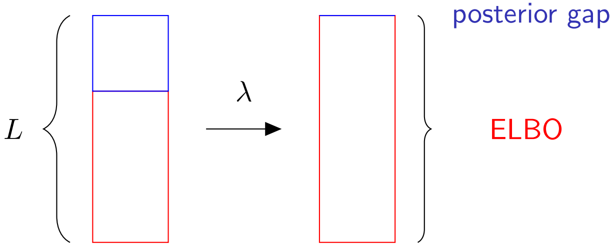
$$\arg\max_{\lambda} \text{ELBO}(\theta, \lambda; x) = \arg\min_{\lambda} \text{KL}[q(z \mid x;\ \lambda) \| p(z \mid x;\ \theta)]$$

Strategy 3a: Conjugacy — Tractable Posterior Inference

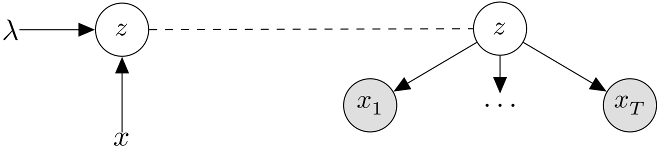
Suppose we can tractably calculate $p(z \mid x;\, \theta)$. Then $\mathrm{KL}[q(z \mid x;\, \lambda) \| p(z \mid x;\, \theta)]$ is minimized when,

$$q(z \mid x;\, \lambda) = p(z \mid x;\, \theta)$$

- The E-step in Expectation Maximization algorithm [Dempster et al. 1977]



Example: Model 1 - Naive Bayes

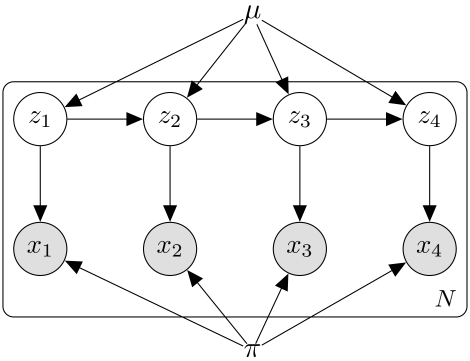


$$p(z \mid x;\, \theta) = \frac{p(x, z;\, \theta)}{\sum_{z'=1}^{K} p(x, z';\, \theta)}$$

So λ is given by the parameters of the categorical distribution, i.e.

$$\lambda = [p(z = 1 \mid x;\, \theta), \dots, p(z = K \mid x;\, \theta)]$$

Example: Model 3 — HMM



$$p(x, z; \theta) = p(z_0) \prod_{t=1}^{T} p(z_t \mid z_{t-1}; \mu) p(x_t \mid z_t; \pi)$$

Example: Model 3 — HMM

Run forward/backward dynamic programming to calculate posterior marginals,

$$p(z_t, z_{t+1} \mid x;\, \theta)$$

variational parameters $\lambda \in \mathbb{R}^{TK^2}$ store edge marginals. These are enough to calculate

$$q(z;\, \lambda) = p(z \mid x;\, \theta)$$

(i.e. the exact posterior) over any sequence z.

Example: Model 3 — HMM

Run forward/backward dynamic programming to calculate posterior marginals,

$$p(z_t, z_{t+1} \,|\, x;\, \theta)$$

variational parameters $\lambda \in \mathbb{R}^{TK^2}$ store edge marginals. These are enough to calculate

$$q(z;\, \lambda) = p(z \,|\, x;\, \theta)$$

(i.e. the exact posterior) over any sequence z.

Connection: Gradient Ascent on Log Marginal Likelihood

Why not perform gradient ascent directly on log marginal likelihood?

$$\log p(x;\, \theta) = \log \sum_z p(x, z;\, \theta)$$

Same as optimizing ELBO with posterior inference (i.e EM). Gradients of model parameters given by (where $q(z \mid x;\, \lambda) = p(z \mid x;\, \theta)$):

$$\nabla_\theta \log p(x;\, \theta) = \mathbb{E}_{q(z \mid x;\, \lambda)}[\nabla_\theta \log p(x, z;\, \theta)]$$

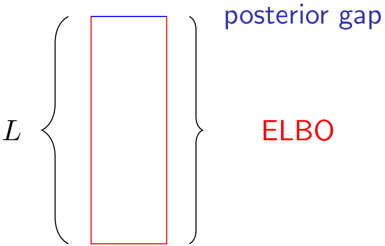
Connection: Gradient Ascent on Log Marginal Likelihood

Why not perform gradient ascent directly on log marginal likelihood?

$$\log p(x;\, \theta) = \log \sum_z p(x, z;\, \theta)$$

Same as optimizing ELBO with posterior inference (i.e EM). Gradients of model parameters given by (where $q(z \,|\, x;\, \lambda) = p(z \,|\, x;\, \theta)$):

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



Connection: Gradient Ascent on Log Marginal Likelihood

- Practically, this means we don't have to manually perform posterior inference in the E-step. Can just calculate $\log p(x; \theta)$ and call backpropagation.
- Example: in deep HMM, just implement forward algorithm to calculate $\log p(x; \theta)$ and backpropagate using autodiff. No need to implement backward algorithm. (Or vice versa).

(See Eisner [2016]: "Inside-Outside and Forward-Backward Algorithms Are Just Backprop")

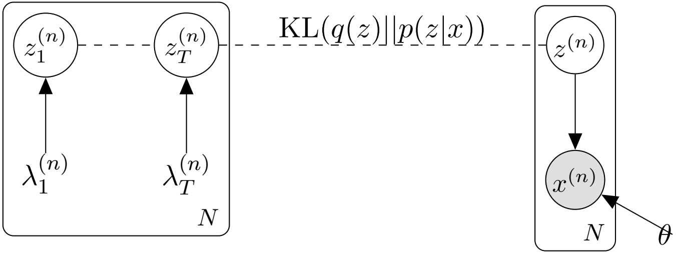
Strategy 3b: Conditional Conjugacy

- Let $p(z \,|\, x;\, \theta)$ be intractable, but suppose $p(x, z;\, \theta)$ is **conditionally conjugate**, meaning $p(z_t \,|\, x, z_{-t};\, \theta)$ is exponential family.
- Restrict the family of distributions q so that it factorizes over z_t, i.e.

$$q(z;\, \lambda) = \prod_{t=1}^{T} q(z_t;\, \lambda_t)$$

  (**mean field** family)
- Further choose $q(z_t;\, \lambda_t)$ so that it is in the same family as $p(z_t \,|\, x, z_{-t};\, \theta)$.

Strategy 3b: Conditional Conjugacy



$$q(z;\lambda) = \prod_{t=1}^{T} q(z_t;\lambda_t)$$

Mean Field Family

- Optimize ELBO via coordinate ascent, i.e. iterate for $\lambda_1, \dots, \lambda_T$

$$\arg\max_{\lambda_t} \ \mathrm{KL}\left[\prod_{t=1}^{T} q(z_t;\, \lambda_t) \| p(z \,|\, x;\, \theta)\right]$$

- Coordinate ascent updates will take the form

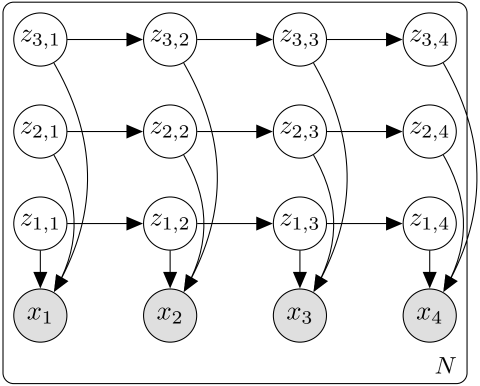
$$q(z_t;\, \lambda_t) \propto \exp\left(\mathbb{E}_{q(z_{-t};\, \lambda_{-t})}[\log p(x, z;\, \theta)]\right)$$

where

$$\mathbb{E}_{q(z_{-t};\, \lambda_{-t})}[\log p(x, z;\, \theta)] = \sum_{j \neq t} \prod_{j \neq t} q(z_j;\, \lambda_j) \log p(x, z;\, \theta)$$

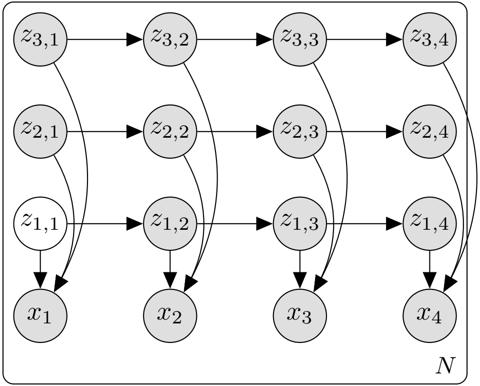
- Since $p(z_t \,|\, x, z_{-t})$ was assumed to be in the exponential family, above updates can be derived in closed form.

Example: Model 3 — Factorial HMM



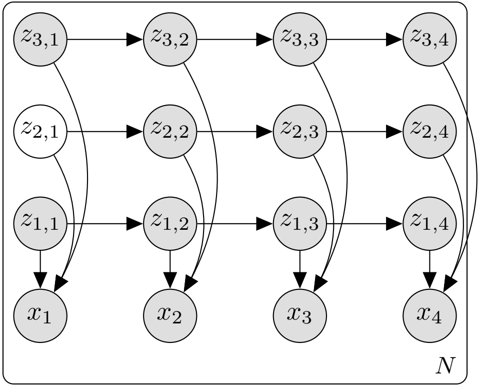
$$p(x, z; \theta) = \prod_{l=1}^{L} \prod_{t=1}^{T} p(z_{l,t} \mid z_{l,t-1}; \theta) p(x_t \mid z_{l,t}; \theta)$$

Example: Model 3 — Factorial HMM



$$q(z_{1,1};\, \lambda_{1,1}) \propto \exp\left(\mathbb{E}_{q(z_{-(1,1)};\, \lambda_{-(1,1)})}[\log p(x, z;\, \theta)]\right)$$

Example: Model 3 — Factorial HMM



$$q(z_{2,1};\, \lambda_{2,1}) \propto \exp\left(\mathbb{E}_{q(z_{-(2,1)};\, \lambda_{-(2,1)})}[\log p(x, z;\, \theta)]\right)$$

Example: Model 3 — Factorial HMM

Exact Inference:

- Naive: K states, L levels $\implies$ HMM with K^L states $\implies$ $O(TK^{2L})$
- Smarter: $O(TLK^{L+1})$

Mean Field:

- Gaussian emissions: $O(TLK^2)$ [Ghahramani and Jordan 1996].
- Categorical emission: need more variational approximations, but ultimately $O(LKVT)$ [Nepal and Yates 2013].

Example: Model 3 — Factorial HMM

Exact Inference:

- Naive: K states, L levels $\implies$ HMM with K^L states $\implies$ $O(TK^{2L})$
- Smarter: $O(TLK^{L+1})$

Mean Field:

- Gaussian emissions: $O(TLK^2)$ [Ghahramani and Jordan 1996].
- Categorical emission: need more variational approximations, but ultimately $O(LKVT)$ [Nepal and Yates 2013].

Advanced Topics

1. Gumbel-Softmax: Extend reparameterization to discrete variables.
2. Flows: Optimize a tighter bound by making the variational family q more flexible.
3. Importance Weighting: Optimize a tighter bound through importance sampling.

Challenges of Discrete Variables

Review: we can always use score function estimator

$$\nabla_\lambda \operatorname{ELBO}(x, \theta, \lambda) = \mathbb{E}_q\left[\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} \nabla_\lambda \log q(z \,|\, x;\, \lambda)\right]$$

$$= \mathbb{E}_q\left[\left(\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} - B\right) \nabla_\lambda \log q(z \,|\, x;\, \lambda)\right]$$

- $\mathbb{E}_q[B \nabla_\lambda \log q(z \,|\, x;\, \lambda)] = 0$ (since $\mathbb{E}[\nabla \log q] = \sum q \nabla \log q = \sum \nabla q = 0$)
- Control variate B (not dependent on z, but can depend on x).
- Estimate this quantity with another neural net [Mnih and Gregor 2014]

$$\left(B(x;\, \psi) - \log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)}\right)^2$$

Challenges of Discrete Variables

Review: we can always use score function estimator

$$\nabla_\lambda \operatorname{ELBO}(x, \theta, \lambda) = \mathbb{E}_q\Big[\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} \nabla_\lambda \log q(z \,|\, x;\, \lambda)\Big]$$

$$= \mathbb{E}_q\Big[\Big(\log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)} - B\Big) \nabla_\lambda \log q(z \,|\, x;\, \lambda)\Big]$$

- $\mathbb{E}_q[B \nabla_\lambda \log q(z \,|\, x;\, \lambda)] = 0$ (since $\mathbb{E}[\nabla \log q] = \sum q \nabla \log q = \sum \nabla q = 0$)
- Control variate B (not dependent on z, but can depend on x).
- Estimate this quantity with another neural net [Mnih and Gregor 2014]

$$\Big(B(x;\, \psi) - \log \frac{p(x, z;\, \theta)}{q(z \,|\, x;\, \lambda)}\Big)^2$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

The "Gumbel-Max" trick [Papandreou and Yuille 2011]

$$p(z_k = 1; \alpha) = \frac{\alpha_k}{\sum_{j=1}^{K} \alpha_j}$$

where $z = [0, 0, \dots, 1, \dots, 0]$ is a one-hot vector.

Can sample from $p(z; \alpha)$ by

1. Drawing independent Gumbel noise $\epsilon = \epsilon_1, \dots, \epsilon_K$

$$\epsilon_k = -\log(-\log u_k) \qquad u_k \sim \mathcal{U}(0, 1)$$

2. Adding ϵ_k to $\log \alpha_k$, finding argmax, i.e.

$$i = \arg\max_k [\log \alpha_k + \epsilon_k] \qquad z_i = 1$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

The "Gumbel-Max" trick [Papandreou and Yuille 2011]

$$p(z_k = 1; \alpha) = \frac{\alpha_k}{\sum_{j=1}^{K} \alpha_j}$$

where $z = [0, 0, \dots, 1, \dots, 0]$ is a one-hot vector.

Can sample from $p(z; \alpha)$ by

1. Drawing independent Gumbel noise $\epsilon = \epsilon_1, \dots, \epsilon_K$

$$\epsilon_k = -\log(-\log u_k) \qquad u_k \sim \mathcal{U}(0, 1)$$

2. Adding ϵ_k to $\log \alpha_k$, finding argmax, i.e.

$$i = \arg\max_k [\log \alpha_k + \epsilon_k] \qquad z_i = 1$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

The "Gumbel-Max" trick [Papandreou and Yuille 2011]

$$p(z_k = 1; \alpha) = \frac{\alpha_k}{\sum_{j=1}^{K} \alpha_j}$$

where $z = [0, 0, \dots, 1, \dots, 0]$ is a one-hot vector.

Can sample from $p(z; \alpha)$ by

1. Drawing independent Gumbel noise $\epsilon = \epsilon_1, \dots, \epsilon_K$

$$\epsilon_k = -\log(-\log u_k) \qquad u_k \sim \mathcal{U}(0, 1)$$

2. Adding ϵ_k to $\log \alpha_k$, finding argmax, i.e.

$$i = \arg\max_k [\log \alpha_k + \epsilon_k] \qquad z_i = 1$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

Reparameterization:

$$z = \arg\max_{s \in \Delta^{K-1}} (\log \alpha + \epsilon)^\top s = g(\epsilon, \alpha)$$

$z = g(\epsilon, \alpha)$ is a deterministic function applied to stochastic noise.

Let's try applying this:

$$q(z_k = 1 \mid x; \lambda) = \frac{\alpha_k}{\sum_{j=1}^{K} \alpha_j} \qquad \alpha = \text{enc}(x; \lambda)$$

(Recalling $R_{\theta,\lambda}(z) = \log \frac{p(x,z;\theta)}{q(z \mid x;\lambda)}$),

$$\nabla_\lambda \mathbb{E}_{q(z \mid x;\lambda)}[R_{\theta,\lambda}(z)] = \nabla_\lambda \mathbb{E}_{\epsilon\sim\text{Gumbel}}[R_{\theta,\lambda}(g(\epsilon,\alpha))]$$

$$= \mathbb{E}_{\epsilon\sim\text{Gumbel}}[\nabla_\lambda R_{\theta,\lambda}(g(\epsilon,\alpha))]$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

Reparameterization:

$$z = \underset{s \in \Delta^{K-1}}{\arg\max}\ (\log \alpha + \epsilon)^\top s = g(\epsilon, \alpha)$$

$z = g(\epsilon, \alpha)$ is a deterministic function applied to stochastic noise.

Let's try applying this:

$$q(z_k = 1 \mid x; \lambda) = \frac{\alpha_k}{\sum_{j=1}^{K} \alpha_j} \qquad \alpha = \text{enc}(x; \lambda)$$

(Recalling $R_{\theta,\lambda}(z) = \log \frac{p(x, z; \theta)}{q(z \mid x; \lambda)}$),

$$\nabla_\lambda \mathbb{E}_{q(z \mid x; \lambda)}[R_{\theta,\lambda}(z)] = \nabla_\lambda \mathbb{E}_{\epsilon \sim \text{Gumbel}}[R_{\theta,\lambda}(g(\epsilon, \alpha))]$$

$$= \mathbb{E}_{\epsilon \sim \text{Gumbel}}[\nabla_\lambda R_{\theta,\lambda}(g(\epsilon, \alpha))]$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

Reparameterization:

$$z = \arg\max_{s \in \Delta^{K-1}} (\log \alpha + \epsilon)^\top s = g(\epsilon, \alpha)$$

$z = g(\epsilon, \alpha)$ is a deterministic function applied to stochastic noise.

Let's try applying this:

$$q(z_k = 1 \mid x; \lambda) = \frac{\alpha_k}{\sum_{j=1}^K \alpha_j} \qquad \alpha = \text{enc}(x; \lambda)$$

(Recalling $R_{\theta,\lambda}(z) = \log \frac{p(x,z;\theta)}{q(z \mid x; \lambda)}$),

$$\begin{aligned}\nabla_\lambda \mathbb{E}_{q(z \mid x; \lambda)}[R_{\theta,\lambda}(z)] &= \nabla_\lambda \mathbb{E}_{\epsilon \sim \text{Gumbel}}[R_{\theta,\lambda}(g(\epsilon, \alpha))] \\ &= \mathbb{E}_{\epsilon \sim \text{Gumbel}}[\nabla_\lambda R_{\theta,\lambda}(g(\epsilon, \alpha))]\end{aligned}$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

But this won't work, because zero gradients (almost everywhere)

$$z = g(\epsilon, \alpha) = \arg\max_{s \in \Delta^{K-1}} (\log \alpha + \epsilon)^\top s \implies \nabla_\lambda R_{\theta,\lambda}(z) = 0$$

Gumbel-Softmax trick: replace $\arg\max$ with $\mathrm{softmax}$

$$z = \mathrm{softmax}\left(\frac{\log \alpha + \epsilon}{\tau}\right) \qquad z_k = \frac{\exp((\log \alpha_k + \epsilon_k)/\tau)}{\sum_{j=1}^{K} \exp((\log \alpha_j + \epsilon_j)/\tau)}$$

(where τ is a temperature term.)

$$\nabla_\lambda \mathbb{E}_{q(z \mid x; \lambda)}[R_{\theta,\lambda}(z)] \approx \mathbb{E}_{\epsilon \sim \text{Gumbel}}\left[\nabla_\lambda R_{\theta,\lambda}\left(\mathrm{softmax}\left(\frac{\log \alpha + \epsilon}{\tau}\right)\right)\right]$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

But this won't work, because zero gradients (almost everywhere)

$$z = g(\epsilon, \alpha) = \arg\max_{s \in \Delta^{K-1}} (\log \alpha + \epsilon)^\top s \implies \nabla_\lambda R_{\theta,\lambda}(z) = 0$$

Gumbel-Softmax trick: replace $\arg\max$ with $\mathrm{softmax}$

$$z = \mathrm{softmax}\left(\frac{\log \alpha + \epsilon}{\tau}\right) \qquad z_k = \frac{\exp((\log \alpha_k + \epsilon_k)/\tau)}{\sum_{j=1}^{K} \exp((\log \alpha_j + \epsilon_j)/\tau)}$$

(where τ is a temperature term.)

$$\nabla_\lambda \mathbb{E}_{q(z \mid x; \lambda)}[R_{\theta,\lambda}(z)] \approx \mathbb{E}_{\epsilon \sim \text{Gumbel}}\left[\nabla_\lambda R_{\theta,\lambda}\left(\mathrm{softmax}\left(\frac{\log \alpha + \epsilon}{\tau}\right)\right)\right]$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

But this won't work, because zero gradients (almost everywhere)

$$z = g(\epsilon, \alpha) = \arg\max_{s \in \Delta^{K-1}} (\log \alpha + \epsilon)^\top s \implies \nabla_\lambda R_{\theta,\lambda}(z) = 0$$

Gumbel-Softmax trick: replace $\arg\max$ with $\mathrm{softmax}$

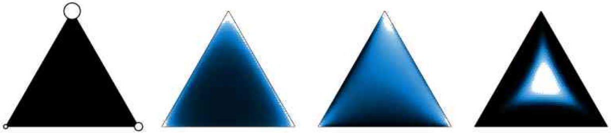
$$z = \mathrm{softmax}\left(\frac{\log \alpha + \epsilon}{\tau}\right) \qquad z_k = \frac{\exp((\log \alpha_k + \epsilon_k)/\tau)}{\sum_{j=1}^{K} \exp((\log \alpha_j + \epsilon_j)/\tau)}$$

(where τ is a temperature term.)

$$\nabla_\lambda \mathbb{E}_{q(z \mid x;\, \lambda)}[R_{\theta,\lambda}(z)] \approx \mathbb{E}_{\epsilon \sim \text{Gumbel}}\left[\nabla_\lambda R_{\theta,\lambda}\left(\mathrm{softmax}\left(\frac{\log \alpha + \epsilon}{\tau}\right)\right)\right]$$

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

- Approaches a discrete distribution as $\tau \to 0$ (anneal τ during training).
- Reparameterizable by construction
- Differentiable and has non-zero gradients



(a) $\lambda = 0$ (b) $\lambda = 1/2$ (c) $\lambda = 1$ (d) $\lambda = 2$

(from Maddison et al. [2017])

Gumbel-Softmax: Discrete Reparameterization [Jang et al. 2017; Maddison et al. 2017]

- See Maddison et al. [2017] on whether we can use the original categorical densities $p(z), q(z)$, or need to use relaxed densities $p_{\mathsf{GS}}(z), q_{\mathsf{GS}}(z)$.
- Requires that $p(x \mid z; \theta)$ "makes sense" for non-discrete z (e.g. attention).
- Lower-variance, but biased gradient estimator. Variance $\to \infty$ as $\tau \to 0$.

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Recall

$$\log p(x;\,\theta) = \mathrm{ELBO}(\theta, \lambda; x) - \mathrm{KL}[q(z \,|\, x;\, \lambda) \,\|\, p(z \,|\, x;\, \theta)]$$

Bound is tight when variational posterior equals true posterior

$$q(z \,|\, x;\, \lambda) = p(z \,|\, x;\, \theta) \implies \log p(x;\, \theta) = \mathrm{ELBO}(\theta, \lambda;\, x)$$

We want to make $q(z \,|\, x;\, \lambda)$ as flexible as possible: can we do better than just Gaussian?

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Recall

$$\log p(x;\, \theta) = \text{ELBO}(\theta, \lambda; x) - \text{KL}[q(z \,|\, x;\, \lambda) \,\|\, p(z \,|\, x;\, \theta)]$$

Bound is tight when variational posterior equals true posterior

$$q(z \,|\, x;\, \lambda) = p(z \,|\, x;\, \theta) \implies \log p(x;\, \theta) = \text{ELBO}(\theta, \lambda;\, x)$$

We want to make $q(z \,|\, x;\, \lambda)$ as flexible as possible: can we do better than just Gaussian?

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Idea: transform a sample from a simple initial variational distribution,

$$z_0 \sim q(z \mid x;\, \lambda) = \mathcal{N}(\mu, \sigma^2) \qquad \mu, \sigma^2 = \text{enc}(x;\, \lambda)$$

into a more complex one

$$z_K = f_K \circ \cdots \circ f_2 \circ f_1(z_0;\, \lambda)$$

where $f_i(z_{i-1};\, \lambda)$'s are **invertible** transformations (whose parameters are absorbed by λ).

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Idea: transform a sample from a simple initial variational distribution,

$$z_0 \sim q(z \,|\, x;\, \lambda) = \mathcal{N}(\mu, \sigma^2) \qquad \mu, \sigma^2 = \text{enc}(x;\, \lambda)$$

into a more complex one

$$z_K = f_K \circ \cdots \circ f_2 \circ f_1(z_0;\, \lambda)$$

where $f_i(z_{i-1};\, \lambda)$'s are **invertible** transformations (whose parameters are absorbed by λ).

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Sample from final variational posterior is given by z_K. Density is given by the change of variables formula:

$$\log q_K(z_K \mid x;\, \lambda) = \log q(z_0 \mid x;\, \lambda) + \sum_{k=1}^{K} \log \left| \frac{\partial f_k^{-1}}{\partial z_k} \right|$$

$$= \underbrace{\log q(z_0 \mid x;\, \lambda)}_{\text{log density of Gaussian}} - \sum_{k=1}^{K} \underbrace{\log \left| \frac{\partial f_k}{\partial z_{k-1}} \right|}_{\text{log determinant of Jacobian}}$$

Determinant calculation is $O(N^3)$ in general, but can be made faster depending on parameterization of f_k

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Can still use reparameterization to obtain gradients. Letting $F(z) = f_K \circ \cdots \circ f_1(z)$,

$$\begin{aligned}
\text{ELBO}(\theta, \lambda; x) &= \nabla_\lambda \mathbb{E}_{q_K(z_K \mid x;\lambda)}\left[\log \frac{p(x, z;\, \theta)}{q_K(z_K \mid x;\, \lambda)}\right] \\
&= \nabla_\lambda \mathbb{E}_{q(z_0 \mid x;\lambda)}\left[\log \frac{p(x, F(z_0);\, \theta)}{q(z_0 \mid x;\, \lambda)} - \log \left|\frac{\partial F}{\partial z_0}\right|\right] \\
&= \mathbb{E}_{\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}\left[\nabla_\lambda \left(\log \frac{p(x, F(z_0);\, \theta)}{q(z_0 \mid x;\, \lambda)} - \log \left|\frac{\partial F}{\partial z_0}\right|\right)\right]
\end{aligned}$$

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]

Examples of $f_k(z_{k-1};\, \lambda)$

- Normalizing Flows [Rezende and Mohamed 2015]

$$f_k(z_{k-1}) = z_{k-1} + u_k h(w_k^\top z_{k-1} + b_k)$$

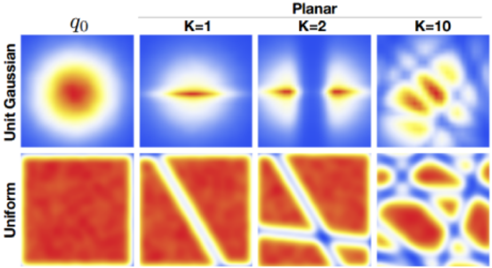
- Inverse Autoregressive Flows [Kingma et al. 2016]

$$f_k(z_{k-1}) = z_{k-1} \odot \sigma_k + \mu_k$$

$$\sigma_{k,d} = \text{sigmoid}(\text{NN}(z_{k-1,<d})) \qquad \mu_{k,d} = \text{NN}(z_{k-1,<d})$$

(In this case the Jacobian is upper triangular, so determinant is just the product of diagonals)

Flows [Rezende and Mohamed 2015; Kingma et al. 2016]



(from Rezende and Mohamed [2015])

Importance Weighted Autoencoder (IWAE) [Burda et al. 2015]

- Flows are a way of tightening the ELBO by making the variational family more flexible.
- Not the only way: can obtain a tighter lower bound on $\log p(x;\, \theta)$ by using multiple importance samples.

Consider:

$$I_K = \frac{1}{K} \sum_{k=1}^{K} \frac{p(x, z^{(k)};\, \theta)}{q(z^{(k)} \,|\, x;\, \lambda)},$$

where $z^{(1:K)} \sim \prod_{k=1}^{K} q(z^{(k)} \,|\, x;\, \lambda)$.

Note that I_K is an unbiased estimator of $p(x;\, \theta)$:

$$\mathbb{E}_{q(z^{(1:K)} \,|\, x;\, \lambda)} [I_K] = p(x;\, \theta).$$

Importance Weighted Autoencoder (IWAE) [Burda et al. 2015]

- Flows are a way of tightening the ELBO by making the variational family more flexible.
- Not the only way: can obtain a tighter lower bound on $\log p(x;\,\theta)$ by using multiple importance samples.

Consider:

$$I_K = \frac{1}{K}\sum_{k=1}^{K}\frac{p(x, z^{(k)};\,\theta)}{q(z^{(k)}\,|\,x;\,\lambda)},$$

where $z^{(1:K)} \sim \prod_{k=1}^{K} q(z^{(k)}\,|\,x;\,\lambda)$.

Note that I_K is an unbiased estimator of $p(x;\,\theta)$:

$$\mathbb{E}_{q(z^{(1:K)}\,|\,x;\,\lambda)}[I_K] = p(x;\,\theta).$$

Importance Weighted Autoencoder (IWAE) [Burda et al. 2015]

Any unbiased estimator of $p(x;\,\theta)$ can be used to obtain a lower bound, using Jensen's inequality:

$$p(x;\,\theta) = \mathbb{E}_{q(z^{(1:K)}\,|\,x;\lambda)}\left[I_K\right]$$

$$\implies \log p(x;\,\theta) \geq \mathbb{E}_{q(z^{(1:K)}\,|\,x;\lambda)}\left[\log I_K\right]$$

$$= \mathbb{E}_{q(z^{(1:K)}\,|\,x;\lambda)}\left[\log \frac{1}{K}\sum_{k=1}^{K}\frac{p(x, z^{(k)};\,\theta)}{q(z^{(k)}\,|\,x;\,\lambda)}\right]$$

However, can also show [Burda et al. 2015]:

- $\log p(x;\,\theta) \geq \mathbb{E}\left[\log I_K\right] \geq \mathbb{E}\left[\log I_{K-1}\right]$
- $\lim_{K\to\infty} \mathbb{E}\left[\log I_K\right] = \log p(x;\,\theta)$ under mild conditions

Importance Weighted Autoencoder (IWAE) [Burda et al. 2015]

$$\mathbb{E}_{q(z^{(1:K)} \mid x;\lambda)} \left[\log \frac{1}{K} \sum_{k=1}^{K} \frac{p(x, z^{(k)};\, \theta)}{q(z^{(k)} \mid x;\, \lambda)} \right]$$

- Note that with $K = 1$, we recover the ELBO.
- Can interpret $\frac{p(x, z^{(k)};\, \theta)}{q(z^{(k)} \mid x;\lambda)}$ as importance weights.
- If $q(z \mid x;\, \lambda)$ is reparameterizable, we can use the reparameterization trick to optimize $\mathbb{E}\left[\log I_K\right]$ directly.
- Otherwise, need score function gradient estimators [Mnih and Rezende 2016].

Sentence VAE Example [Bowman et al. 2016]

Generative Model (Model 2):

- Draw $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
- Draw $x_t \,|\, \mathbf{z} \sim \mathrm{CRNNLM}(\theta, \mathbf{z})$

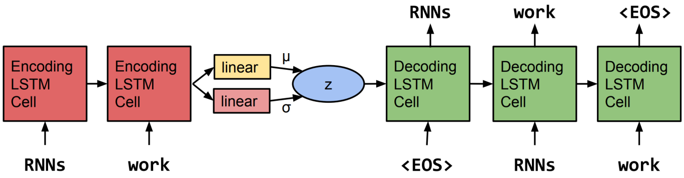
Variational Model (Amortized): Deep Diagonal Gaussians,

$$q(\mathbf{z} \,|\, x; \lambda) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^{\mathbf{2}})$$

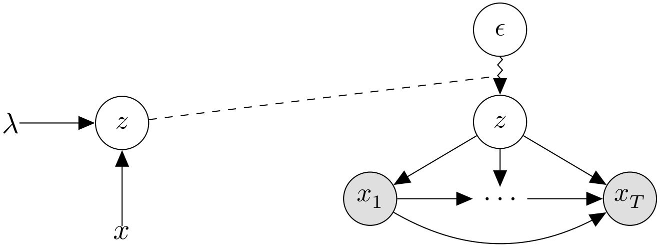
$$\tilde{\boldsymbol{h}}_T = \mathrm{RNN}(x; \psi)$$

$$\boldsymbol{\mu} = \mathbf{W}_1 \tilde{\boldsymbol{h}}_T \qquad \boldsymbol{\sigma}^{\mathbf{2}} = \exp(\mathbf{W}_2 \tilde{\boldsymbol{h}}_T) \quad \lambda = \{\mathbf{W}_1, \mathbf{W}_2, \psi\}$$

Sentence VAE Example [Bowman et al. 2016]



(from Bowman et al. [2016])



Issue 1: Posterior Collapse

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

$$= \underbrace{\mathbb{E}_{q(z \,|\, x;\, \lambda)}[\log p(x \,|\, z;\, \theta)]}_{\text{Reconstruction likelihood}} - \underbrace{\mathrm{KL}[q(z \,|\, x;\, \lambda) \| p(z)]}_{\text{Regularizer}}$$

Model	L/ELBO	Reconstruction	KL
RNN LM	-329.10	-	-
RNN VAE	-330.20	-330.19	0.01

(On Yahoo Corpus from Yang et al. [2017])

Issue 1: Posterior Collapse

- x and z become independent, and $p(x, z; \theta)$ reduces to a non-LV language model.
- Chen et al. [2017]: If it's possible to model $p_\star(x)$ without making use of z, then ELBO optimum is at:

$$p_\star(x) = p(x \,|\, z; \theta) = p(x; \theta) \qquad q(z \,|\, x; \lambda) = p(z)$$

$$\mathrm{KL}[q(z \,|\, x; \lambda) \| p(z)] = 0$$

Mitigating Posterior Collapse

Use less powerful likelihood models [Miao et al. 2016; Yang et al. 2017], or "word dropout" [Bowman et al. 2016].

Model	LL/ELBO	Reconstruction	KL
RNN LM	-329.1	-	-
RNN VAE	-330.2	-330.2	0.01
+ Word Drop	-334.2	-332.8	1.44
CNN VAE	-332.1	-322.1	10.0

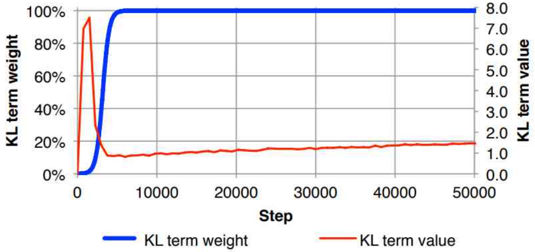
(On Yahoo Corpus from Yang et al. [2017])

Mitigating Posterior Collapse

Gradually anneal multiplier on KL term, i.e.

$$\mathbb{E}_{q(z \mid x;\, \lambda)}[\log p(x \mid z;\, \theta)] - \beta\, \mathrm{KL}[q(z \mid x;\, \lambda) \| p(z)]$$

β goes from 0 to 1 as training progresses



(from Bowman et al. [2016])

Mitigating Posterior Collapse

Other approaches:

- Use auxiliary losses (e.g. train z as part of a topic model) [Dieng et al. 2017; Wang et al. 2018]
- Use von Mises–Fisher distribution with a fixed concentration parameter [Guu et al. 2017; Xu and Durrett 2018]
- Combine stochastic/amortized variational inference [Kim et al. 2018]
- Add skip connections [Dieng et al. 2018]

In practice, often necessary to combine various methods.

Issue 2: Evaluation

- ELBO always lower bounds $\log p(x;\, \theta)$, so can calculate an upper bound on PPL efficiently.
- When reporting ELBO, should also separately report,

$$\mathrm{KL}[q(z \,|\, x;\, \lambda) \| p(z)]$$

to give an indication of how much the latent variable is being "used".

Issue 2: Evaluation

Also can evaluate $\log p(x;\, \theta)$ with importance sampling

$$p(x;\, \theta) = \mathbb{E}_{q(z \,|\, x;\, \lambda)} \left[\frac{p(x \,|\, z;\, \theta) p(z)}{q(z \,|\, x;\, \lambda)} \right]$$
$$\approx \frac{1}{K} \sum_{k=1}^{K} \frac{p(x|z^{(k)};\, \theta) p(z^{(k)})}{q(z^{(k)} \,|\, x;\, \lambda)}$$

So

$$\implies \log p(x;\, \theta) \approx \log \frac{1}{K} \sum_{k=1}^{K} \frac{p(x|z^{(k)};\, \theta) p(z^{(k)})}{q(z^{(k)} \,|\, x;\, \lambda)}$$

Evaluation

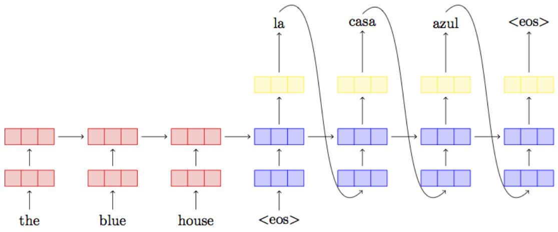
Qualitative evaluation

- Evaluate samples from prior/variational posterior.
- Interpolation in latent space.

i went to the store to buy some groceries .
i store to buy some groceries .
i were to buy any groceries .
horses are to buy any groceries .
horses are to buy any animal .
horses the favorite any animal .
horses the favorite favorite animal .
horses are my favorite animal .

(from Bowman et al. [2016])

Encoder/Decoder [Sutskever et al. 2014; Cho et al. 2014]



Given: Source information $s = s_1, \ldots, s_M$.

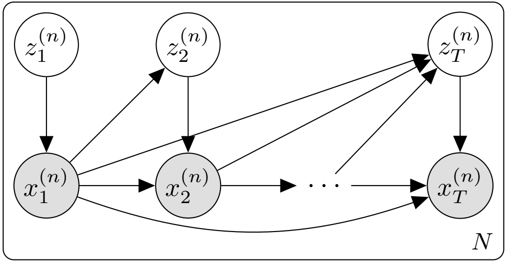
Generative process:

- Draw $x_{1:T} \mid s \sim \mathrm{CRNNLM}(\theta, \mathbf{enc}(s))$.

Latent, Per-token Experts [Yang et al. 2018]

Generative process: For $t = 1, \dots, T$,

- Draw $z_t \mid x_{<t}, s \sim \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h}_t)$.
- Draw $x_t \mid z_t, x_{<t}, s \sim \mathrm{softmax}(\boldsymbol{W} \tanh(\boldsymbol{Q}_{z_t}\boldsymbol{h}_t); \theta)$



If $\boldsymbol{U} \in \mathbb{R}^{K \times d}$, used K experts; increases the flexibility of per-token distribution.

Case-Study: Latent Per-token Experts [Yang et al. 2018]

Learning: z_t are independent given $x_{<t}$, so we can marginalize at each time-step (Method 3: Conjugacy).

$$\arg\max_{\theta} \log p(x \mid s; \theta) =$$

$$\arg\max_{\theta} \log \prod_{t=1}^{T} \sum_{k=1}^{K} p(z_t{=}k \mid s, x_{<t}; \theta)\, p(x_t \mid z_t{=}k, x_{<t}, s; \theta).$$

Test-time:

$$\arg\max_{x_{1:T}} \prod_{t=1}^{T} \sum_{k=1}^{K} p(z_t{=}k \mid s, x_{<t}; \theta)\, p(x_t \mid z_t{=}k, x_{<t}, s; \theta).$$

Case-Study: Latent, Per-token Experts [Yang et al. 2018]

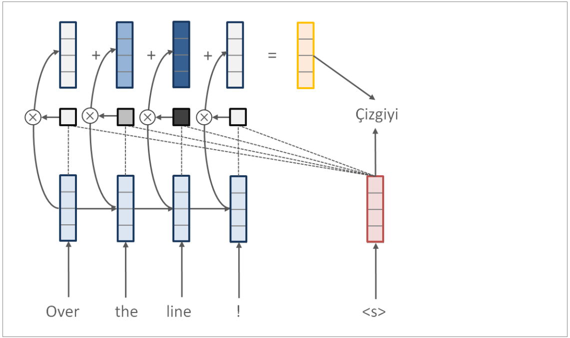
PTB language modeling results (s is constant):

Model	PPL
Merity et al. [2018]	57.30
Softmax-mixture [Yang et al. 2018]	54.44

Dialogue generation results (s is context):

Model	BLEU	
	Prec	Rec
No mixture	14.1	11.1
Softmax-mixture [Yang et al. 2018]	15.7	12.3

Attention [Bahdanau et al. 2015]



Decoding with an attention mechanism:

$$x_t \mid x_{<t}, s \sim \mathrm{softmax}(\boldsymbol{W}[\boldsymbol{h}_t, \sum_{m=1}^{M} \alpha_{t,m}\, \mathbf{enc}(s)_m]).$$

Copy Attention [Gu et al. 2016; Gulcehre et al. 2016]

Copy attention models copying words directly from s.

Generative process: For $t = 1, \dots, T$,

- Set $\boldsymbol{\alpha}_t$ to be attention weights.
- Draw $z_t \mid x_{<t}, s \sim \mathrm{Bern}(\mathrm{MLP}([\boldsymbol{h}_t, \mathbf{enc}(s)]))$.
- If $z_t = 0$
    - Draw $x_t \mid z_t, x_{<t}, s \sim \mathrm{softmax}(\boldsymbol{W}\boldsymbol{h}_t)$.
- Else
    - Draw $x_t \in \{s_1, \dots, s_M\} \mid z_t, x_{<t}, s \sim \mathrm{Cat}(\boldsymbol{\alpha}_t)$.

Copy Attention

Learning: Can maximize the log per-token marginal [Gu et al. 2016], as with per-token experts:

$$\max_{\theta} \log p(x_1, \dots, x_T \mid s; \theta)$$
$$= \max_{\theta} \log \prod_{t=1}^{T} \sum_{z' \in \{0,1\}} p(z_t = z' \mid s, x_{<t}; \theta)\, p(x_t \mid z', x_{<t}, x; \theta).$$

Test-time:

$$\arg\max_{x_{1:T}} \prod_{t=1}^{T} \sum_{z' \in \{0,1\}} p(z_t{=}z' \mid s, x_{<t}; \theta)\, p(x_t \mid z', x_{<t}, s; \theta).$$

Attention as a Latent Variable [Deng et al. 2018]

Generative process: For $t = 1, \dots, T$,

- Set $\boldsymbol{\alpha}_t$ to be attention weights.
- Draw $z_t \mid x_{<t}, s \sim \mathrm{Cat}(\boldsymbol{\alpha}_t)$.
- Draw $x_t \mid z_t, x_{<t}, s \sim \mathrm{softmax}(\boldsymbol{W}[\boldsymbol{h}_t, \mathbf{enc}(s_{z_t})]; \theta)$.

Attention as a Latent Variable [Deng et al. 2018]

Marginal likelihood under latent attention model:

$$p(x_{1:T} \mid s;\, \theta) = \prod_{t=1}^{T} \sum_{m=1}^{M} \alpha_{t,m} \, \mathrm{softmax}(\boldsymbol{W}[\boldsymbol{h}_t, \mathbf{enc}(s_m)];\, \theta)_{x_t}.$$

Standard attention likelihood:

$$p(x_{1:T} \mid s;\, \theta) = \prod_{t=1}^{T} \mathrm{softmax}(\boldsymbol{W}[\boldsymbol{h}_t, \sum_{m=1}^{M} \alpha_{t,m} \, \mathbf{enc}(s_m)];\, \theta)_{x_t}.$$

Attention as a Latent Variable [Deng et al. 2018]

Learning Strategy #1: Maximize the log marginal via enumeration as above.

Learning Strategy #2: Maximize the ELBO with AVI:

$$\max_{\lambda,\theta} \mathbb{E}_{q(z_t;\,\lambda)}\left[\log p(x_t \,|\, x_{<t}, z_t, s)\right] - \mathrm{KL}[q(z_t;\,\lambda) \| p(z_t \,|\, x_{<t}, s)].$$

- $q(z_t \,|\, x;\, \lambda)$ approximates $p(z_t \,|\, x_{1:T}, s;\, \theta)$; implemented with a BLSTM.
- q isn't reparameterizable, so gradients obtained using REINFORCE + baseline.

Attention as a Latent Variable [Deng et al. 2018]

Test-time: Calculate $p(x_t \mid x_{<t}, s; \theta)$ by summing out z_t.

MT Results on IWSLT-2014:

Model	PPL	BLEU
Standard Attn	7.03	32.31
Latent Attn (marginal)	6.33	33.08
Latent Attn (ELBO)	6.13	33.09

Encoder/Decoder with Structured Latent Variables

At least two EMNLP 2018 papers augment encoder/decoder text generation models with *structured* latent variables:

1. Lee et al. [2018] generate $x_{1:T}$ by iteratively refining sequences of words $z_{1:T}$.
2. Wiseman et al. [2018] generate $x_{1:T}$ conditioned on a latent template or plan $z_{1:S}$.

Summary as a Latent Variable [Miao and Blunsom 2016]

Generative process for a document $x = x_1, \dots, x_T$:

- Draw a latent summary $z_1, \dots, z_M \sim \text{RNNLM}(\theta)$
- Draw $x_1, \dots, x_T \mid z_{1:M} \sim \text{CRNNLM}(\theta, z)$

Posterior Inference:

$$p(z_{1:M} \mid x_{1:T}; \theta) = p(\text{summary} \mid \text{document}; \theta).$$

Summary as a Latent Variable [Miao and Blunsom 2016]

Generative process for a document $x = x_1, \dots, x_T$:

- Draw a latent summary $z_1, \dots, z_M \sim \mathrm{RNNLM}(\theta)$
- Draw $x_1, \dots, x_T \,|\, z_{1:M} \sim \mathrm{CRNNLM}(\theta, z)$

Posterior Inference:

$$p(z_{1:M} \,|\, x_{1:T};\, \theta) = p(\text{summary} \,|\, \text{document};\, \theta).$$

Summary as a Latent Variable [Miao and Blunsom 2016]

Learning: Maximize the ELBO with amortized family:

$$\max_{\lambda,\theta} \mathbb{E}_{q(z_{1:M};\,\lambda)}\left[\log p(x_{1:T} \mid z_{1:M};\,\theta)\right] - \mathrm{KL}[q(z_{1:M};\,\lambda) \| p(z_{1:M};\,\theta)]$$

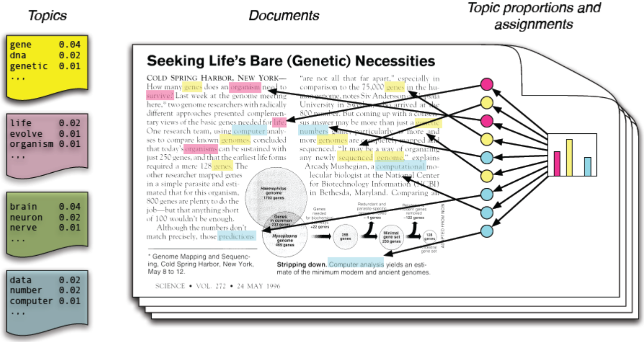
- $q(z_{1:M};\,\lambda)$ approximates $p(z_{1:M} \mid x_{1:T};\,\theta)$; also implemented with encoder/decoder RNNs.
- $q(z_{1:M};\,\lambda)$ not reparameterizable, so gradients use REINFORCE + baselines.

Summary as a Latent Variable [Miao and Blunsom 2016]

Semi-supervised Training: Can also use documents *without* corresponding summaries in training.

- Train $q(z_{1:M};\lambda) \approx p(z_{1:M} \mid x_{1:T};\theta)$ with labeled examples.
- Infer summary z for an *unlabeled* document with q.
- Use inferred z to improve model $p(x_{1:T} \mid z_{1:M};\theta)$.
- Allows for outperforming strictly supervised models!

Topic Models [Blei et al. 2003]



Generative process: for each document $x^{(n)} = x_1^{(n)}, \dots, x_T^{(n)}$,

- Draw topic distribution $\mathbf{z}_{top}^{(n)} \sim Dir(\boldsymbol{\alpha})$
- For $t = 1, \dots, T$:
  - Draw topic $z_t^{(n)} \sim Cat(\mathbf{z}_{top}^{(n)})$
  - Draw $x_t \sim Cat(\boldsymbol{\beta}_{z_t^{(n)}})$

Simple, Deep Topic Models [Miao et al. 2017]

Motivation: easy to learn deep topic models with VI if $q(\mathbf{z}_{top}^{(n)}; \lambda)$ is reparameterizable.

Idea: draw $\mathbf{z}_{top}^{(n)}$ from a transformation of a Gaussian.

- Draw $\mathbf{z}_0^{(n)} \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2)$
- Set $\mathbf{z}_{top}^{(n)} = \text{softmax}(\boldsymbol{W}\mathbf{z}_0^{(n)})$.
- Use analogous transformation when drawing from $q(\mathbf{z}_{top}^{(n)}; \lambda)$.

Simple, Deep Topic Models [Miao et al. 2017]

Learning Step #1: Marginalize out per-word latents $z_t^{(n)}$.

$$p(\{x^{(n)}\}_{n=1}^N, \{\mathbf{z}_{top}^{(n)}\}_{n=1}^N; \theta) = \prod_{n=1}^N p(\mathbf{z}_{top}^{(n)} \mid \theta) \prod_{t=1}^T \sum_{k=1}^K z_{top,k}^{(n)} \beta_{k,x_t^{(n)}}$$

Learning Step #2: Use AVI to optimize resulting ELBO.

$$\max_{\lambda,\theta} \mathbb{E}_{q(\mathbf{z}_{top}^{(n)}; \lambda)} \left[\log p(x^{(n)} \mid \mathbf{z}_{top}^{(n)}; \theta)\right] - \mathrm{KL}[\mathcal{N}(\mathbf{z}_0^{(n)}; \lambda) \| \mathcal{N}(\mathbf{z}_0^{(n)}; \boldsymbol{\mu}_0, \boldsymbol{\sigma}_0^2)]$$

Simple, Deep Topic Models [Miao et al. 2017]

Perplexities on held-out documents, for three datasets:

Model	MXM	20News	RCV1
OnlineLDA [Hoffman et al. 2010]	342	1015	1058
AVI-LDA [Miao et al. 2017]	272	830	602

Deep Latent-Variable NLP: Two Views

Deep Models & LV Models are naturally complementary:

- Rich set of model choices: discrete, continuous, and structured.
- Real applications across NLP including some state-of-the-art models.

Deep Models & LV Models are frustratingly incompatible:

- Many interesting approaches to the problem: reparameterization, score-function, and more.
- Lots of area for research into improved approaches.

Deep Latent-Variable NLP: Two Views

Deep Models & LV Models are naturally complementary:

- Rich set of model choices: discrete, continuous, and structured.
- Real applications across NLP including some state-of-the-art models.

Deep Models & LV Models are frustratingly incompatible:

- Many interesting approaches to the problem: reparameterization, score-function, and more.
- Lots of area for research into improved approaches.

Implementation

- Modern toolkits make it easy to implement these models.
- Combine the flexibility of auto-differentiation for optimization (PyTorch) with distribution and VI libraries (Pyro).

In fact, we have implemented this entire tutorial. See website link:
http://bit.do/lvnlp

Implementation

- Modern toolkits make it easy to implement these models.
- Combine the flexibility of auto-differentiation for optimization (PyTorch) with distribution and VI libraries (Pyro).

In fact, we have implemented this entire tutorial. See website link:

`http://bit.do/lvnlp`

Charu C. Aggarwal and ChengXiang Zhai. 2012. A Survey of Text Clustering Algorithms. In *Mining Text Data*, pages 77–128. Springer.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *Proceedings of ICLR*.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyal, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating Sentences from a Continuous Space. In *Proceedings of CoNLL*.

Peter F Brown, Peter V Desouza, Robert L Mercer, Vincent J Della Pietra, and Jenifer C Lai. 1992. Class-based N-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. 2015. Importance Weighted Autoencoders. In *Proceedings of ICLR*.

Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2017. Variational Lossy Autoencoder. In *Proceedings of ICLR*.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In *Proceedings of the Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38.

Yuntian Deng, Yoon Kim, Justin Chiu, Demi Guo, and Alexander M. Rush. 2018. Latent Alignment and Variational Attention. In *Proceedings of NIPS*.

Adji B. Dieng, Yoon Kim, Alexander M. Rush, and David M. Blei. 2018. Avoiding Latent Variable Collapse with Generative Skip Models. In *Proceedings of the ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*.

Adji B. Dieng, Chong Wang, Jianfeng Gao, and John Paisley. 2017. TopicRNN: A Recurrent Neural Network With Long-Range Semantic Dependency. In *Proceedings of ICLR*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12.

Jason Eisner. 2016. Inside-Outside and Forward-Backward Algorithms Are Just Backprop (Tutorial Paper). In *Proceedings of the Workshop on Structured Prediction for NLP*.

Ekaterina Garmash and Christof Monz. 2016. Ensemble Learning for Multi-source Neural Machine Translation. In *Proceedings of COLING*.

Zoubin Ghahramani and Michael I. Jordan. 1996. Factorial Hidden Markov Models. In *Proceedings of NIPS*.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1631–1640.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 140–149.

Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. 2017. Generating Sentences by Editing Prototypes. *arXiv:1709.08878*.

William P Headden III, Mark Johnson, and David McClosky. 2009. Improving Unsupervised Dependency Parsing with Richer Contexts and Smoothing. In *Proceedings of NAACL*.

Marti A Hearst. 1997. Texttiling: Segmenting text into multi-paragraph subtopic passages. *Computational linguistics*, 23(1):33–64.

Matthew Hoffman, Francis R Bach, and David M Blei. 2010. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. 2017. Toward Controlled Generation of Text. In *Proceedings of ICML*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive Mixtures of Local Experts. *Neural Computation*, 3(1):79–87.

Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical Reparameterization with Gumbel-Softmax. In *Proceedings of ICLR*.

Yoon Kim, Sam Wiseman, Andrew C. Miller, David Sontag, and Alexander M. Rush. 2018. Semi-Amortized Variational Autoencoders. In *Proceedings of ICML*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of ICLR*.

Diederik P. Kingma, Tim Salimans, and Max Welling. 2016. Improving Variational Inference with Autoregressive Flow. *arXiv:1606.04934*.

Diederik P. Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *Proceedings of ICLR*.

Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought Vectors. In *Proceedings of NIPS*.

Dan Klein and Christopher D Manning. 2004. Corpus-based Induction of Syntactic Structure: Models of Dependency and Constituency. In *Proceedings of ACL*.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proceedings of EMNLP*.

Stefan Lee, Senthil Purushwalkam Shiva Prakash, Michael Cogswell, Viresh Ranjan, David Crandall, and Dhruv Batra. 2016. Stochastic Multiple Choice Learning for Training Diverse Deep Ensembles. In *Proceedings of NIPS*.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *Proceedings of ICLR*.

Bernard Merialdo. 1994. Tagging English Text with a Probabilistic Model. *Computational Linguistics*, 20(2):155–171.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2018. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*.

Yishu Miao and Phil Blunsom. 2016. Language as a Latent Variable: Discrete Generative Models for Sentence Compression. In *Proceedings of EMNLP*.

Yishu Miao, Edward Grefenstette, and Phil Blunsom. 2017. Discovering Discrete Latent Topics with Neural Variational Inference. In *Proceedings of ICML*.

Yishu Miao, Lei Yu, and Phil Blunsom. 2016. Neural Variational Inference for Text Processing. In *Proceedings of ICML*.

Andriy Mnih and Danilo J. Rezende. 2016. Variational Inference for Monte Carlo Objectives. In *Proceedings of ICML*.

Andryi Mnih and Karol Gregor. 2014. Neural Variational Inference and Learning in Belief Networks. In *Proceedings of ICML*.

Anjan Nepal and Alexander Yates. 2013. Factorial Hidden Markov Models for Learning Representations of Natural Language. *arXiv:1312.6168*.

George Papandreou and Alan L. Yuille. 2011. Perturb-and-Map Random Fields: Using Discrete Optimization to Learn and Sample from Energy Models. In *Proceedings of ICCV*.

Danilo J. Rezende and Shakir Mohamed. 2015. Variational Inference with Normalizing Flows. In *Proceedings of ICML*.

Noah A. Smith and Jason Eisner. 2005. Contrastive Estimation: Training Log-Linear Models on Unlabeled Data. In *Proceedings of ACL*.

Ilya Sutskever, Oriol Vinyals, and Quoc Le. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of NIPS*.

Ke Tran, Yonatan Bisk, Ashish Vaswani, Daniel Marcu, and Kevin Knight. 2016. Unsupervised Neural Hidden Markov Models. In *Proceedings of the Workshop on Structured Prediction for NLP*.

Stephan Vogel, Hermann Ney, and Christoph Tillmann. 1996. Hmm-based word alignment in statistical translation. In *Proceedings of the 16th conference on Computational linguistics-Volume 2*, pages 836–841. Association for Computational Linguistics.

Wenlin Wang, Zhe Gan, Wenqi Wang, Dinghan Shen, Jiaji Huang, Wei Ping, Sanjeev Satheesh, and Lawrence Carin. 2018. Topic Compositional Neural Language Model. In *Proceedings of AISTATS*.

Peter Willett. 1988. Recent Trends in Hierarchic Document Clustering: A Critical Review. *Information Processing & Management*, 24(5):577–597.

Ronald J. Williams. 1992. Simple Statistical Gradient-following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, 8.

Sam Wiseman, Stuart M. Shieber, and Alexander M. Rush. 2018. Learning Neural Templates for Text Generation. In *Proceedings of EMNLP*.

Jiacheng Xu and Greg Durrett. 2018. Spherical Latent Spaces for Stable Variational Autoencoders. In *Proceedings of EMNLP*.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. 2018. Breaking the Softmax Bottleneck: A High-Rank RNN Language Model. In *Proceedings of ICLR*.

Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. 2017. Improved Variational Autoencoders for Text Modeling using Dilated Convolutions. In *Proceedings of ICML*.